Airline Control System Version 2

Concepts and Facilities

Release 4.1

SH19-6953-13

Airline Control System Version 2

Concepts and Facilities

Release 4.1

SH19-6953-13

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xiii.

Fourteenth Edition (June 2008)

This edition applies to Release 4, Modification Level 1, of Airline Control System Version 2, Program Number 5695-068, and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers’ comments appears at the back of this publication. If the form has been removed, address your comments to:

ALCS Development
2455 South Road
P923
Poughkeepsie NY 12601-5400
USA

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

The Director of Licencing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department 830A
522 South Road
Mail Drop P131
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Programming interface information

This Concepts and Facilities manual is intended to help the customer to understand the functions and facilities of Airline Control System Version 2, Program Number 5695-068. This Concepts and Facilities manual documents General-Use Programming Interface and Associated Guidance Information provided by Airline Control System Version 2 Program Number 5695-068.

General-Use programming interfaces allow the customer to write programs that obtain the services of Airline Control System Version 2

Trademarks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States or other countries or both:

AIX	APPN	CICS	CUA
DB2	DFSMS	DFSMS/MVS	DFSMSdfp
DFSMSdss	DFSMShsm	DFSMSrmm	DFSORT
ESA/390	ESCON	IBM	IMS
MQSeries	MVS	MVS/DFP	MVS/ESA
MVS/XA	NetView	PR/SM	RACF
RMF	S/370	S/390	SAA
SNA	System/370	System/390	VisualAge
VTAM	WebSphere	z/Architecture	z/OS
zSeries			
Language Environment		OpenEdition	
Parallel Sysplex		Processor Resource/Systems Manager	
Resource Measurement Facility		Systems Application Architecture	

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About this book

This book presents conceptual information for Release 4.1 of Airline Control System (ALCS) Version 2, an IBM* licensed program.

ALCS is one of a family of IBM programs designed to satisfy the needs of airlines and other industries with similar requirements for high-volume and high-availability transaction processing.

The product, which is also known as TPF/MVS, provides the Transaction Processing Facility (TPF) application programming interface (API) for z/OS* environments. It supersedes ALCS/Multiple Virtual Storage/Extended Architecture (ALCS/MVS/XA*), known as ALCS Version 1.

Throughout this book:

- Airline Control System Version 2 is abbreviated to ALCS unless the context makes it necessary to distinguish between ALCS Version 2 Release 4.1, and the predecessor products.
- Airlines Line Control Interconnection (ALCI) includes the function of network extension facility (NEF).
- Advanced Communications Function for the Virtual Telecommunication Method is abbreviated to VTAM*.
- TPF refers to all versions of Transaction Processing Facility and its predecessor, Airlines Control Program (ACP).
- MVS* refers to z/OS.

This book describes the overall function of ALCS and how ALCS uses MVS and VTAM services. It explains the concept of ALCS resources and how ALCS maps its database layout on to VSAM clusters. It is also used as a link to all the other manuals in the library.

Who should read this book

This book is intended to help system programmers, application programmers and operators to gain an overall understanding of the functions and facilities of ALCS.

It may also be useful to executives and data processing professionals wishing to choose a transaction processing system.

How this book is organized

This book is organized as follows:

Chapter 1, “ALCS Version 2 concepts and facilities”
Provides an overview of ALCS communication and database facilities and describes message flow in an ALCS system. It also outlines multiprogramming and multiprocessing.

Chapter 2, “Communication management”
Describes the communication protocols that ALCS supports.

Chapter 3, “ALCS data sharing and data management”
Describes how ALCS shares data with other systems.

Chapter 4, "ALCS database file management"
Describes the ALCS direct-access files and the addressing methods that ALCS uses. It also describes lists structures and index structures.

Chapter 5, "Sequential file management"
Describes the sequential files that ALCS provides for applications to use, and the sequential files that ALCS uses for system purposes.

Chapter 6, "Entry management"
Describes how ALCS creates and dispatches entries.

Chapter 7, "Storage management"
Describes how ALCS uses the storage space associated with the MVS region where ALCS runs. In particular it describes the use of home address space.

Chapter 8, "Automated operations"
Describes how NetView can be used to provide automated-operations support for the MVS environment.

Appendix A, "ALCS pool file support"
Gives more details about type 1 and type 2 pool support.

Appendix B, "Long-term pool space recovery – Recoup"
Gives more detail about chain structures, and how to describe them for Recoup.

Appendix C, "Communication management for the SLC network"
Describes SLC links and procedures.

Appendix D, "ALCS services"
Lists the services that ALCS provides for applications.

Appendix E, "Direct-access files"
Gives more detail about the ALCS direct-access files.

Appendix F, "Application global area"
Explains the layout of the ALCS application global area.

Appendix G, "Application program management"
Introduces the concept of the ALCS application program load list and the program configuration table

Appendix H, "Acronyms and abbreviations"
Lists acronyms and abbreviations used throughout the ALCS library. Not every term necessarily occurs in this book.

The book also contains a glossary of terms, a bibliography, and an index.

Chapter 1. ALCS Version 2 concepts and facilities

This chapter provides an overview of ALCS and describes how the online monitor processes input and output messages. It also outlines multiprogramming and multiprocessing in an ALCS environment.

1.1 Overview of ALCS

ALCS is a software interface between application programs and the z/OS operating system. It runs as a job or started task under z/OS, providing real-time transaction processing facilities for airlines, banks, hotels, and other industries that generate high transaction rates and require fast response times and high system availability.

Typical applications are passenger and cargo reservations for airlines and railroads, hotel booking systems, and credit card authorization.

ALCS provides:

- High performance and capacity
- A high level of system availability
- High transaction rates
- A wide range of communication facilities
- Connectivity with other transaction processing platforms
- Access to relational databases for business applications

Figure 1 shows a simplified view of an ALCS system where user terminals connect to the ALCS online system. ALCS application programs (running on the online system) handle specific tasks associated with transaction processing.

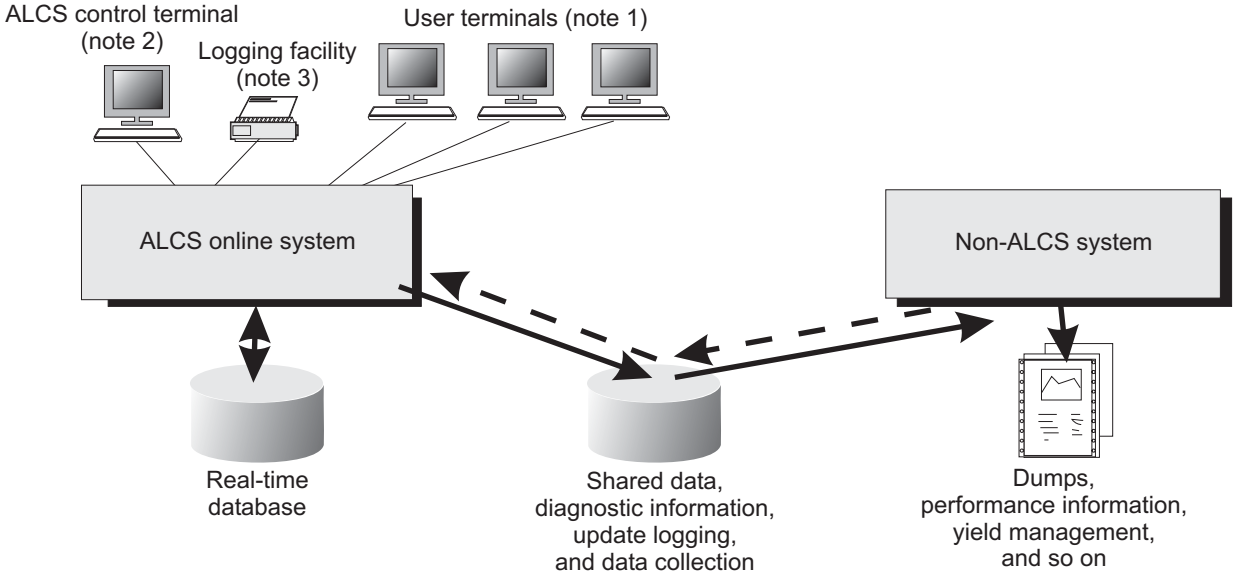


Figure 1. ALCS components overview

Note: ALCS terminals are known traditionally as:

1. Agent sets
2. Computer-room agent set (CRAS)
3. Read-only computer-room agent set (RO CRAS)

The real-time database is optimized for high-speed access; the offline processing uses separate shared files. The ALCS system is controlled and monitored from special terminals in secure areas.

The application programming interface (API) of ALCS V2 is compatible with the API of the TPF family:

- ALCS/MVS/XA (also called ALCS V1)
- ALCS/VSE
- TPF and its predecessor ACP

Applications developed for any of the four can be – and are – easily exchanged with users of the other three. You can install proven applications with minimal programming effort.

ALCS Version 2 supports higher transaction rates than ALCS/MVS/XA or ALCS/VSE because it benefits from the increased capacity and function of the z/OS operating system, which allows the use of more powerful hardware configurations.

1.2 General description of ALCS Version 2

ALCS runs in a z/OS environment, and uses many of the facilities of MVS to perform standard data processing operations.

ALCS Version 2 is a **control monitor** which supports programs written to the specifications of the application program interface for TPF, ALCS/VSE, and ALCS/MVS/XA.

ALCS V2 also supports a wide variety of communication protocols for different networks and terminals. Figure 2 on page 3 shows the communication between ALCS and the following:

- Applications
- VTAM and non-VTAM communications resources
- NetView*
- Other processors and programs
- DB2* for z/OS
- WebSphere* MQ for z/OS
- TCP/IP

Appendix H, “Acronyms and abbreviations” on page 167 lists the abbreviations used in Figure 2 on page 3.

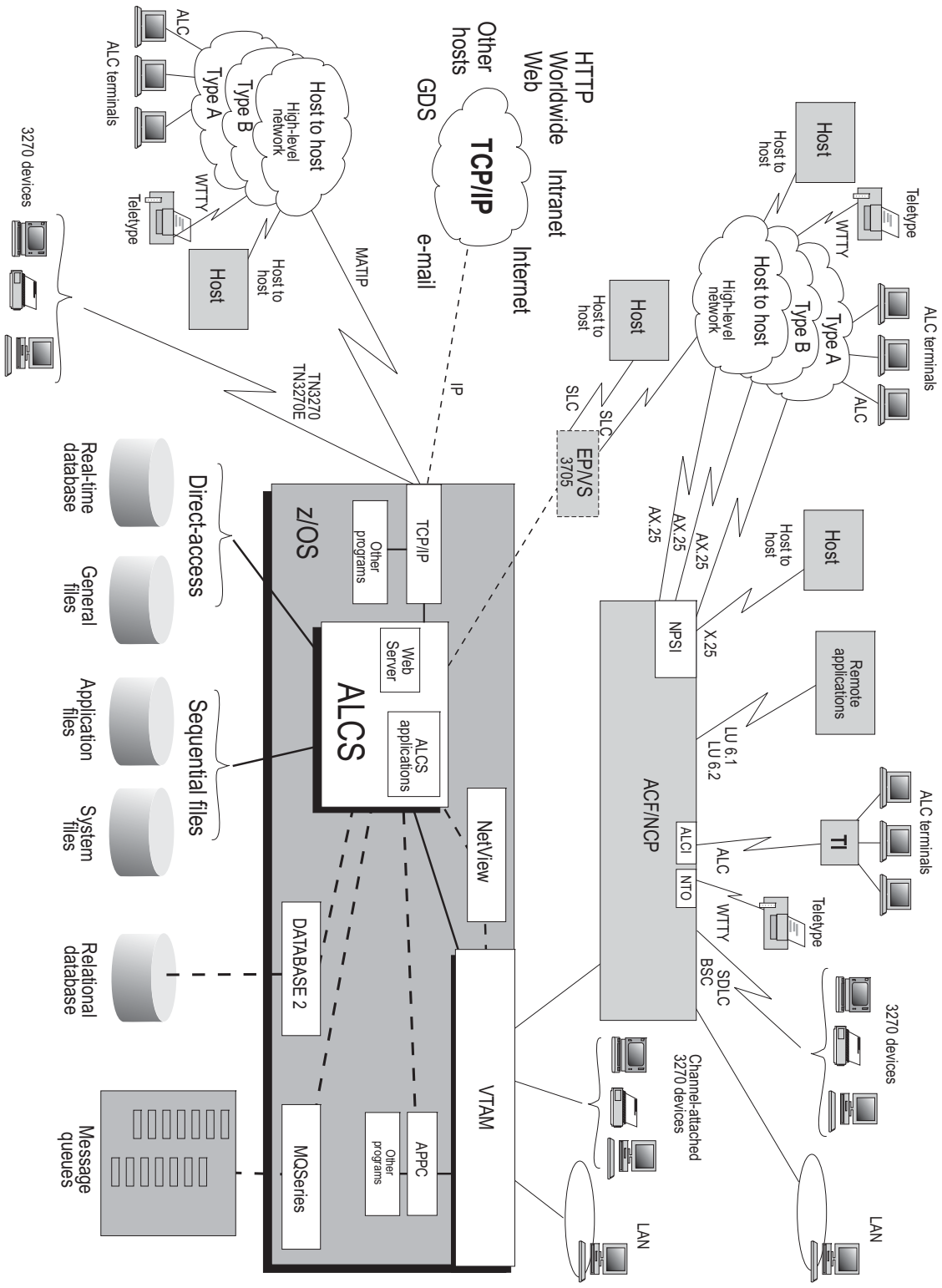


Figure 2. Communication and database facilities

In addition to the control monitor, ALCS includes the following components to generate, maintain, and operate the system:

- Interactive System Productivity Facility (ISPF) panels to simplify:
  - Installation
  - Generation
  - Maintenance
  - Application program assembly or compilation
- Assembler language macrodefinitions that application programs can use. These macros replace the equivalent TPF or ALCS/VSE macrodefinitions.
- C language header files and functions that application programs written in the C language can use.
- Callable services that assembler and high-level language programs can use
- A generation package that includes assembler language macrodefinitions. The ALCS user codes a series of macroinstructions that specify installation-specific characteristics of the system, and then assembles these statements. The assembly generates the configuration-dependent components of the ALCS system.
- A support package, executing under the control of the ALCS online monitor, that provides interactive monitoring, control, and diagnostic facilities additional to those provided with MVS and VTAM.
- A maintenance package that includes MVS offline programs. These programs perform functions such as producing reports.
- Components that provide access to NetView, which allows the use of automated operations.

Figure 3 shows where you can find more information about ALCS components.

Figure 3. Where to find more information about ALCS components

Component	
Application files	5.2, “Application sequential files” on page 96
Database	Chapter 4, “ALCS database file management” on page 61
General files	4.2, “General files and general data sets” on page 69
Message flow	1.5, “Message flow in an ALCS system” on page 13
Real-time database	4.1, “The ALCS real-time database” on page 62
Sharing data	Chapter 3, “ALCS data sharing and data management” on page 55
System files	5.1, “System sequential files” on page 94

1.3 Application programming languages

ALCS allows you to write application programs in several languages. These languages fall into the following main categories:

- Assembler
- SabreTalk
- C
- Other high-level languages (HLLs)

Figure 4 shows the four main categories.

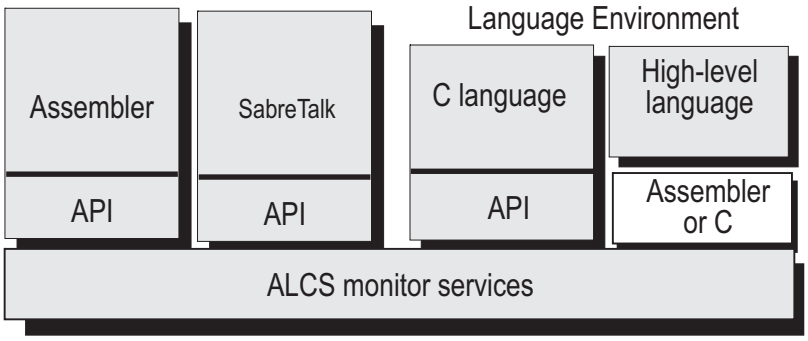


Figure 4. ALCS monitor services

Figure 5 shows the application program interfaces available with ALCS.

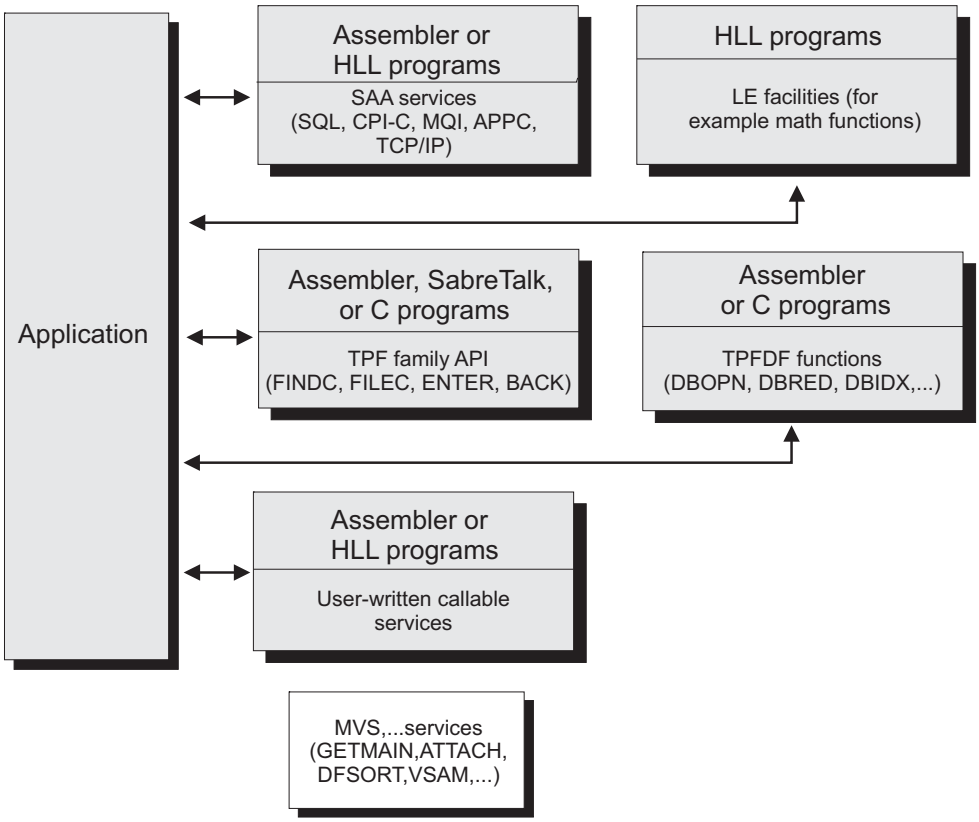


Figure 5. Summary of ALCS application program interfaces

Notes:

1. Not all LE facilities can be used in ALCS application programs
2. Not all TPF family API functions are available to C application programs
3. Do **not** use MVS services in ALCS application programs

Figure 6 on page 6 and Figure 7 on page 7 summarize the characteristics of the languages in these categories.

ALCS Application Programming Guide describes how to choose a language for application programming.

Figure 6. ALCS application programming language

Existing applications	There are large numbers of existing application programs that you may want to use on your ALCS system. It is often cheaper to buy an existing application (and possibly customize or enhance it) than to develop a new one. ALCS Version 2 supports all the programming languages that other TPF-family platforms support.
Performance	The choice of programming language can affect the performance characteristics of the application. Usually, assembler language programs have shorter pathlengths than high-level language programs. This is unlikely to affect transaction response times, but it can affect the size (power) of the processor you need.
Portability	Assembler language programs are easily portable between TPF-family platforms, but difficult to port to or from other platforms. Similarly, SabreTalk and C language programs are easily portable between TPF-family platforms. Porting applications between other (not TPF-family) platforms and ALCS is generally easier if the applications are in a high-level language.
Services	ALCS supports the following programming interfaces: **SQL** For accessing relational data bases. **CPI-C** For synchronous (conversational) communication with other applications. ALCS also supports the related SNA programming interface, **APPC**. **MQI** For asynchronous (queued) communication with other applications via WebSphere MQ for z/OS. **TCP/IP** For communication with other applications and devices.
Monitor services and TPFDF	ALCS provides a wide range of specialized monitor services for assembler and SabreTalk language programs, and a more restricted range for C language programs. The IBM TPFDF product provides an access method for application programs on TPF-family platforms. TPFDF provides services for assembler and C language programs.
Restrictions	These restrictions are different for the different languages, but always include: • ALCS application programs must use ALCS services, TPFDF services, or other interfaces to perform I/O. • More generally, ALCS application programs must not directly invoke services provided by MVS or other MVS subsystems. These restrictions also apply to any programs that your application calls.

Figure 7 (Page 1 of 2). Other application programming languages

	Assembler	SabreTalk	C language	Other HLL
Existing applications	Most existing TPF-family applications are in assembler language. You can run these programs under ALCS with little or no program changes.	Some existing TPF-family applications are in SabreTalk. You can run these programs under ALCS with little or no program changes. There are no SabreTalk applications for other platforms.	Some existing TPF-family applications are in C. You can run these programs under ALCS with little or no program changes.	There are no existing TPF-family applications written in languages other than assembler, SabreTalk, and C.
Performance	Provided that equally efficient algorithms are used, assembler language gives the best possible performance (shortest pathlength).	Typically SabreTalk applications have pathlengths that are longer than assembler applications, but shorter than other high-level languages.	The performance of C language programs is similar to other HLL programs.	Usually, HLL programs have a longer pathlength than assembler language programs. This is unlikely to affect transaction response times, but it can affect the size (power) of the processor you need.
Portability	Easily portable between TPF-family platforms, but difficult to port to or from other platforms.	Easily portable between TPF-family platforms, but not portable to other platforms.	Applications that use the TPF-family API functions (`entrc`, `findc`, and so on) are portable only to and from other TPF-family platforms.	With careful design, it is possible to develop programs in high-level languages that are portable to and from other platforms. These programs must adhere to ALCS language restrictions.
Other services	Assembler application programs can use SQL, CPI-C, APPC, MQI and TCP/IP calls.	Other services are not available to SabreTalk programs.	C application programs can use SQL, CPI-C, APPC, MQI and TCP/IP calls.	Depends if the HLL is supported by: • DB2 for z/OS precompiler (for SQL) • APPC/MVS (for CPI-C and APPC) • WebSphere MQ for z/OS (for MQI) • TCP/IP for MVS or Communication Server (for TCP/IP)
Monitor services and TPFDF	All ALCS monitor services and TPFDF services are available to assembler application programs.	Most ALCS monitor services are available to SabreTalk programs. SabreTalk programs cannot access TPFDF facilities.	Most ALCS monitor services and TPFDF services are available to C programs. You must avoid using these services if want your programs to be portable to other (not TPF-family) platforms.	You can develop your own callable services to make selected ALCS monitor services or TPFDF services available to your HLL programs

Figure 7 (Page 2 of 2). Other application programming languages

	Assembler	SabreTalk	C language	Other HLL
Restrictions	There are a number of restrictions that apply to ALCS assembler language application programming. Refer to *ALCS Application Programming Reference – Assembler* for a complete list of restrictions.	SabreTalk is a special purpose TPF-family language. ALCS does not impose any restrictions on SabreTalk.	Broadly similar to other transaction processing platforms (such as CICS*). The main restriction is that you must not use C I/O functions (except for `stdin` and `stdout`).	Broadly similar to other transaction processing platforms (such as CICS). The main restriction is that you must not use I/O functions.

1.3.1 Callable services for high-level language programs

ALCS provides a wide range of specialized monitor services for assembler and SabreTalk language programs, and a more restricted range for C language programs.

IBM's TPF Database Facility (TPFDF) product provides an access method for application programs on TPF-family platforms. TPFDF provides services for assembler and C language programs.

Programs in other languages cannot access these specialized services directly. Instead, you must develop callable services to interface between these programs and the ALCS environment, including your existing application (if any).

By developing callable services, you can make your high-level language programs independent of the unique characteristics of the ALCS environment. This greatly simplifies porting the applications to or from other platforms. It also reduces the need for ALCS-specific programming skills in high-level language application programmers.

Reasons for choosing to write application programs in high-level languages often include:

- Marketability
- Portability
- Skills availability

If your application code directly invokes specialized ALCS monitor functions then you reduce all these benefits to some extent. Callable services are (usually small) programs that you write in C or assembler language. Your application programs invoke these callable services using the `CALL` interface. The callable service routines interface directly with the data base and existing application programs using ALCS monitor services. Figure 8 shows this schematically.

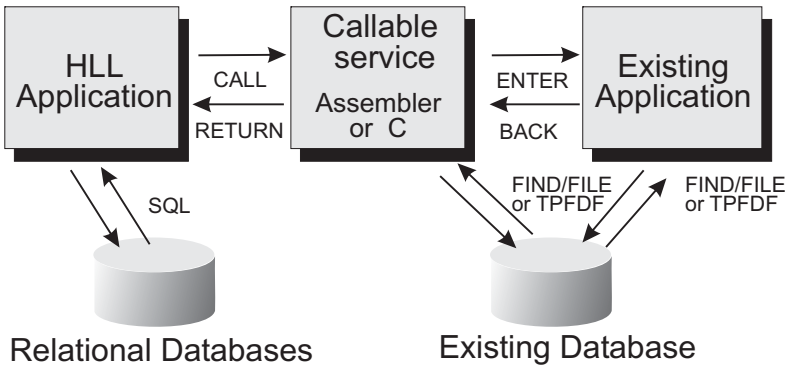


Figure 8. Callable services overview

Note that the purpose of the callable service program in Figure 8 is to **isolate** the application program from application and system functions that may be different on other systems.

ALCS Application Programming Guide describes how to develop callable services for high-level language programs.

7.5, “High-level language storage” on page 114 describes the storage requirements for HLL programs.

1.4 Overview of the ALCS Version 2 system

This section describes the following:

- Application environment
- ALCS commands
- Offline programs
- Generation macros

1.4.1 Application environment

The primary function of an ALCS system is to process messages received from the communication network. ALCS processes each input message as a separate user work item. These work items are called **entries**. Associated with each entry is an **entry control block (ECB)** that is used to control the processing of that entry. 1.7.2, “Entry control block” on page 35 describes the ALCS ECB fields in more detail.

Notes:

1. ALCS entries are not the same as MVS tasks.
2. An ALCS ECB is not the same as an MVS event control block.
3. 1.7, “Multiprogramming and multiprocessing” on page 34 explains how ALCS uses these entries.

When ALCS receives an input message from a communication terminal, the online monitor creates an entry. It then transfers control to the application program which performs the requested function.

The application program normally constructs and sends a response message and then **exits** the entry. Typically, an entry completes processing and exits within a fraction of second.

In addition to input messages, two other types of entry can exist:

- Entries that application programs generate. ALCS provides monitor services that allow application programs to create new entries. These are the **create-type** services.
- Entries that ALCS generates. For example, the ALCS online monitor timer routines create a new entry every minute. The new entry is for application-dependent timer functions.

During the processing of a message, the application programs request the monitor to perform services such as:

- Transferring control between application programs
- Obtaining and returning storage areas
- Initiating input/output operations

Application programs use ALCS application programming interface (API) functions to call the ALCS routines that perform these services, these are:

- Monitor-request macros (for assembler programs)
- C language functions (for C programs)

The monitor can perform many of these services immediately, and can often return control directly to the requesting application. However, for some requests, the monitor cannot return control to the requesting application until some other event completes. For example, in a request to read a record from a direct access storage device (DASD), ALCS cannot return control until the data transfer from DASD to processor storage completes. If ALCS cannot return control immediately, the processing of that entry is suspended.

ALCS saves information so that it can restart the entry. When the other event completes, the suspended processing resumes. In this way, ALCS implements its own subtasking system.

ALCS supports **multiprocessor configurations** by processing multiple entries simultaneously. To ensure the integrity of shared storage areas, ALCS provides facilities to prevent programs that update the same storage area from running simultaneously. These are called **serialization services**.

1.4.2 ALCS commands

ALCS includes a set of commands (variously called **operator commands**, **functional messages**, and **Z messages**) to control its operation. ALCS processes these commands in much the same way as it processes messages that request application functions. The first character of all ALCS commands is `Z`; this is called the **primary action code**. The commands are five characters long and can be followed by parameters.

ALCS commands request such functions as:

- Alter the contents of an ALCS file
- Display information about ALCS communication resources

To prevent unauthorized access, the use of some ALCS commands is restricted. Typical restrictions are:

- The command is only accepted from the Prime CRAS terminal
- The command is only accepted from CRAS terminals
- The command is only accepted while ALCS is in a particular system state

1.4.3 Offline programs

In addition to the online monitor and the application programs, the ALCS system includes a number of independent MVS batch programs. These are called **offline programs**. Offline programs run as normal MVS batch jobs and use standard MVS services.

Examples of ALCS offline programs include:

ALCS system test compiler (STC)
You can use STC to create data records and message records.

STC compiles data on to a sequential data set, called a **data file**, for loading to the real-time database. Input to STC is a user-coded file that describes the contents of real-time database records. Use the `ZDATA` command to load the records from the data file to the real-time database.

You can also use STC to prepare a sequential data set, called a **test unit tape (TUT)**, that contains test messages. The TUT is input to the online monitor test routines. It allows the testing of application functions.

ALCS diagnostic file processor
The diagnostic file processor analyzes the contents of diagnostic files generated by ALCS, and prints formatted reports.

1.4.4 Generation macros

The ALCS monitor program and some of the offline programs need information that varies from one installation to another, including:

- The size of the real-time database
- The real-time and general sequential files that the application needs
- A description of the ALCS communication network

The configuration-dependent tables contain this information. Each table is a separate load module.

You must code the ALCS generation macroinstructions to define the initial ALCS configuration, or to define a change to the ALCS configuration.

Stage 1 of the ALCS generation procedure assembles the macroinstructions to create an MVS job stream.

Stage 2 executes the job stream produced by stage 1 and creates the configuration-dependent tables.

1.5 Message flow in an ALCS system

This section describes the following:

- Entry of a message from a terminal
- Input message processing by the ALCS online monitor
- Messages on TCP/IP
- Messages on WebSphere MQ for z/OS
- Application program processing
- Output message processing by the ALCS online monitor

1.5.1 Entry of a message from a terminal

Figure 2 on page 3 shows the communication networks available to ALCS. Messages can originate from various types of terminal:

- IBM 3270 terminals on a VTAM network
- ALC terminals on a VTAM network (ALCI or AX.25)
- IBM 3270 terminals on a TCP/IP network
- ALC terminals on a TCP/IP network
- Terminals on an SLC high-level network
- NetView operator IDs

IBM 3270 terminals on a VTAM network

An input message consists of one or more lines of text. Pressing the Enter key transmits the message to VTAM. ALCS uses VTAM `RECEIVE` macros to obtain the message from VTAM.

At various stages in the routing of the message to ALCS, control information is added to the basic text of the message. This control information includes:

- Codes that indicate the position of the input message on the display terminal screen.
- Routing codes that indicate the address of the originating terminal.

ALC terminals on a VTAM network (ALCI or AX.25)

Input messages are entered in much the same way as from IBM 3270 terminals connected by VTAM. The main differences are:

- Terminals connected in this way are normally dedicated to a particular system. All input messages are sent to that system.
- The control and routing information that is added during transmission of the message to ALCS is different from that added to IBM 3270 messages.

IBM 3270 terminals on a TCP/IP network

Telnet for IBM 3270 terminals is a service provided by Communication Server in z/OS. Telnet sessions using the TN3270 or TN3270E protocols appear as standard 3270 devices in ALCS.

Input messages are entered in much the same way as from IBM 3270 terminals connected by VTAM. The main differences are:

- Pressing the Enter key transmits the message to the TCP/IP Telnet server. The Telnet server runs a VTAM application which has a session with ALCS. ALCS receives the message from this VTAM application as if it were from a 3270 terminal.
- The control and routing information that is added during transmission of the message to ALCS is different from that added to VTAM messages.

ALC terminals on a TCP/IP network

Input messages are entered in much the same way as from ALC terminals connected by VTAM. The main differences are:

- Terminals connected in this way are normally dedicated to a particular system. All input messages are sent to that system.
- VTAM does not process the messages. Instead, ALCS uses TCP/IP sockets calls to obtain the message from TCP/IP.
- The control and routing information that is added during transmission of the message to ALCS is different from that added to VTAM messages.

Terminals on an SLC high-level network

Input messages are entered in much the same way as from terminals connected by VTAM. The main differences are:

- Terminals connected in this way are normally dedicated to a particular system. All input messages are sent to that system.
- VTAM does not process the messages. Instead, the ALCS online monitor performs I/O operations directly to the communication controller (for example, an IBM 3705) that connects the communication lines from the high-level network.
- The control and routing information that is added during transmission of the message to ALCS is different from that added to VTAM messages.

1.5.2 Input message processing by the ALCS online monitor

ALCS creates an **entry** for each incoming message. The entry control block (ECB) contains 16 **storage levels** (level D0 through level DF) that can associate up to 16 blocks of storage with the entry. ALCS places incoming messages in a storage block associated with level D0.

IBM 3270 Terminals on a VTAM or TCP/IP network

The ALCS online monitor uses VTAM `RECEIVE` macros to get input messages from VTAM. When the ALCS job starts, the online monitor restart routines issue a number of `RECEIVE`s. Each `RECEIVE` specifies an input buffer. When VTAM receives an input message, it puts the message in the buffer and indicates that the `RECEIVE` has completed.

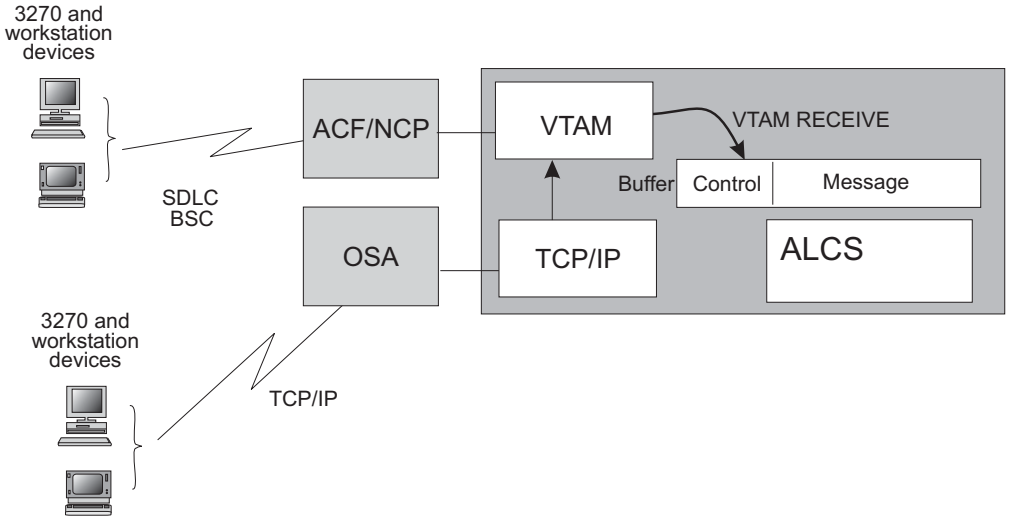


Figure 9. Input message processing: VTAM puts the message in a buffer

As each RECEIVE completes, the ALCS online monitor transfers the input message into a storage block attached to level D0 of the ECB and reissues the RECEIVE so that VTAM can pass another message.

When the online monitor transfers the message into a storage block, it reformats the message text into the format that ALCS applications use. It transforms the terminal address information into an internal format terminal address called a **communication resource identifier (CRI)** and stores the CRI in a **routing control parameter list** (RCPL) and in the message header on level D0.

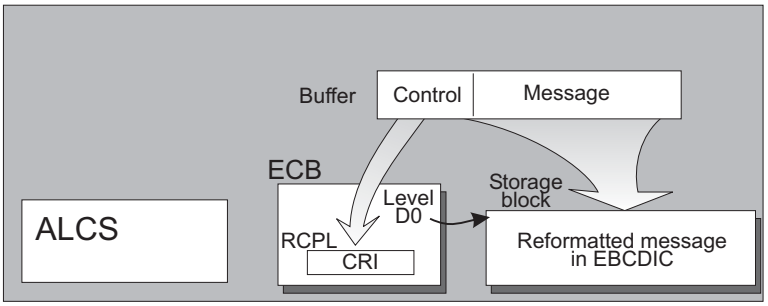


Figure 10. Input message processing: ALCS transfers the data to a storage block

The ECB that contains the reformatted input message is then added to a list of outstanding work items for the ALCS online monitor. This work list is called the **input list**.

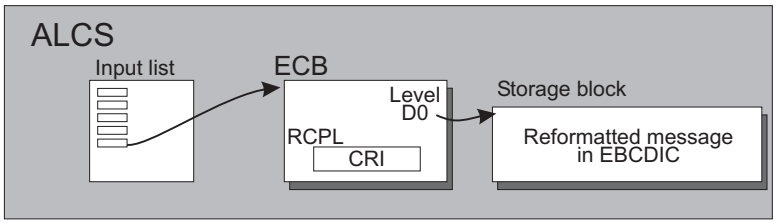


Figure 11. Input message processing: ALCS adds the ECB to the input list

When the ALCS online monitor can process a new task, it removes the ECB from the input list.

An online monitor routine determines which ALCS application processes the message, and passes control to that application.

ALC terminals on a VTAM network (ALCI or AX.25)

The ALCS online monitor processes input messages in much the same way as for other terminals connected by VTAM. It also translates the message text to the EBCDIC code that ALCS applications use.

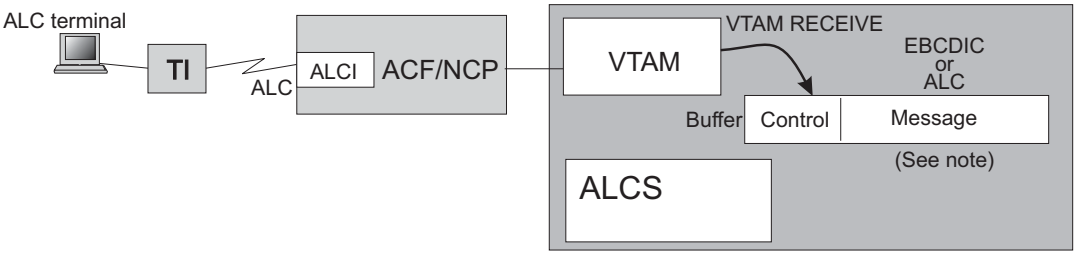


Figure 12. Input message processing: ALCS transfers the data to a storage block and translates it to EBCDIC

Note: Either ALCS or NCP can translate the message from ALC to EBCDIC.

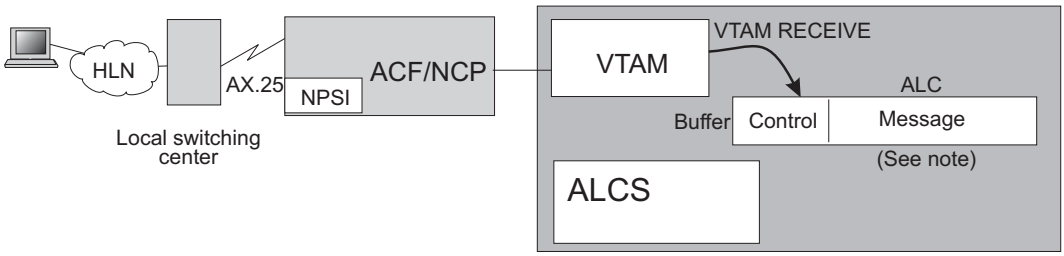


Figure 13. Input message processing: ALCS transfers the data to a storage block and translates it to EBCDIC

Note: ALCS translates the message from ALC to EBCDIC.

The ECB that contains the reformatted input message is then added to the input list, and processing continues as for terminals on a VTAM network.

ALC terminals on a TCP/IP network

The ALCS online monitor uses `SELECTX` socket calls to detect when data is ready to be received on TCP/IP connections, and it uses `RECV` socket calls to get input data from TCP/IP. Each RECV socket call specifies an input buffer where TCP/IP puts the data. These input buffers are storage areas allocated by the ALCS TCP/IP communication initialization routines.

Once a complete input message has been received, the ALCS online monitor transfers the input message into a storage block attached to level D0 of the ECB. It also translates the text to EBCDIC and reformats the message text into the format that ALCS applications use.

The ECB that contains the reformatted input message is then added to the input list, and processing continues as for terminals on a VTAM network.

As well as TCP/IP connections for terminal traffic, you can define TCP/IP connections to remote applications. ALCS communication routines support both client-type connections and server-type connections.

ALCS supports the following MATIP message traffic for a TCP/IP high-level network:

- ATA/IATA Type A, conversational message traffic
- ATA/IATA Type B, conventional message traffic
- Host-to-host traffic

Terminals on an SLC high-level network

ALCS reads these input messages directly from the communication controller.

When a channel of an SLC link is started, ALCS initiates input from the communication controller by issuing the MVS `EXCP` macro. It uses a continuously running channel program that transfers data into a set of input buffers. These input buffers are storage areas allocated by the ALCS SLC communication initialization routines. Since the channel program runs continuously, there is no interruption to indicate when a message has been read in. Instead, the online monitor checks the input buffers at intervals of approximately 200 milliseconds.

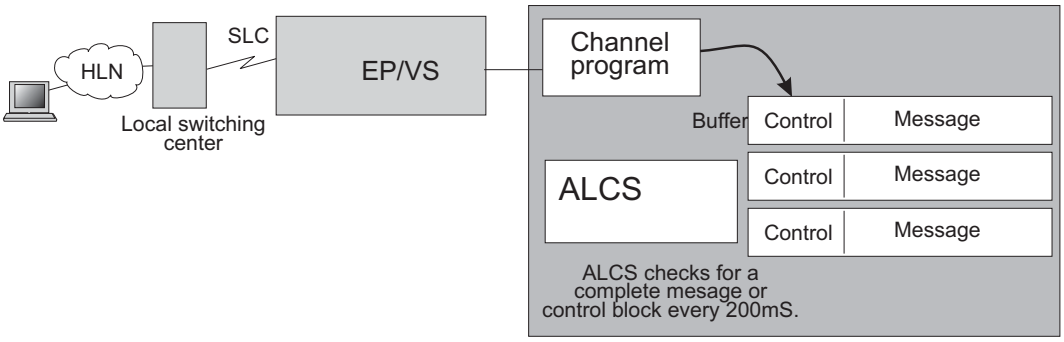


Figure 14. Input message processing: A channel program fills the input buffers

If one or more of the buffers contains a complete message, the online monitor transfers each input message into a storage block attached at level D0 of an ECB.

When ALCS transfers the message into a storage block, it translates the text to EBCDIC and reformats the message text into the format that ALCS applications use. ALCS processes control blocks and does not pass them as messages to applications.

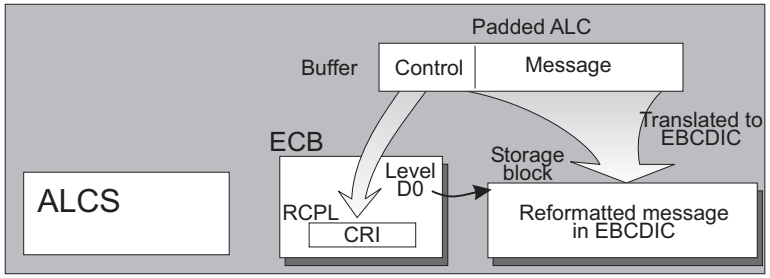


Figure 15. Input message processing: ALCS transfers the data to a storage block and translates it to EBCDIC

The ECB that contains the reformatted input message is then added to the input list, and processing continues as for terminals connected by VTAM.

ALCS supports the following SLC message traffic:

- ATA/IATA Type A, conversational message traffic
- ATA/IATA Type B, conventional message traffic
- Host-to-host traffic

1.5.3 Messages on TCP/IP

To use ALCS TCP/IP communication support, you must have Communication Server IP installed together with the relevant hardware and software to connect MVS to the TCP/IP network.

You can use TCP/IP calls in your ALCS applications to exchange messages with other ALCS applications or applications that are:

- On the same MVS system
- On another platform that supports TCP/IP socket connections.

Figure 2 on page 3 shows how ALCS can connect to remote devices using the TCP/IP protocols.

The ALCS system programmer uses the SCTGEN macro parameters to specify:

- Support for TCP/IP (TCPIP=YES).
- Support for the TCP/IP concurrent server (Listener) (TCPLIST=YES).
- The virtual IP address for binding sockets used by the concurrent server and the TCP/IP communication resources (TCPVIPA=*virtual_ip_address*)
- The name of the TCP/IP address space for the inital connection (TCPNAME=*tcpip_address_space_name*).
- Up to eight TCP/IP port numbers which the concurrent server will use (TCPPORT={*port_number_1*,*port_number_2*,...}).

The ALCS operator uses the ZCTCP command to establish or terminate a connection between ALCS and a specified TCP/IP address space in the same MVS system. The ZCTCP command also starts and stops the ALCS TCP/IP concurrent server (ALCS Listener). Up to eight concurrent servers (Listeners) can be started at the same time using different port numbers.

ALCS Operation and Maintenance describes the ZCTCP command.

E-mail

You can use the ALCS e-mail facility to transmit and receive e-mail messages over TCP/IP networks using an external mail server or mail transfer agent (MTA). ALCS uses the Internet Simple Message Transfer Protocol (SMTP) to communicate with the mail server. E-mail uses the ALCS support for socket programming and the ALCS concurrent server (Listener).

Web Server

You can use the ALCS Web Server facility to deliver web pages and application function to Web Browser clients over TCP/IP networks. ALCS uses the Internet HTTP protocol to communicate with the clients. The Web Server uses ALCS support for socket programming and the ALCS concurrent server (Listener), as well as the ALCS hierarchical file system (HFS).

TCP/IP large messages

ALCS supports large (up to 2 MegaBytes) extended messages to/from ALC terminals connected to ALCS through TCP/IP and to/from TCP/IP connections.

ALCS will use heap storage for input messages, if a message does not fit in any storage block.

As such messages can be very large (up to 2MB) there are certain restrictions (only solicited messages to terminals connected to this ALCS, no ALCS scroll logging, no ALCS message retrieval using the ZRETR command).

1.5.4 Messages on WebSphere MQ for z/OS

You can use MQI calls in your ALCS applications to exchange messages with other ALCS applications or applications that are:

- On the same MVS system
- On another platform that supports message queuing using the IBM WebSphere MQ products

In message queuing with ALCS, the term application queue denotes any queue on which application programs use MQI calls to **put** and **get** messages. In addition to the MQI calls which the IBM WebSphere MQ for z/OS product supports, your application can take advantage of two special message queuing facilities that ALCS provides.

These facilities are the:

- ALCS initiation queue
- ALCS input queue

The ALCS initiation queue

In message queuing, an initiation queue is a local queue on which the queue manager puts trigger messages. The queue manager creates a trigger message on the initiation queue when a predetermined event occurs, such as a message arriving on an application queue. A trigger message contains information about which program is to be started, in order to handle this event.

You can define an WebSphere MQ for z/OS initiation queue to ALCS in order to start an ALCS application automatically when a trigger message is put on this queue. ALCS uses information (in the trigger message) to start an ALCS application that can retrieve messages from the application queue to which the trigger message refers.

You can use the ALCS initiation queue facility when you want to monitor an WebSphere MQ for z/OS initiation queue, but you do not want to create a long-lived ECB to do this.

The ALCS system programmer can use the SCTGEN macro parameters to specify:

- Support for WebSphere MQ for z/OS: MQM=YES
- The name of an WebSphere MQ for z/OS initiation queue for ALCS and, optionally, the lowest system state in which ALCS can open the initiation queue: MQMI=(*initiation_queue*,*system_state*)

ALCS Installation and Customization describes the SCTGEN macro.

ALCS can open an initiation queue during startup (MQM=(YES,CONNECT)), or the ALCS operator can use the ZCMQI command to open this (or another) initiation queue.

(*ALCS Operation and Maintenance* describes the ZCMQI command.)

Figure 16 through Figure 19 give an overview of the ALCS initiation queue.

Initial state: Figure 16 on page 21 shows the initial state where the application queue is empty.

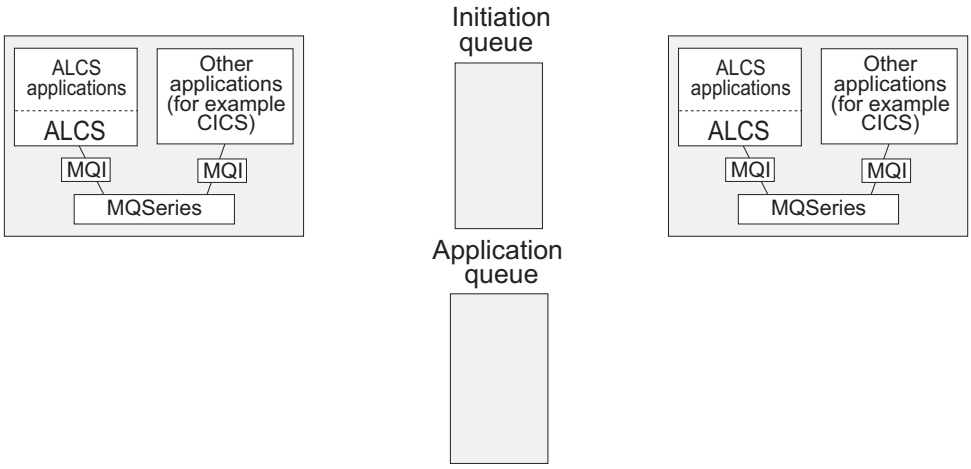


Figure 16. Input message processing: Initial state, application queue empty

Figure 17 shows some messages on the application queue

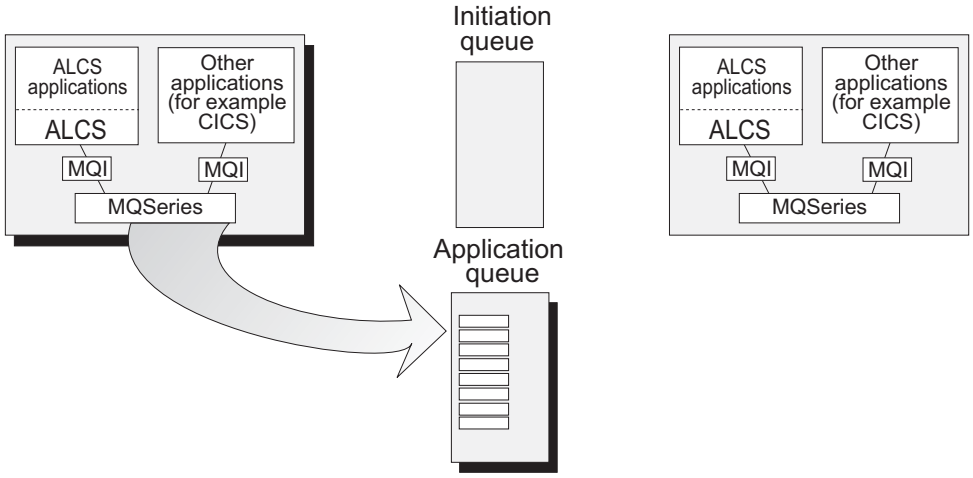


Figure 17. Input message processing: The queue becomes non-empty

WebSphere MQ for z/OS puts a trigger message on the initiation queue to indicate the transition of the application queue from empty to non-empty. Figure 18 shows this trigger message.

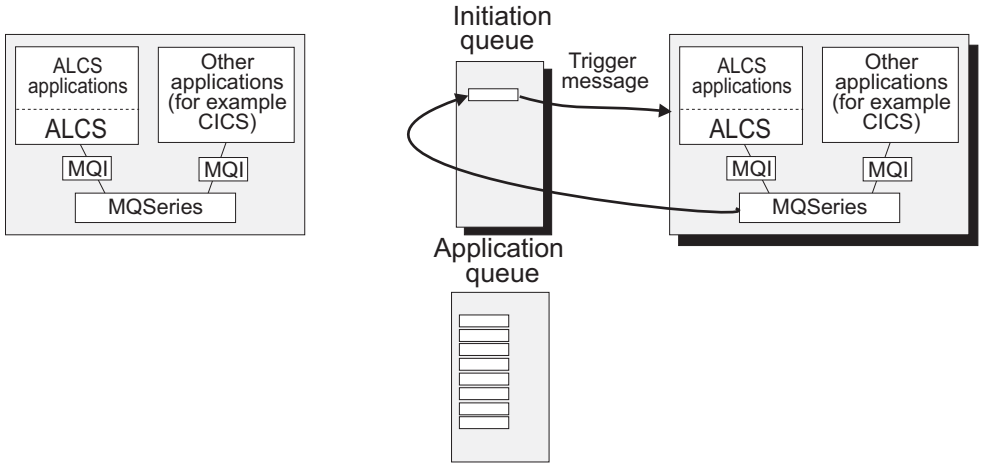


Figure 18. Input message processing: WebSphere MQ for z/OS puts a trigger message on the initiation queue

Figure 19 shows how ALCS transfers the message to storage block and attaches the block to ECB level DO.

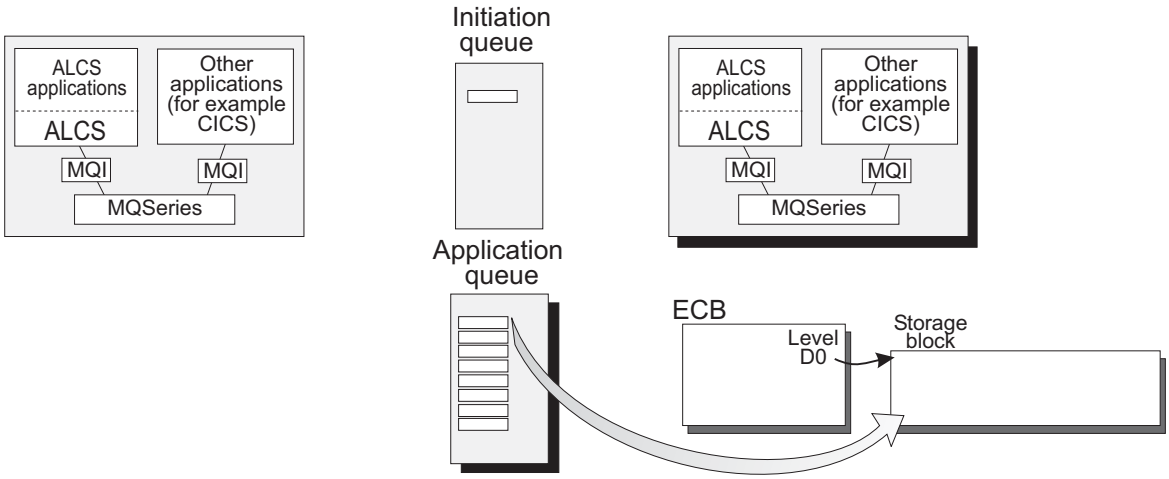


Figure 19. Input message processing: ALCS gets a message and transfers it to a storage block

The ALCS input queue

You can define a local queue to ALCS in order to start an ALCS application automatically when a message is put on this queue.

Messages on the ALCS input queue must be in PPMSG message format. In a PPMSG format message, a routing control parameter list (RCPL) precedes the message text. ALCS uses information in the RCPL to start an ALCS application that can retrieve the message from the application queue.

You can use the ALCS input queue facility when you want to monitor an WebSphere MQ for z/OS local queue, but you do not want to create a long-lived ECB to do this. The ALCS system programmer can use the `SCTGEN` macro parameters to specify:

- Support for WebSphere MQ for z/OS: `MQM=YES`
- The name of an WebSphere MQ for z/OS input queue for ALCS and, optionally, the lowest system state in which ALCS can open the input queue: `MQMQ=`(*input_queue*,*system_state*)

ALCS Installation and Customization describes the `SCTGEN` macro.

ALCS can open an input queue during startup (`MQM=YES,CONNECT`), or the ALCS operator can use the `ZCMQI` command to open this (or another) input queue.

ALCS Operation and Maintenance describes the `ZCMQI` command.

Figure 20 shows how ALCS transfers the message to a storage block and attaches the block to ECB level DO.

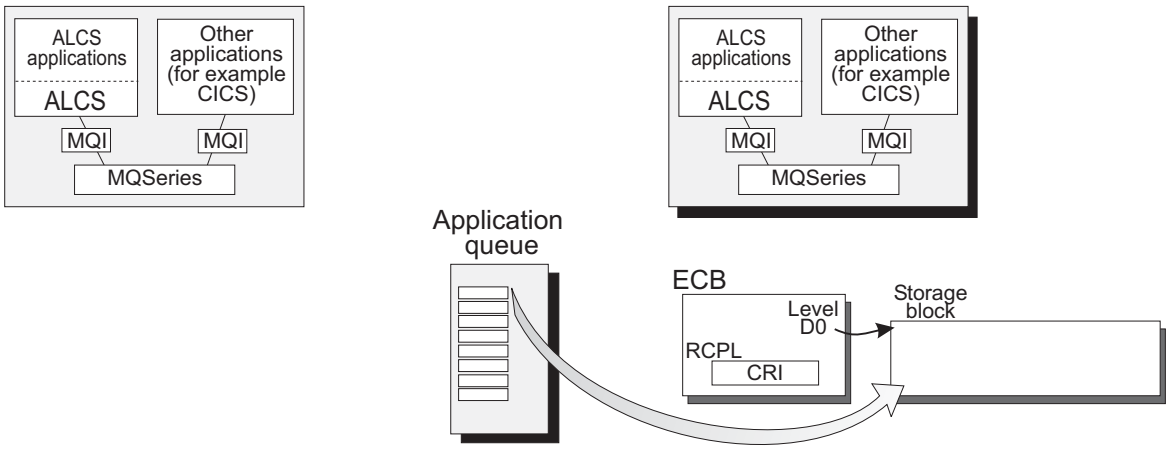


Figure 20. Input message processing: The ALCS input queue

MQ Bridge

The ALCS MQ Bridge allows messages received on MQ request queues to be formatted and passed on to legacy applications as if they came from ordinary terminal devices.

Output from the applications is routed from the ALCS output routines to the MQ Bridge in order to send it on to the corresponding MQ response queues.

The MQ queues are defined in the ALCS communication table. *MQ terminals* are also defined in the ALCS communication table and they are associated with an MQ queue definition. Applications can use normal terminal records and device addressing.

This support is intended to provide connectivity to current ALCS applications from remote systems (for example, web servers).

1.5.5 Application program processing

Input message routing

When ALCS receives an input message, it creates a new entry to process the message. Then the ALCS router passes control to an application program called an **input-message editor**. There is an input-message editor for each application, but they do not have to be unique; several applications can use the same input-message editor.

The ALCS router checks whether there is input routing for the terminal. If there is, it routes the input message to the corresponding input message editor. If there is not, it routes the input message to the ALCS command processor.

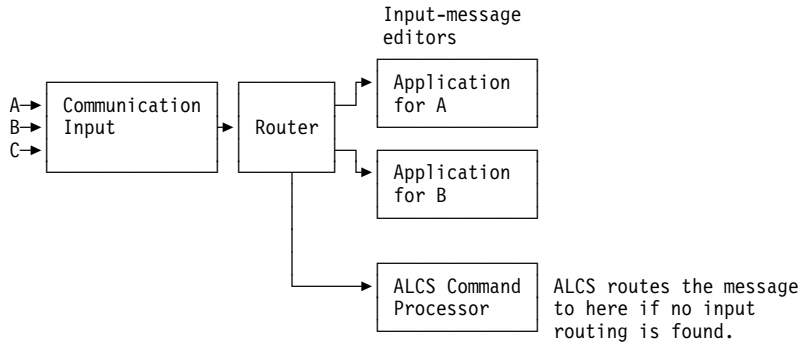


Figure 21. Input message processing: ALCS checks the routing for a message

The input-message routing for a terminal can be specified by:

- The COMDEF macro in the ALCS communication generation
- The ZROUT command
- The ZACOM command

Using program function (PF) keys to enter messages

If a terminal has program function (PF) keys, you can use these to enter messages. ALCS automatically converts the resulting input into normal input messages or commands.

ALCS provides standard conversion for PF keys (which is the same as ALCS/MVS/XA). You can provide an installation-wide exit to override these PF key settings, for example to adhere to IBM's Common User Access* (CUA)* recommendations. *ALCS Installation and Customization* describes the ALCS installation-wide exits.

Also, end users can use the ZAKEY command to customize their own PF key settings. This is described in *ALCS Operation and Maintenance*.

Input messages beginning with Z

ALCS normally assumes that any input message starting with Z is an ALCS command (or possibly a user command added using an installation-wide exit) and passes the message directly to the ALCS command processor.

You can override this default processing if your **application** expects some input messages to begin with Z. For example, a check-in application might require a passenger name as an input message. If you do this, your application input-message editor must identify messages that really are ALCS commands and pass them on to the ALCS command processor (otherwise the end user will not be able to use the ALCS commands).

Figure 22 on page 26 shows a possible routing arrangement for an airline application system. The system has two applications, reservations and message switching. Each application has its own input-message editor.

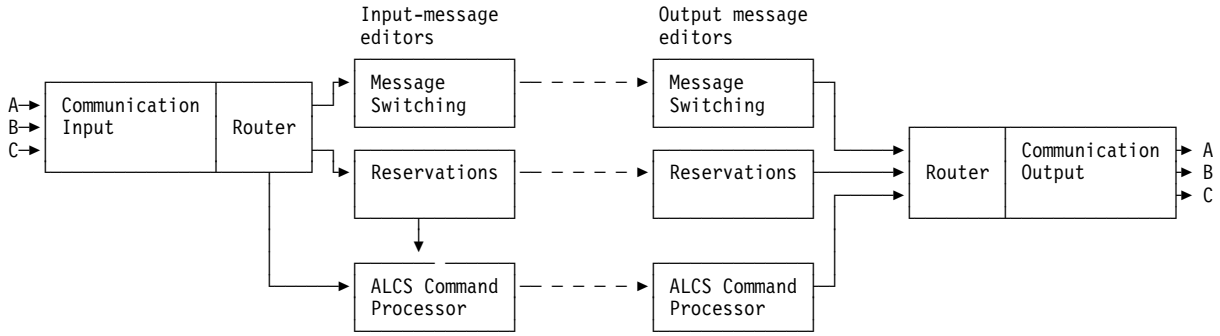


Figure 22. Example of routing for an airline application system

Note that in Figure 22 there is a connection from the reservations input-message editor to the ALCS command processor. That is because the reservations input-message editor accepts input messages that begin with the character `Z`. The reservations input-message editor checks these input messages to see whether they are ALCS commands. If they are commands it enters (passes control to) the ALCS command processor.

This checking process is one example of the type of processing that an input-message editor can perform.

Input messages — general processing

In general, the input-message editor decides how to process the input message. To do this it can:

- Check the address of the originating terminal. If it is not authorized to use the application, the input-message editor can reject the input message, in which case it must send a response to the originator. The input-message editor can enter an output-message editor to do this.
- Check the type of the originating terminal. The input-message editor can carry out message editing that depends on the originating terminal type. For example, International Programmed Airlines Reservation System (IPARS) input messages from WTTY terminals do not use the same format as input messages from IBM 3270 terminals.
- Check the input message to see what function the message requests. One application can support a number of different functions. Different input messages can request different functions. For example, the IPARS reservations application supports functions such as:
  - Display flights to a destination at a specific time
  - Reserve seats on a flight
  - List the passengers who have reservations on a flight

  The IPARS input-message editor uses the first 1 or 2 characters of the input message to identify the function that the message requests. These are called the **primary action code** and the **secondary action code** respectively.
- Check that the end user terminal is authorized to request the function.

  ALCS checks that the terminal is authorized to use ALCS.

  End users may have to satisfy additional authorization requirements before they can use certain applications, or particular functions of an application. The application can conveniently perform this authorization checking during input-message editing.

When the input-message edit processing completes, the input-message editor enters the application program that starts to process the input message. Typically the input-message editor decides which application program to enter, depending on the function that the input message requests.

Processing continues through one or more application programs. For most terminals, the application programs must send a response message to the originating terminal. To do this, the application programs construct the response message, and finally enter the output-message editor.

Data gathering transactions

ALCS creates a new entry to process each input message. However, some applications require several physical input messages to perform a single logical function. For example, the IPARS reservations application builds a record called a **passenger name record** (PNR) that contains information about a passenger. The end user enters separate input messages for separate pieces of information about the passenger. One message can specify the passenger's name, another the passenger's telephone number, and so on.

This type of processing, where consecutive input messages from the same terminal operate together to perform a single logical function, is called a **data gathering transaction**.

Because each input message is a separate entry, the entries must save information about the partially complete data gathering transaction. Each entry saves more information until all the data is available. The final input message of the data gathering transaction checks that all the information is available, and that it is consistent.

The IPARS agents assembly area

One way to implement data-gathering transactions is to use fixed-file records to save information about partially completed transactions. For example, IPARS uses a fixed-file record called the agents assembly area (AAA). Figure 23 and Figure 24 on page 28 show how a record is built from separate input messages.

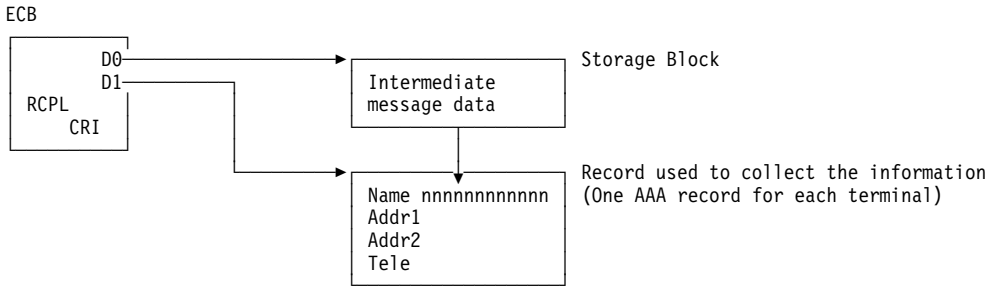


Figure 23. Data gathering transactions: The application stores a message and discards the ECB

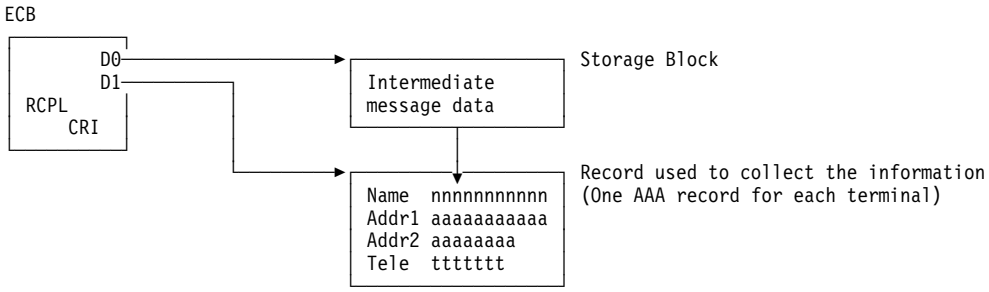


Figure 24. Data gathering transactions: The application stores each intermediate message and discards each ECB

IPARS allocates one AAA record for each terminal. The fixed-file record ordinal number is directly associated with a terminal ordinal number (**communication resource ordinal**).

The ALCS terminal hold facility

Some types of terminal can enter another input message before they receive the response to a previous message. If the application uses a fixed-file record to save information about partially completed transactions, two input messages can update the record at the same time. To prevent this, the application can use the record hold facility. However, the record should not be held for the duration of a whole input message. To avoid this, the application can use the ALCS **terminal hold facility** (also called **AAA hold**).

To use the terminal hold facility, the input-message editor sets terminal hold on and the output-message editor sets it off. They use the `COMCC` monitor-request macro (or the `comcc` C language function) to set terminal hold on and off.

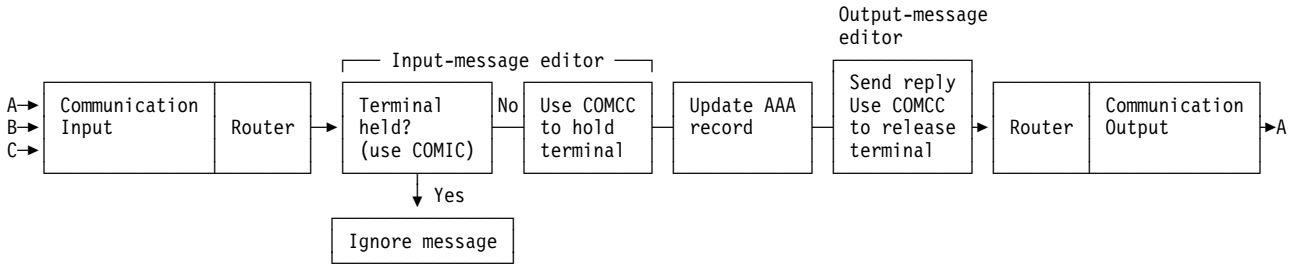


Figure 25. The ALCS terminal hold facility

Before the input-message editor sets terminal hold on, it checks (with the `COMIC` monitor-request macro or `comic` C language function) to see whether it is already on. If terminal hold is already on, the application is still processing a previous input message; the input-message editor ignores the second input message (it does not even send a reply).

Output message processing by application programs

After the application programs construct the response message, they enter an output-message editor. As with input-message editors, each application can have its own output-message editor, or several applications can use the same output-message editor.

Figure 22 on page 26 shows a possible routing arrangement for an airline application system. The system has two applications, reservations and message switching. Each application has its own output-message editor.

The output-message editor eventually issues a SEND-type (SENDC or ROUTC) monitor-request macro. These macros transmit the response to the originating terminal.

In addition to issuing these macros and functions, the output-message editor can provide functions such as:

Canned messages: The application program does not provide the text of the message. Instead, it provides a code that identifies a particular standard message (called a canned message) from a list of standard messages. The output-message editor constructs and sends the response.

Scrolling ALCS allows an application to construct a response message that contains:

- More columns than the output terminal supports
- More lines than the output terminal supports
- A combination of both

The output-message editor saves the whole response message (in pool records) if it cannot fit on one screen. It then builds and sends a new output message that fits on one screen. The output-message editor also supports ALCS commands that scroll the screen over a large message. Figure 26 shows the effect of scrolling.

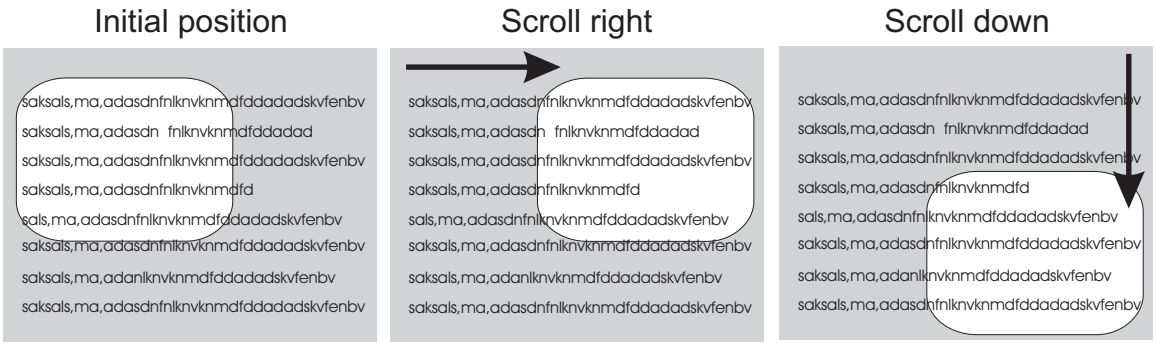


Figure 26. Scrolling moves a screen-sized window over a large response message

Some applications provide scrolling facilities, but ALCS provides a monitor service for scrolling which you can use instead. It is implemented using the DISPC monitor-request macro and the ZSCRL command. These are described *ALCS Application Programming Reference – Assembler* and *ALCS Operation and Maintenance*.

1.5.6 Output message processing by the ALCS online monitor

ALCS application programs construct response messages and then call an output-message editor that uses a SEND-type monitor-request macro to request transmission of the message.

To process one of these macros or functions, the ALCS online monitor first checks that the message is in the standard ALCS output message format (for example,

that the character count is correct, and that the CRI in the output message or RCPL is valid).

IBM 3270 terminals on a VTAM or TCP/IP network

The CRI determines the routing information in the format required by VTAM. The online monitor then uses a VTAM SEND macro to transmit the message to the terminal.

For IBM 3270 terminals on a TCP/IP network, VTAM passes the message to the Telnet server which transmits it over the TCP/IP network to the Telnet client.

ALC terminals on a VTAM network (ALCI or AX.25)

The CRI determines the routing information in the format required by VTAM.

The online monitor translates the message text from the EBCDIC code used by ALCS applications to the transmission code used on the high-level network (HLN).

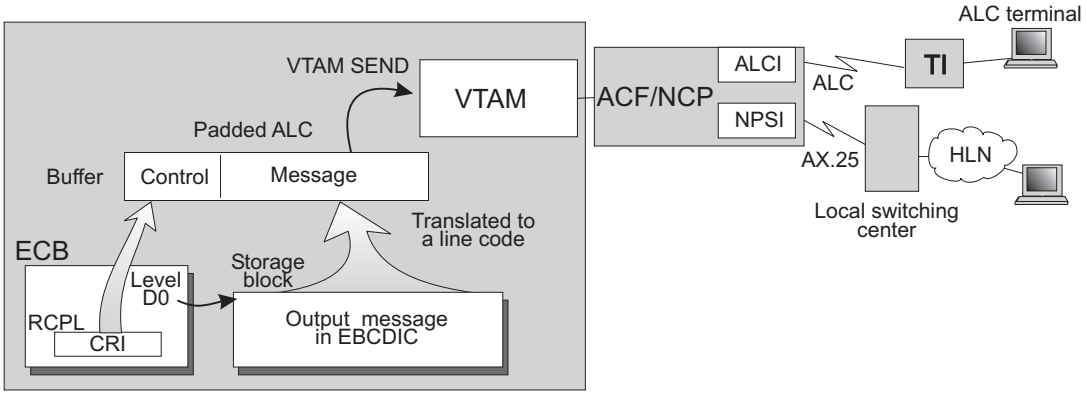


Figure 27. Output message processing: ALCS translates the EBCDIC data to the line code and passes it to VTAM

The online monitor then uses a VTAM SEND macro to transmit the message to the local HLN switching center, which forwards it to the remote terminal.

ALC terminals on a TCP/IP network

The CRI determines which TCP/IP connection is used for transmitting the message. It also determines the routing information, if the terminal is connected through a high-level network.

The online monitor:

- Translates the message text from the EBCDIC code used by ALCS applications, to the transmission code used on the network.
- Adds the message text and routing information to the output queue for the TCP/IP connection.

When the TCP/IP connection is free, ALCS transmits the highest priority message from its queue.

ALCS uses the SEND socket calls to transmit the message over the TCP/IP network to the remote terminal.

Terminals on an SLC high-level network

The CRI determines which SLC link is used for transmitting the message. It also determines the routing information, as required by the HLN.

The online monitor:

- Translates the message text from the EBCDIC code used by ALCS applications, to the transmission code used on the high-level network.
- Adds the message text and routing information to the output queue for the SLC link.

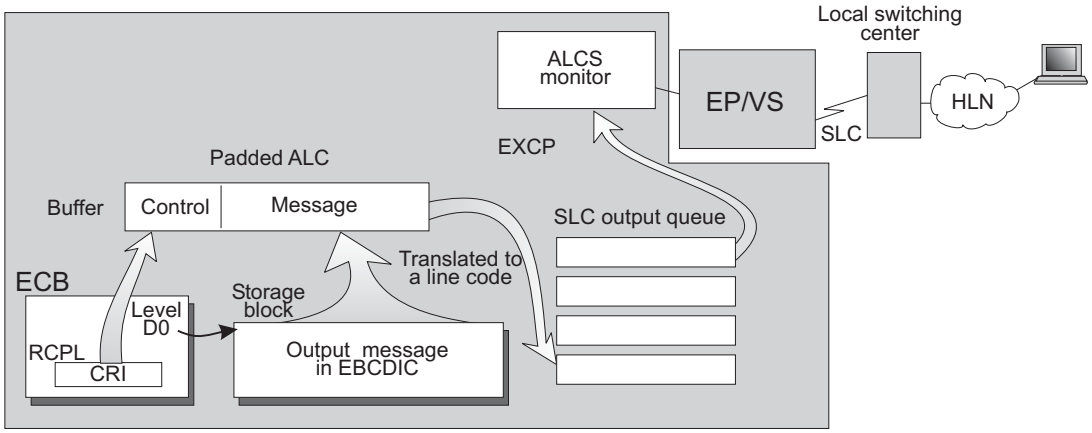


Figure 28. Output message processing: ALCS translates the EBCDIC data to the line code and adds it to an SLC queue

When a channel (for this SLC link) is free, ALCS transmits the highest priority message from the queue for this SLC link.

In this way, ALCS transmits the message to the local HLN switching center, which forwards it to the remote terminal.

1.6 Standard record and storage block sizes

ALCS supports up to eight standard sizes for records on DASD. These sizes are called **L1**, **L2**, and so on, up to **L8**. ALCS supports the same standard sizes for sequential file records, plus another standard size called **L0**. ALCS storage management allocates blocks of storage to entries in these same standard sizes. Application programs can use these standard size storage blocks for reading and writing DASD and sequential file records or for other purposes (such as for work areas).

When you define the characteristics of your ALCS installation in the ALCS generation, you specify which of these sizes your ALCS system will support, and what the actual sizes (in bytes) are. When you are deciding which of the nine sizes to support, and what actual sizes (in bytes) they are, you need to be aware of:

- How ALCS stores DASD records
- ALCS minimum requirements
- Application portability and TPF compatibility

1.6.1 How ALCS stores DASD records

This section describes how ALCS stores DASD records its own data sets. It does **not** describe how (for example) DB2 for z/OS stores its relational database.

ALCS stores records on DASD in VSAM control intervals. Although VSAM supports a variety of control interval formats, ALCS imposes the following restrictions:

- The control interval size must be the same as the physical record size.
- Each control interval must contain one and only one record.
- Within any one cluster, all records must be the same size.

Figure 29 shows the format of a VSAM control interval. Notice that VSAM requires the control interval size to be a multiple of 512 bytes (for large CIs, the CI size must be a multiple of 2KB). In addition to the record itself, the VSAM control interval contains a record definition field (RDF) and a control interval definition field (CIDF) which together occupy the last eight bytes of the control interval. (Although ALCS does not exploit this, VSAM allows there to be unused space between the end of the VSAM logical record and the 8-byte RDF/CIDF.)

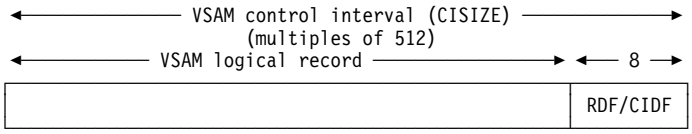


Figure 29. VSAM control interval format

Offline programs which access ALCS general files (general data sets) read (VSAM `GET`) and write (VSAM `PUT`) VSAM logical records. But online application programs read (find service) or write (file service) standard size ALCS records (L1, L2, and so on). ALCS stores the standard size ALCS record at the start of the VSAM logical record, and reserves the last 56 bytes of the VSAM logical record to contain ALCS control information. Typically, there are some unused bytes between the end of the standard size ALCS record and the start of the 48-byte reserved area. Figure 30 shows this layout.

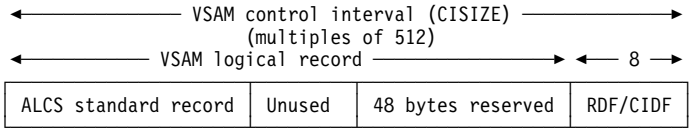


Figure 30. ALCS record in a VSAM control interval

For example, a 512-byte control interval contains a 504-byte VSAM logical record. Because ALCS reserves 48 bytes, this allows a maximum of 456 bytes for the ALCS standard size record. Most ALCS installations define size L1 as 381 bytes which leaves 75 bytes unused.

1.6.2 ALCS minimum requirements for standard sizes

All ALCS systems must support at least sizes L0, L1, L2, and L3. ALCS itself uses sizes L1, L2, and L3. ALCS assumes that these are greater than or equal to the following minimum sizes in bytes:

L1 381 bytes
L2 1055 bytes
L3 4000 bytes

Note: You cannot change size L0, it is always 127 bytes.

1.6.3 Application portability and TPF compatibility

When you define the standard record and block sizes for your ALCS system, consider that:

- You may want to buy applications developed to run on other systems (TPF or ALCS). These applications may contain dependencies on the standard sizes defined on the vendor's system.
- You may want to sell applications that you develop. It may be difficult to port your application if it contains dependencies on standard sizes that are not defined, or are defined differently on your customer's system.

TPF only supports sizes L0, L1, L2, and L4 for application program use, and it does not allow these sizes to vary from installation to installation. Many applications developed for use with TPF depend on these TPF record sizes; they might not work unless you define the sizes as follows:

L0 127 bytes
L1 381 bytes
L2 1055 bytes
L4 4095 bytes

You should also consider using these sizes for L0, L1, L2, and L4 if you plan to develop new ALCS applications. This will make your application easier to port if you ever sell it to a TPF user, or install TPF yourself.

Note that you can greatly enhance portability of applications that you develop by exploiting the IBM TPFDF product. TPFDF helps you to write applications that do not contain dependencies on the actual sizes of records that you access.

1.6.4 Recommendations and requirements for record and block sizes

Figure 31 summarizes the ALCS requirements and recommendations for standard record and block sizes.

Figure 31. Storage block sizes

Storage reference	Number of bytes of application data	Notes
L0	127	Fixed by ALCS
L1	381	Recommended for TPF compatibility
L2	1055	Recommended for TPF compatibility
L3	4000	Minimum
L4	4095	Recommended for TPF compatibility
L5	Up to 32K	As required
L6	Up to 32K	As required
L7	Up to 32K	As required
L8	Up to 32K	As required

TPF compatibility

If your application program must be compatible with TPF, do not use sizes L3 and L5–L8 explicitly. You can, however, use these sizes through the IBM TPFDF program. TPFDF allows you to use any record size (decided by the database administrator) without explicitly specifying the size in your application programs.

1.6.5 Sequential file records

ALCS provides services that allow application programs to read and write standard size sequential file records (L0, L1, and so on). Sequential file records are stored in conventional record formats, without additional ALCS or VSAM control information.

ALCS also provides services that allow application programs to read and write arbitrary size sequential file records.

1.7 Multiprogramming and multiprocessing

Entries: An input message is an example of an ALCS **entry**. An entry is a unit of work, similar to an MVS task. ALCS can process entries independently of each other. It can use:

Multiprogramming

Interleaves the processing of several entries on the same processor

If an application program that is processing one entry waits (for example, for I/O), ALCS can start or resume processing another entry. In this way, ALCS can interleave the processing of several entries on the same processor.

Multiprocessing

Processes several entries at the same time on different processors

If ALCS runs on a multiprocessor it can process entries simultaneously on different processors. For example, on a four-way multiprocessor, ALCS can process four entries simultaneously.

Relationship between entries and tasks: ALCS does not ATTACH a new MVS task for each entry. Several ALCS entries can run under the same MVS task. To process entries, ALCS ATTACHes one or more MVS tasks during ALCS initialization. The number of tasks is a run-time option; it determines the number of processors that ALCS can use to process entries (each task can run on a separate processor). Each of these tasks can start or resume processing of any entry. One entry can start processing under one task, wait, then resume under another task.

1.7.1 Re-entrant application programs

One ALCS application program often processes more than one entry. For example, several end users can request a time display at the same time. Each request (input message) is a separate entry, but there is only one program that generates the time display response (output message).

If one application program processes several entries, ALCS uses the same copy of the program for all entries. Because of this, ALCS application programs must be **reentrant**. A reentrant program does not modify itself; for example, it does not contain work areas or switches that it modifies during execution.

If an ALCS application program is not reentrant, it can fail unpredictably. ALCS terminates any application program that attempts to modify itself.

1.7.2 Entry control block

Because application programs must be reentrant, they cannot use internal data areas and switches. Instead, they must use storage that is associated with an entry, not with a program. ALCS provides an area of storage for each entry. This storage area is called the **entry control block (ECB).** The ALCS online monitor creates a new ECB for each new entry.

Figure 32 on page 36 shows the ECB format. The figure shows some assembler symbols such as:

- EBW000 (in a user work area)
- CE1FA0 (a storage level)
- EBROUT (an entry-origin field)

For assembler programs, the ALCS EB0EB macro generates the EB0EB DSECT that defines these labels.

Similarly for C-language programs, the c$eb0eb header file generates the eb0eb struct that defines the labels.

Note: The $ character is the national currency symbol (X'B5').

Other high-level language programs (for example COBOL) do not have direct access to the ECB.

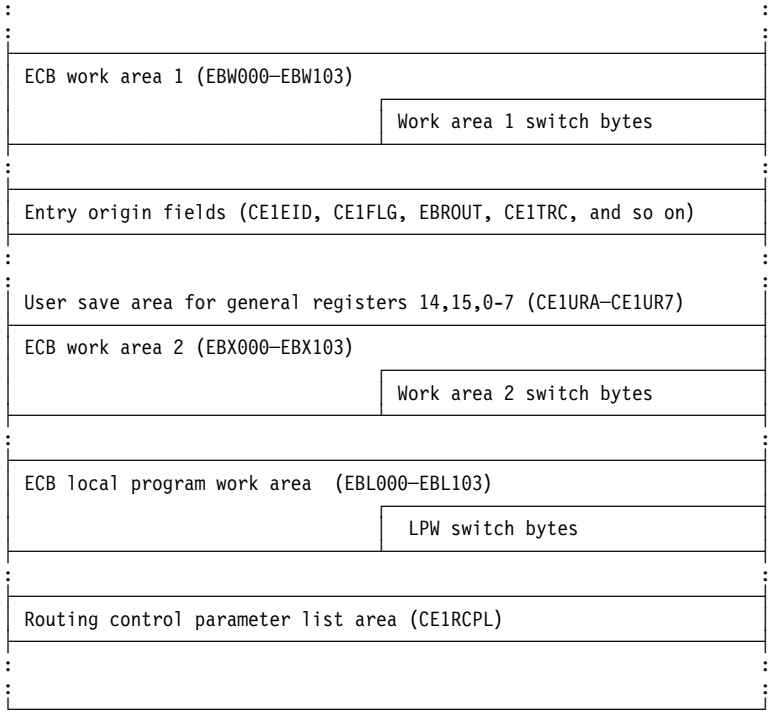


Figure 32. Some entry-control-block areas

An application program can safely store information about a particular message in the ECB. Information about another message does not overwrite it because each message is a separate entry and has its own ECB.

User-defined ECB fields

ALCS application programs can use the ECB work areas for any purpose. The areas are:

- Work area 1 (EBW000 through EBW103) and its switch bytes
- Work area 2 (EBX000 through EBX103) and its switch bytes

Programs typically use these areas for intermediate results, for passing parameters between programs, and so on.

ALCS also provides a local program work area (EBL000 through EBL103 and its switch bytes) which is “local” to the program. It is cleared to binary zeros on entry to the program and the contents are saved/restored across enter/back.

TPF compatibility

Do not use the local program work area in programs that must be compatible with TPF

Note: ALCS does not provide local program work area support for C language programs.

However, unrelated programs (programs which do not call each other) can, and often do, use ECB work area fields for different purposes. Therefore, fields defined within the ECB work areas are usually **specific** to a particular program or group of programs.

ALCS provides a special area (CE1USA) in the ECB, where the system programmer can define fields which are **installation-wide**.

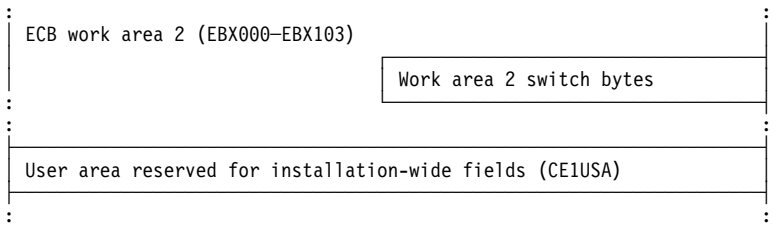


Figure 33. Installation-wide ECB user fields

Installation-wide fields can contain the same information for all entries, regardless of which application programs are using the ECB.

ECB levels and attached storage blocks

Application programs that read or write records do not use the ECB to store the records. Instead, they use additional storage called **storage blocks**. Application programs can also use storage blocks for work areas if they need more storage than the ECB itself.

Storage levels

To obtain a storage block, an application program uses a monitor-request macro or C language function. The monitor-request macro specifies an ECB field called a **storage level**. The ALCS ECB contains 16 storage levels for application program use. The monitor-request macro obtains a storage block and attaches the block to the storage level; that is, it saves information such as the block address and block size in the storage level.

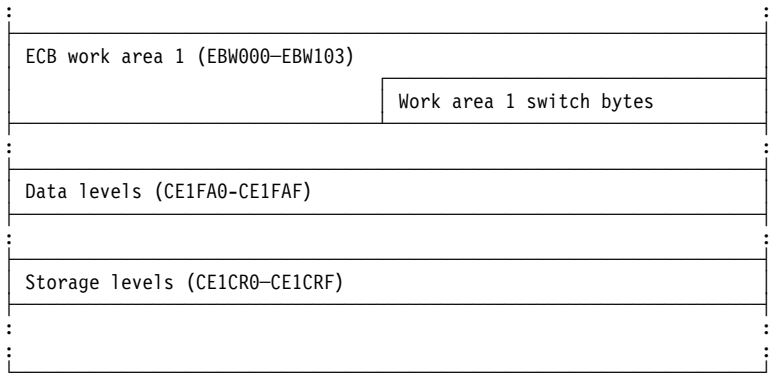


Figure 34. ECB levels

Some storage blocks can be associated with an entry but not attached to a storage level. These include:

- Detached storage blocks
- Automatic storage blocks

Detached storage blocks: Application programs can use the `DETAC` monitor-request macro (`detac` C function) to detach a storage block from a storage level. They can then obtain another storage block and attach it at that storage level. A detached storage block is still associated with the entry (application programs can use the `ATTAC` (`attac` C function) macro to re-attach it).

Automatic storage blocks: Assembler application programs can use the ALASC monitor-request macro to obtain an automatic storage block. An automatic storage block is associated with an entry, but it is not attached at a storage level. The BACKC monitor-request macro releases automatic storage blocks.

See *ALCS Application Programming Guide* for a description of how to obtain and use storage blocks.

Data levels

The ALCS ECB includes 16 fields called **data levels**. Each data level is associated with a corresponding storage level. Some monitor-request macros use a data level and the associated storage level. For example, the FILEC macro writes the contents of a storage block to DASD. The storage level contains information about the storage block and the data level contains information about where to write the record.

1.7.3 Data event control blocks (DECBs)

A data event control block (DECB) contains a storage level and data level. An application program can use a DECB as an alternative to using a storage level or data level in the ECB. Although a DECB does not physically reside in an ECB, the DECB fields specify the same information as those in the ECB. An application program can dynamically acquire a DECB by using the DECBC FUNC=CREATE monitor-request macro (tpf_decb_create C function).

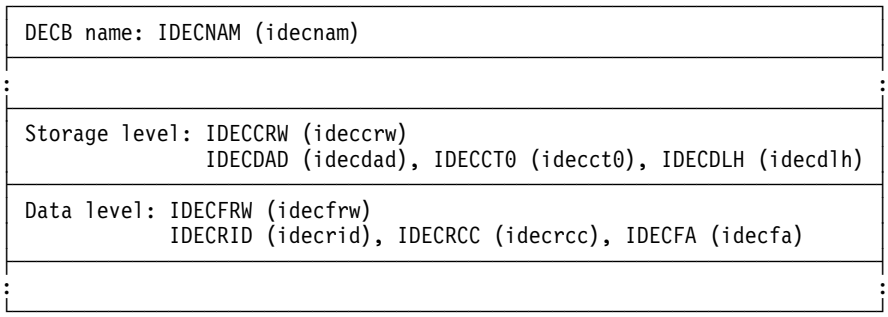


Figure 35. DECB level

For Assembler programs, the ALCS IDECB macro generates the IDECB DSECT that defines the labels for the fields in a DECB.

Similarly for C-language programs, the c$decb header file generates a C data structure defined as type TPF_DECB that defines the labels.

Note: The $ character is the national currency symbol (X'B5').

See *ALCS Application Programming Guide* for more information on the use of DECBs.

1.7.4 Data collection area

Each ECB has an associated data-collection area.

During the life of an entry, the ALCS online monitor accumulates statistics in fields in this area. If data collection is active, the data-collection routines write these statistics to the **data-collection** sequential file when the ECB exits. If a

data-collection file is not defined, ALCS writes the information to the diagnostic file. There is an optional extension to this area which installations can use.

You can use offline programs such as:

- The ALCS statistical report generator (SRG)
- The Service Level Reporter (SLR).

to process the statistics and produce reports that show how entries use ALCS resources.

1.7.5 Serialization – forcing exclusive access to resources

Each ALCS entry has its own ECB, its own DECBs and its own attached and detached storage blocks (or heap and stack for high-level languages), which all belong exclusively to that entry. ALCS application programs can therefore use the storage without interference from other entries.

However, all ALCS entries can share other resources, in particular:

- Database records
- The application global area
- Sequential files.

In order to allow application programs to share these resources without inadvertently overwriting their contents, ALCS allows programs to force exclusive access to a resource while they are using it, as follows:

- Force exclusive access to the resource
- Perform instructions that use the resource
- Release the resource so that other entries can use it.

While one entry has exclusive access to a resource, ALCS queues any other entries that want to use it until the first entry releases it.

Chapter 2. Communication management

This chapter describes the communication protocols that ALCS supports.

2.1 ALCS communication resources and resource addressing

ALCS communication support can receive data from various sources, and send data to various destinations. For example, it can receive data (messages) from IBM 3270 displays, and it can send data (messages) to IBM 3270 displays and printers.

ALCS can also receive messages from, and send messages to, applications and NetView operator IDs. An ALCS application is a set of related functions that application programs provide. Figure 22 on page 26 shows how the ALCS input-message router passes messages to an input-message editor (destination).

These sources and destinations are examples of **ALCS communication resources**. Figure 36 shows the ALCS communication resources.

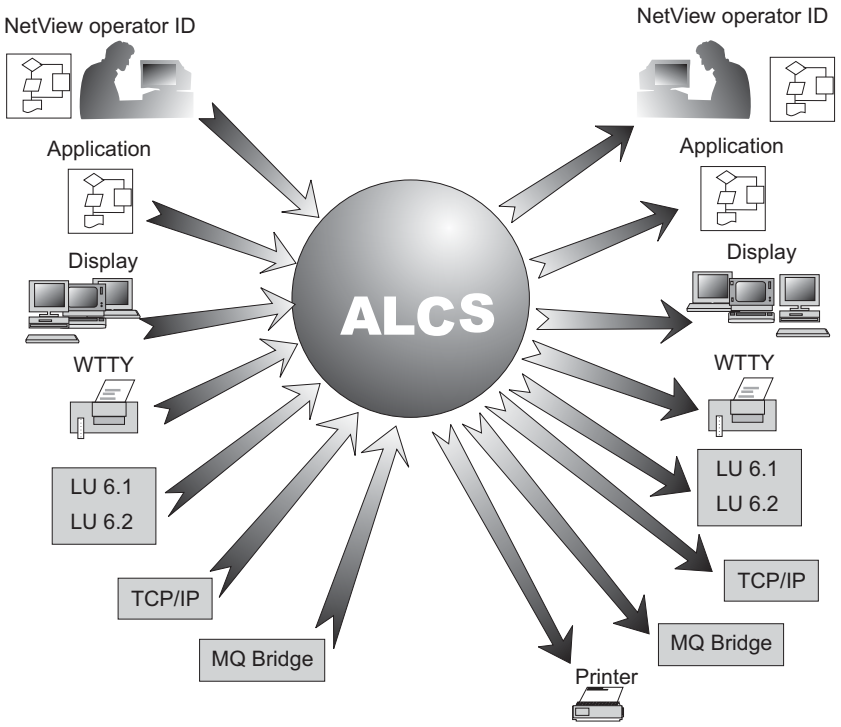


Figure 36. ALCS communication resources: Sources and destinations

The ALCS communication generation defines every ALCS communication resource. The LDTYPE= parameter of the `COMDEF` macro (in the ALCS communication generation) specifies the **logical-device type** of an ALCS communication resource.

Generally, ALCS communication support does not need to know about the communication network components (links, terminal control units, and so on) that connect terminals. ALCS does not control or use these components directly. Consequently, these components are not ALCS communication resources. For

example, IBM 3270 terminal control units, and SDLC links that connect them, are not ALCS communication resources.

However, there are exceptions. For example, ALCS can send messages to ALC terminals connected through an SLC high-level network (HLN). To do this, ALCS communication support must use the correct SLC link (the link that connects the HLN). SLC links are ALCS communication resources because ALCS must know about them.

Your system programmer or network control group chooses all the VTAM LU names; the same person (or group) should choose the ALCS CRNs for resources that do not have VTAM names.

All VTAM LU names must be unique, and all CRNs must be unique. Figure 37 shows the relationship between CRNs and SNA LU names.

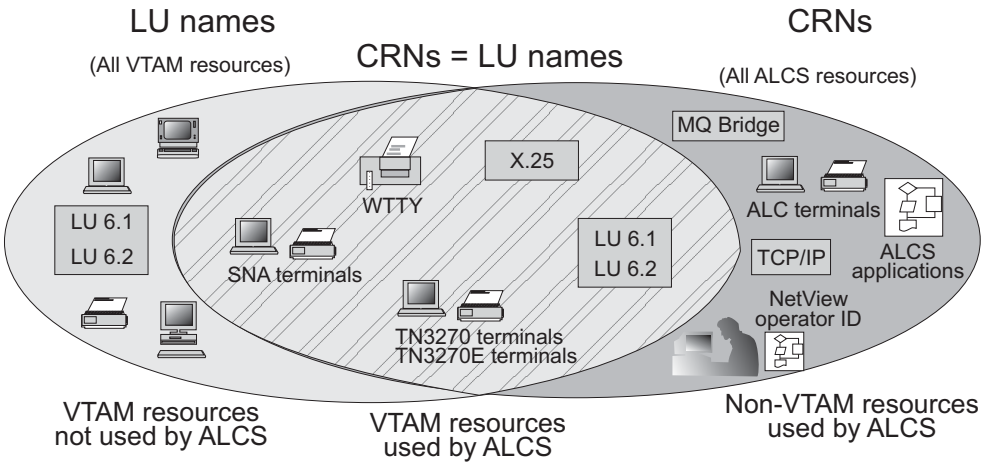


Figure 37. ALCS communication resource names and LU names overview

An ALCS communication resource is known in two ways:

- An external name, the communications resource name (CRN)
- An internal identifier, the communications resource identifier (CRI)

Figure 38 on page 43 summarizes the use of CRNs and CRIs on different communication resources; the numbers in parentheses refer to the notes that follow the figure.

Figure 2 on page 3 shows a summary of ALCS communications.

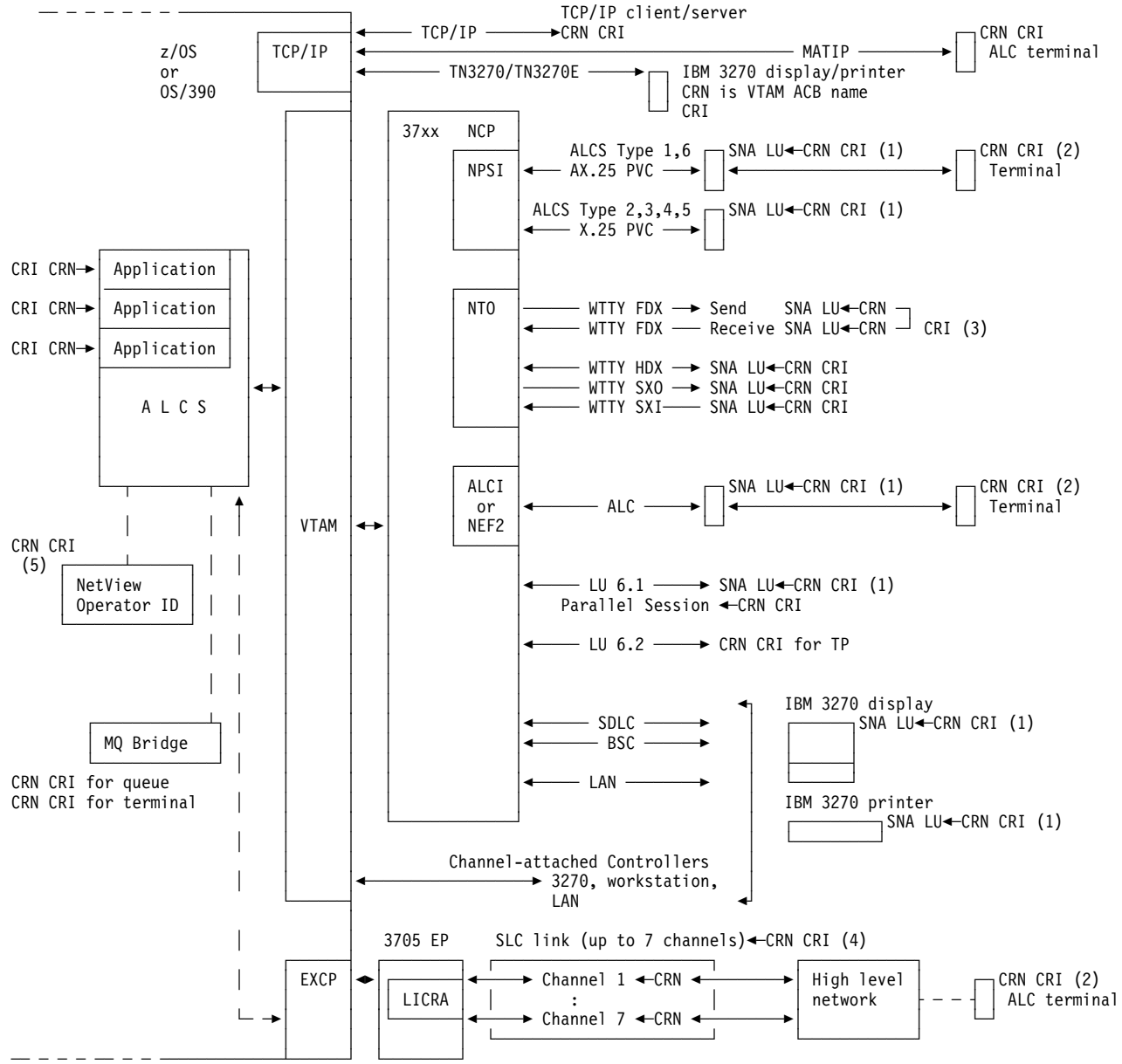


Figure 38. Communication resource names (SNA LU, CRNs, and CRIs)

Notes:

1. The CRN is the same as the SNA LU name.
2. Specify a CRN for non-VTAM resource.
3. A WTTY FDX link has two CRNs, but only one CRI.
4. The SLC link has a CRN and CRI. The channel CRN is the link CRN with the channel number appended. There is no channel CRI.
5. The CRN is the same as the NetView operator ID.

2.1.1 Communication resource name (CRN)

Every ALCS communication resource has a unique name, up to 8 characters in length, called the **communication resource name (CRN)**. If a communication resource is an SNA* logical unit (LU), the CRN is the same as the LU name. For communication resources which do not have an LU name, specify the CRN for a non-VTAM resource.

In both cases, the NAME= parameter of the COMDEF macro (in the ALCS communication generation) specifies the CRN for an ALCS communication resource.

ALCS commands and responses can use the CRN to identify a particular communication resource.

Special cases for CRNs

- **WTTY full-duplex (FDX)** links are unusual because each link has two CRNs, one CRN for the send line and one for the receive line.
- **NetView CRNs** are unusual in that the CRN is the NetView operator ID, not a physical device.
- **SLC CRNs** are unusual because each SLC channel has a CRN (the CRN refers individually to the SLC channel). But each SLC link also has a CRN (the CRN refers collectively to all channels of the link). The link CRN is up to 7 characters in length. The channel CRNs are the link CRN with the channel number appended.

Reserved CRNs

ALCS reserves the following CRNs, therefore do not use them as the operand of NAME in any COMDEF macro:

```
ALCSAPPL
ALCSLINK
APPC
AP1 through AP255
AT1 through AT255
MQ
NETVIEW
NONE
PRC
ROC
SLCLINK
TCPIP
TERMINAL
WTTY
```

2.1.2 Communication resource identifier (CRI)

The CRN is the **external** identifier for an ALCS communication resource. However, application programs normally do not use the CRN to refer to communication resources. Instead, they use an **internal** identifier called the **communication resource identifier (CRI)** to refer to communication resources.

The CRI is a unique 3-byte number which ALCS automatically assigns to each communication resource. The numbers are normally assigned sequentially, but you can specify an explicit number (or range).

ALCS Installation and Customization describes how to specify an explicit CRI or a range of CRIs. The CRI corresponds to the TPF LN/IA/TA format address (but note that it cannot be split up into line, interchange, and terminal address components) and the ALCS/VSE LN/ARID format address.

Special cases for CRIs

- **SLC links:** The individual channels of a link have their own CRNs, but share the CRI of the link.
- **WTTY full-duplex links:** The send and receive sides of a link have their own CRNs, but share the CRI of the link.

Most ALCS applications (specifically those that use IMSG, OMSG, and AMSG message formats) use the full 3-byte CRI to address communication resources. But some applications (specifically those that use XMSG format messages) use a 1-byte communication resource address. In ALCS, these applications use the low-order byte of the CRI. For readers familiar with TPF or ALCS/VSE, this corresponds to using the symbolic line number (SLN).

2.1.3 Computer room agent set (CRAS)

A CRAS terminal is an ALCS terminal that is authorized for restricted commands. Many ALCS commands can only be entered from a CRAS terminal. See *ALCS Operation and Maintenance* for details of these commands. ALCS supports 512 CRAS terminals in four types:

- 1 Prime CRAS
- 1 Receive Only CRAS (RO CRAS)
- 255 Alternate CRAS terminals
- 255 Alternate CRAS printers

In addition to this, many other terminals can be assigned CRAS ***authority*** of Prime CRAS or alternate CRAS. It is possible that these terminals themselves may have CRAS status of Prime CRAS or alternate CRAS, or indeed may have no CRAS status. There is an unlimited number of terminals that may have Prime CRAS or alternate CRAS ***authority***.

A terminal with a particular CRAS status has the authority associated with that CRAS, for example AT1 CRAS has AT1 authority.

At any given time CRAS terminal AT1 may also have Prime CRAS ***authority*** and the user of the terminal can issue all the commands associated with the Prime CRAS. Each CRAS terminal has an additional CRN (a CRAS CRN) which is associated with its normal CRN. This topic is discussed in more detail in 2.1.5, “Special addressing for CRAS terminals” on page 47.

Prime CRAS (CRN PRC) is the primary terminal that controls the ALCS system. RO CRAS (CRN ROC) is a printer to which ALCS sends certain messages about system function and progress.

There can be up to 255 alternate CRAS terminals (CRN AT1 through CRN AT255). There can be up to 255 alternate CRAS printers (CRN AP1 through CRN AP255).

It is possible to define a CRAS printer as AT*nnn*, but this is not advisable. IBM recommends that you use AP*nnn* for defining a printer unless an existing configuration dictates that you cannot.

The terminals with Prime CRAS and alternate CRAS (AT1 through AT16) authority are the only terminals that can

- Transfer or assign Prime or RO CRAS
- Transfer or assign AT1 through AT16
- Transfer or assign AP1 through AP16

Alternate CRAS terminals (AT1 through AT16) are the only CRAS terminals that can be fallback CRASs for Prime CRAS. AP1 through AP16 are normally the fallback CRAS printers for RO CRAS. However, if AT1 through AT16 are printer devices, they too can be fallback CRAS printers for RO CRAS.

Note: When you assign a fallback terminal or printer, consider the implications of, for example, the prime CRAS function being inadvertently assigned to:

- A terminal in a non-secure area
- A remote terminal outside the computing center

Figure 39 summarizes ALCS CRAS terminals.

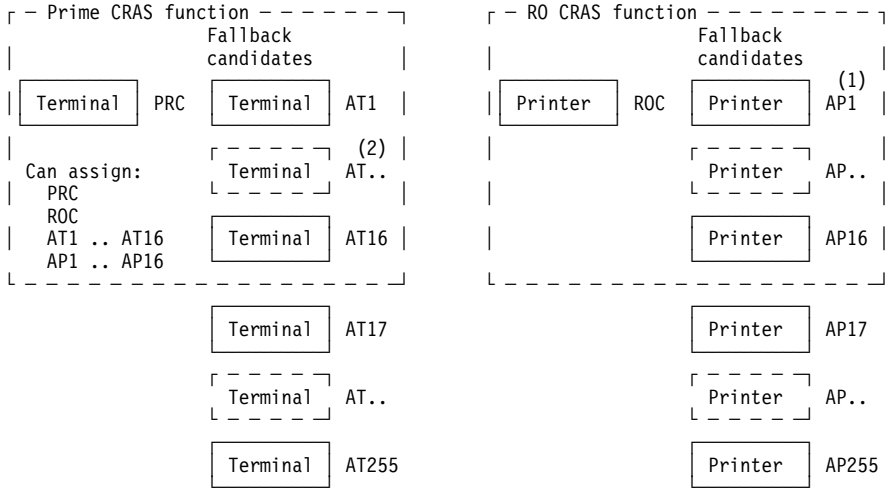


Figure 39. CRAS CRNs for terminals and printers

Assigning Prime or alternate CRAS authority to a terminal other than the originator requires Prime CRAS authority. Assigning Prime or alternate CRAS authority to the originating terminal only requires the appropriate SAF authorization for the user logged on to the terminal. See 2.1.4, “CRAS authority and Security Authorization Facility (SAF)” on page 47 for more details on SAF authorizations and *ALCS Installation and Customization* for a full description of using SAF compliant security programs in an ALCS installation.

Notes:

1. **Alternate CRAS printer AP1**

   By defining a NetView operator ID as AP1 you can route network control blocks (NCBs) to the NetView Network Problem Determination Application (NPDA) database. ALCS can receive NCBs from the SITA HLN connected by an SLC link (using the P.1124 protocol) or by an X.25 PVC link.

2. **Alternate CRAS AT4**

   By defining a printer as AT4, you can route all error and status messages relating to communication to one particular printer (possibly located in a communication control office). By default, ALCS routes all communication messages to AT4; if AT4 is not defined, or is not a printer, then ALCS routes the messages to RO CRAS.

CRAS routing

You can use the installation-wide exit AXA0 to change the routing for ALCS communication error and status messages to suit your installation. *ALCS Installation and Customization* describes the AXA0 installation-wide exit.

NetView

If you have NetView installed, you can define NetView operator ID as ALCS CRAS terminals.

2.1.4 CRAS authority and Security Authorization Facility (SAF)

ALCS interfaces with RACF (or another SAF-compliant security product) to protect specific functions of the system to control which end users can invoke them. It does this by controlling which users have authority to use terminals which have Prime CRAS or alternate CRAS authorities.

Transferring or assigning CRAS terminals or authorities may only be performed if the current users of the terminals have sufficient SAF authority.

Example 1: User A is logged on to the Prime CRAS and user B is logged on to the alternate CRAS AT1. In order to transfer the two CRAS devices user B must have the SAF authority to use a terminal with Prime CRAS authority.

Note: CRAS authorities are hierarchical in nature so it follows that user A must already have sufficient SAF authority to be alternate CRAS AT1.

Example 2: User A is logged on to a terminal with no CRAS status or authority. In order to assign Prime CRAS ***authority*** to himself user A requires SAF authority to use a terminal with Prime CRAS authority. For further information on SAF see *ALCS Installation and Customization*. For information on the ALCS `ZACOM` command which assigns and removes CRAS authority, see *ALCS Operation and Maintenance*.

2.1.5 Special addressing for CRAS terminals

The ALCS communication generation assigns a CRN and CRI to each ALCS communication resource. For each CRAS terminal (up to 512) it also associates a **CRAS CRN** and **CRAS CRI** with the normal CRN and CRI. Each CRAS resource can be addressed using:

- The CRN and CRI assigned during communication generation

- The CRAS CRN and CRAS CRI

The CRAS CRIs are normally only used to send messages. They are not normally found in input messages. The CRI in input messages is the actual CRI of the resource and not the CRAS CRI, even if the resource entering the message is a CRAS. Figure 40 shows which CRI ALCS uses for CRAS messages.

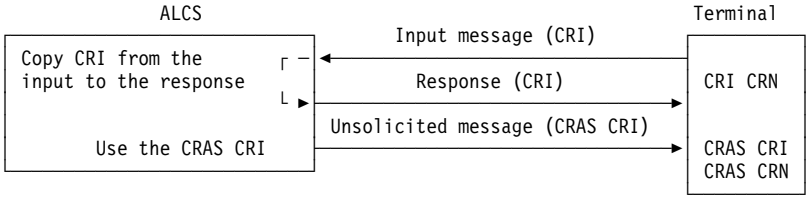


Figure 40. The CRI in messages to and from a CRAS

ALCS reserves some CRNs and ranges of CRIs for CRAS terminals. Figure 41 lists these ranges.

Figure 41. CRAS CRNs and CRI ranges

CRAS Type	CRAS CRI	CRAS CRN	Remarks
RO	X'000000'	ROC	Must be a printer or NetView operator ID. For testing purposes, you can assign the RO CRAS to a test (STV) 3270 printer so that the output is directed to the ALCS diagnostic file instead of a physical printer.
Alternate printer	X'000001'	AP1	Use this only for the NetView operator ID that connects to the Network Problem Determination Application (NPDA).
Alternate printer	X'000002' through X'0000FF'	AP2 through AP255	Must be a printer or NetView operator ID. For testing purposes, you can assign these to test (STV) 3270 printers so that the output is directed to the ALCS diagnostic file instead of a physical printer.
Prime	X'010000'	PRC	Must be a display or NetView operator ID.
Alternate	X'010001' through X'0100FF'	AT1 through AT255	Can be a display, printer, or NetView operator ID.

2.1.6 Communication resource ordinal

The ALCS communication generation assigns a unique number (the **communication resource ordinal**) to each communication resource. The numbers are normally assigned sequentially, but you can specify an explicit number (or range).

ALCS Installation and Customization describes how to specify an explicit ordinal (ORD) or a range of ordinals (IORD). Application programs can use the communication resource ordinal for any function that requires a unique number associated with each communication resource. For example, an application can have a fixed-file record type that has one fixed-file record for each communication resource. The application can use the communication resource ordinal as the record ordinal. This associates each record with the corresponding communication resource.

ALCS uses this technique for the **resource control record**. The IPARS application (supplied with ALCS) uses this technique for the **agents assembly area** (AAA).

Application programs can use the COMIC monitor-request macro (or the C language comic function) to get the communication resource ordinal for a communication resource.

2.2 Message router

Figure 42 shows an overview of the message routing that ALCS provides.

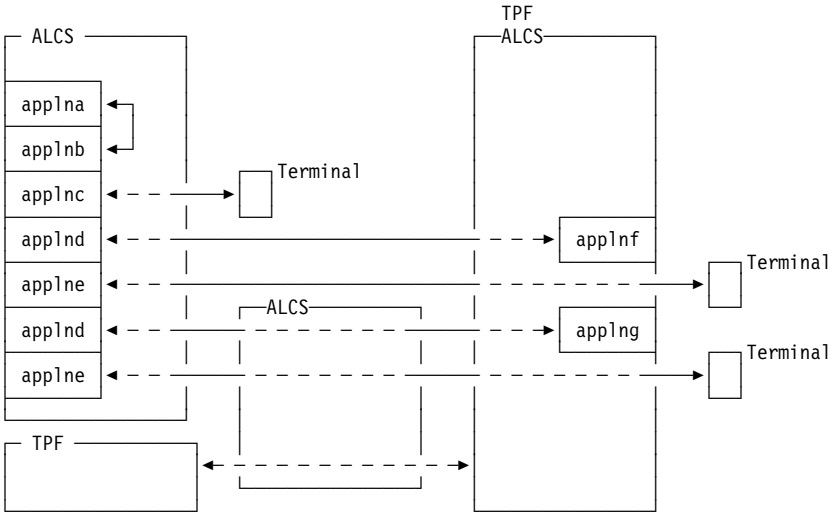


Figure 42. ALCS message routing to and from communication resources

The ALCS message router provides the following message routing functions:

- From an application to a terminal or to another application. The destination terminal or application can be owned by the same ALCS system as the originating application, or by another system. Use the ALCS ROUTC macro to route messages from an application. If an ALCS application issues a send-type macro to send a message to a terminal owned by another system, the send-type macro processing passes the message to the ALCS message router.
- From a terminal to an application. The destination application can be owned by the same ALCS system as the originating terminal, or by another system. Use the ALCS ZROUT or ZACOM command to change the application to which messages from a terminal are routed.
- From an application owned by another system, to a terminal or to another application. The destination terminal or application can be owned by this ALCS or by a third system.
- From a terminal owned by another system, to an application. The destination application can be owned by this ALCS or by a third system.

For each input or output message, the message routing information is in a routing control parameter list (RCPL). This information includes the origin, destination, and characteristics of the message.

Assembler programmers use the RC0PL DSECT macro, to reference fields in the RCPL. See *ALCS Application Programming Reference – Assembler* for more information about the RCPL.

C language programmers use the <c$rc0pl.h> header to reference fields in the RCPL. See *ALCS Application Programming Reference – C Language* for more information about the RCPL.

Specify candidate paths for routing messages between ALCS and a system in a different processor (for example, another ALCS system, or a TPF system) in the ALCS communication generation. Candidate paths include Type 2 SLC links that are reserved for use by the ALCS message router, and LU 6.1 links.

The ALCS message router sends and receives data messages on a message routing path to another system in PPMSG format; that is, the RCPL precedes the message text.

2.2.1 Addressing other-system resources

ALCS can communicate with resources (terminals and applications) that other ALCS or TPF systems own and control. ALCS routes messages to other-system resources using a **cross-system ID**. This is the 3-byte identifier (CRI or LN/IA/TA) by which the resource is known to the other system.

Each communication resource known to ALCS has a unique CRI, a unique CRN, and a unique ordinal number, whether or not ALCS owns the resource itself. Each combination of terminal ID and communication ID is unique within ALCS.
Figure 43 gives an overview of the cross-system ID.

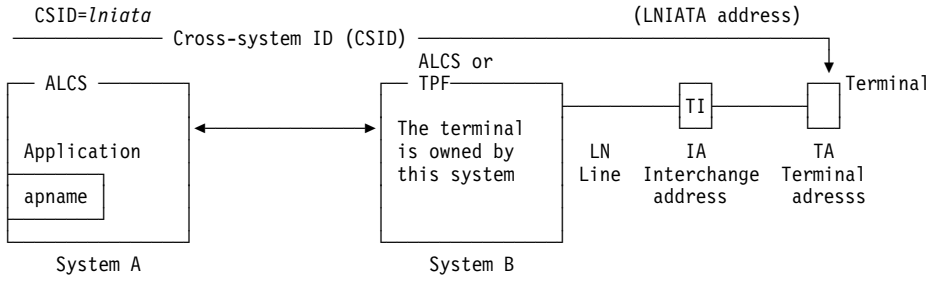


Figure 43. The cross-system ID (CSID)

Note: The application uses the local CRI to communicate with the terminal. The mapping process of the local CRI to the link and terminal ID is performed by:

- System A or system B (when system B is ALCS)
- System A (when system B is TPF)

2.3 Logon and logoff, and sine in and sine out

Logon and logoff

ALCS is a VTAM application program. Before a terminal that is an SNA LU can communicate with ALCS, there must be a session between the terminal and ALCS. For example, the end user can enter a special VTAM message, called a logon, that starts the session.

Logon is subject to the end user having sufficient SAF authority to use the terminal with the current CRAS status of Prime CRAS or alternate CRAS.

The end user can use the ALCS ZLOGF command (described in *ALCS Operation and Maintenance*) to log off from ALCS; that is, to stop the session with ALCS. After ZLOGF, the end user can log on to another VTAM application program or can log on to ALCS again.

In this way, logon and logoff start and stop the session between a terminal that is an SNA LU and ALCS.

Logon and logoff do not apply to terminals that are not SNA LUs. For example, ALC terminals that connect through NEF or ALCI are not LUs. Each NEF or ALCI LU corresponds to many terminals. There must be a session between each NEF or ALCI LU and ALCS, but the individual terminals cannot log on or log off. Similarly, terminals that connect through an SLC or AX.25 HLN are not LUs; they cannot log on or log off.

The CRAS status of a terminal remains across end user logon or logoff. However other CRAS authorities of a terminal are removed across end user logon or logoff.

Sine in and sine out

Some applications provide different functions to different end users of the same application. For example, an airline seat reservations application can provide one set of functions for the end users who make seat reservations, and a different set of functions for end users who update flight schedule information.

If an application provides different functions in this way, the end users must specify the functions that they require. This process is called **sine in**. To sine in, the end user must enter a sine-in message. Depending on the application, a sine-in message can include:

- End-user identification
- Function required
- Authorization code

Note that sine in is an application program function. ALCS does not require a sine in and it does not require the application to support a sine in.

If the application supports a sine in, it can also support a sine out. If the sine in includes an authorization code (password) an end user can sine out to prevent unauthorized access to the functions.

2.4 Printer shadowing

Printer shadowing is an ALCS facility that allows up to 16 printers to receive a copy of every message sent to a particular printer. Any type of printer can be shadowed and any type of printer can be used for shadowing. The user is responsible for ensuring that the shadow printers are compatible (line length, character set and so on) with the original printer.

Use the ALCS ZACOM command to control the use of shadow printers and the ZDCOM command to display information about printer shadowing. *ALCS Operation and Maintenance* describes these commands. See also installation-wide exits APR2 through APR4 in *ALCS Installation and Customization*.

2.5 Printer sharing

Printer sharing is an ALCS facility that allows 3270-type printer resources to be shared between ALCS and other applications. You can set up any printer to be shared.

Printer sharing is set up using the ISTATUS= parameter on either the COMDEF or COMDFLT macro during the ALCS generation process.

2.6 Printer redirection

Printer redirection is an ALCS facility that allows output messages for a particular printer to be sent to another printer. You can redirect output for any printer; any type of printer can be used for the redirected output. The user is responsible for ensuring that both printers are compatible (line length, character set and so on). The redirected output can again be redirected. ALCS allows up to 16 printers to be redirected one to another in a chain (but not in a loop).

Use the ZACOM command to control the use of printer redirection, and the ZDCOM command to display information about printer redirection. *ALCS Operation and Maintenance* describes these commands. See also installation wide exits APR7 through APR9 in *ALCS Installation and Customization*.

2.7 Specifying communication resources

The method of specifying ALCS communication resources depends on whether it is the initial definition of the ALCS communication configuration or an update to an existing configuration.

Defining the initial communication configuration

To define the initial ALCS communication configuration, run the ALCS communication generation to generate a **base** communication load module. *ALCS Installation and Customization* describes how to generate a communication load module.

Updating an existing communication configuration

The procedure to update an existing communication configuration depends on the type of update. ALCS supports three different types of update:

- Adding, deleting, and replacing resources
- Changing the characteristics of an existing resource
- Adding a resource type

ALCS Operation and Maintenance describes how to generate and load a communication load module.

2.8 Online Communication Table Maintenance (OCTM)

Online Communication Table Maintenance (OCTM) is an ALCS facility for managing the ALCS communication table. OCTM allows communications network changes to be defined in the ALCS communication table via an online process, eliminating the need to perform an offline communications generation.

The primary benefits of OCTM are:

1. Provides the ALCS end users with the ability to directly update the online communication table.
2. Reduces the offline communications generation process to the management of only non-terminal resources.
3. Eliminates the need for communication table consolidations.

ALCS customers who wish to use OCTM for managing their terminal and X.25 PVC resources must implement a Communications End User System (CEUS). The CEUS is a set of programs which provide a front-end to OCTM and which interface with OCTM via the `COMTC` Communication Table Update monitor-request macro. The CEUS will be unique for each ALCS system because it must provide functionality that meets the specific operational and network requirements of that ALCS system (and the users of that system).

A sample CEUS package can be obtained from IBM via the ALCS Web Site. This CEUS utilizes the ALCS 3270 mapping support, providing various screen maps that enable ALCS end users to submit online updates to the ALCS communication table via OCTM. ALCS users can modify the IBM CEUS so that it fits their specific requirements.

OCTM does not support the full range of communication resources that are supported in the ALCS communication table. OCTM is specifically designed to support all the terminal resource types, plus the X.25 PVCs which support terminal connectivity (X.25 PVC types 1, 6, and 7).

Chapter 3. ALCS data sharing and data management

There are a number of standard reference works that describe general database design techniques. Many of these design techniques are equally applicable to ALCS, but before applying them to ALCS you should be aware of some special characteristics of ALCS databases. (These special characteristics also apply to other members of the TPF family of products.)

3.1 Standard ALCS structures

The ALCS space-recovery facility (Recoup) supports a number of *standard* structures for lists, indexes, and so on. Use these standard structures, where possible, rather than inventing new ones.

3.2 TPFDF

Designing and coding application programs to access data through structures of lists, indexes, and so on can be a complex and time-consuming task. The amount of data that you store can change after a time. If it increases, you may need to introduce indexes to improve performance. If it decreases, the index structures may become an unnecessary overhead. The cost of changing existing application programs to do this can prevent you from optimizing your application's performance.

IBM's Transaction Processing Facility Database Facility (TPFDF) product simplifies the initial designing and coding of the application, and subsequent optimization. With TPFDF, your program stores and retrieves data without knowing the indexes, lists, and so on that TPFDF is using for the data. It also allows your data base administrator to change the structures (adding or deleting indexes, and so on) without requiring any changes to the application programs.

“Bibliography” on page 193 contains a list of TPFDF books.

3.3 Sharing data with non-ALCS applications

Some ALCS applications need to share data with applications that do not run under ALCS.

For example, an airline might have applications such as seat reservation, check-in, ticketing, and cargo running under ALCS. The same airline might have other applications such as flight planning, accounting, and frequent flyer database running under CICS, IMS, or other platforms.

Both the seat reservation and the flight planning applications might need access to information about existing flight schedules. The reservations and check-in applications may need access to the frequent flyer database, and so on.

ALCS supports a number of methods for sharing data. The best method depends on a number of factors, including:

- What platforms the different applications run on. Do all the applications run under z/OS, or do some run on other IBM or non-IBM platforms?
- Whether or not all the applications need absolutely up-to-date information.
- What performance characteristics are required. Do many ALCS transactions access the data (therefore you need fast access with a short path length) or few ALCS transactions (therefore performance is less important)?
- How complex the data is.

Figure 44 shows the four methods that ALCS uses to share data.

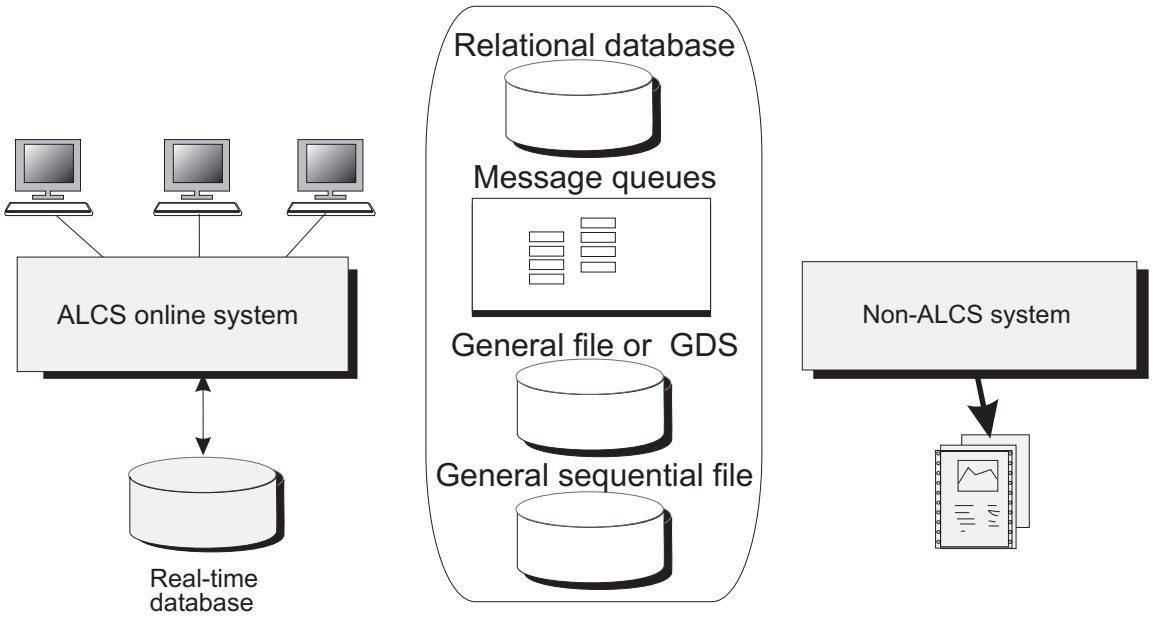


Figure 44. ALCS data sharing: Overview of the available methods

3.3.1 Relational databases

If you place the data in a relational database then both ALCS and non-ALCS applications can access the data using SQL.

Relational databases are a powerful and flexible way of storing and sharing data. They can also be distributed across a variety of platforms, including not only z/OS, but also OS/400 and OS/2.

Accessing relational databases imposes much greater overheads than accessing ALCS's own files. ALCS restricts the maximum number of transactions that can concurrently access relational databases.

Shared relational databases are probably suitable if:

- Applications on different platforms (including non-z/OS platforms) need to share the data.
- Both the ALCS and non-ALCS applications need access to absolutely up-to-date information.
- Only a small proportion of the ALCS transactions access the data.
- Both the ALCS application and the non-ALCS application require the sophisticated capabilities of SQL.

An example where a shared relational database might be suitable is an ALCS application that changes an airline flight schedule and a non-ALCS flight planning application. Only a very small proportion of the total ALCS transactions change the flight schedules. But those that do may need to interrogate the flight planning database in complex ways. Figure 45 shows how an ALCS application can use SQL to share data with a non-ALCS system.

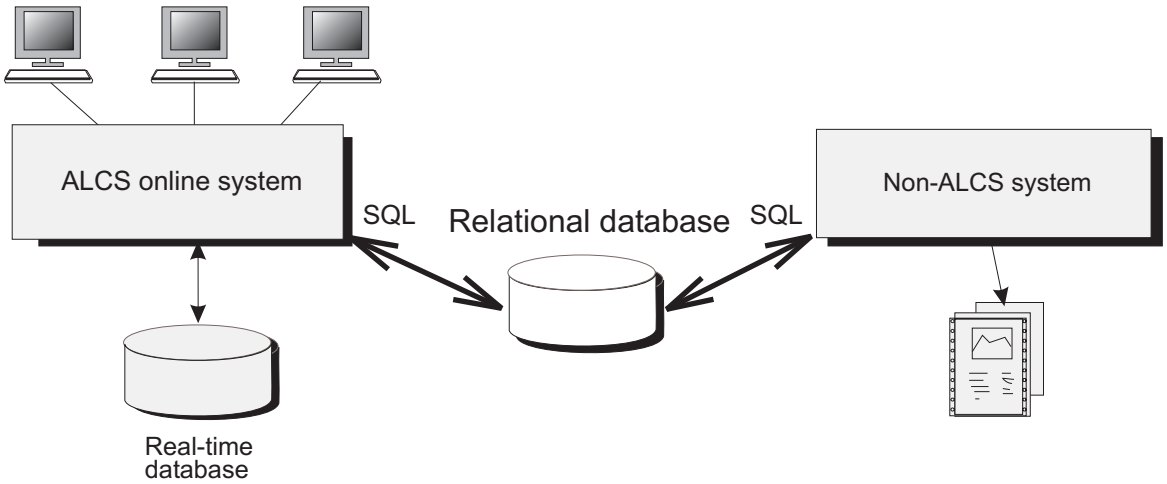


Figure 45. ALCS data sharing: Using a relational database

Note that although Figure 45 shows only two applications sharing the data, more than two applications can share it if required.

3.3.2 Real-time data export and import

An alternative to sharing a relational database is to transmit data between applications in real time – using, for example, WebSphere MQ. By transmitting data with WebSphere MQ, ALCS applications can provide non-ALCS applications with up-to-date information without incurring the overheads of direct updates to a relational database.

The non-ALCS applications can either use the data immediately or use it to update their own databases (which might be relational databases). Similarly, ALCS applications can receive data which they can either use immediately or use to update the ALCS database.

WebSphere MQ is a powerful and flexible way of transferring data between applications on a variety of platforms, including both IBM and non-IBM platforms.

Real-time data export and import are probably suitable if:

- Applications on different platforms (including non-IBM platforms) need to share the data.
- Both the ALCS and non-ALCS applications need access to absolutely up-to-date information.
- A large proportion of the ALCS transactions access the data.
- The ALCS application does not require the sophisticated capabilities of SQL.

An example where real-time data export might be suitable is an ALCS seat reservation or check-in application passing the distance travelled by an airline passenger to a non-ALCS frequent flyer application. Similarly, the non-ALCS frequent flyer application might pass information such as seating and meal preferences back to the ALCS check-in application.Figure 46 shows how an ALCS application can use WebSphere MQ to share data with a non-ALCS system.

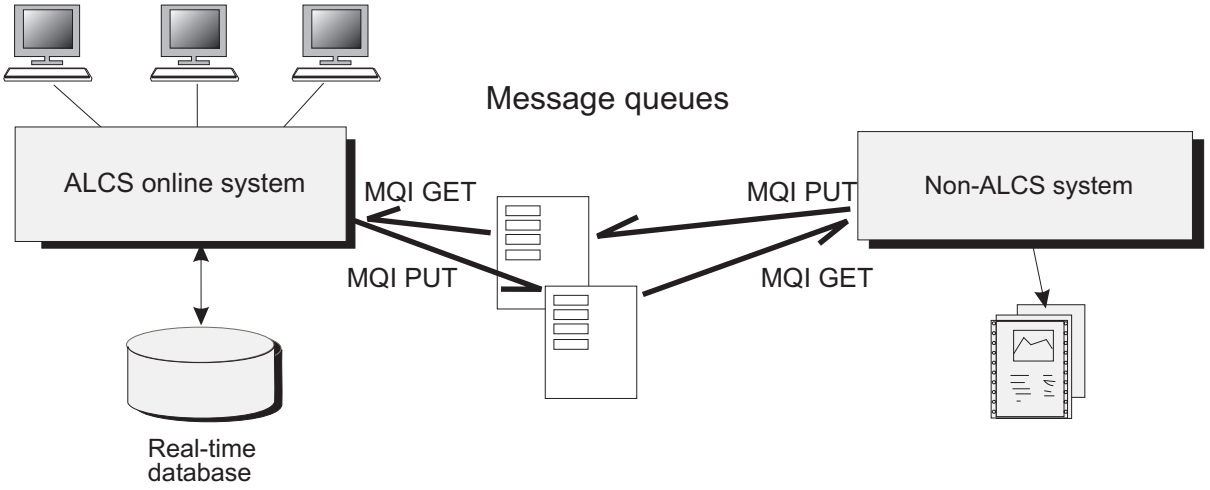


Figure 46. ALCS data sharing: Real-time import and export (MQI)

Note that although Figure 46 shows only two applications exchanging data, much more complex configurations are possible. For example, the message queue managers, which service the MQ GETs and PUTs, can create multiple copies of a piece of data (called *messages* in this context) placing each copy on a different queue. This allows an application to *broadcast* data to multiple destination applications.

3.3.3 Shared general file or GDS

ALCS can share access to general files (or GDSs) with other MVS applications. ALCS applications access these data sets using ALCS DASD I/O services. Non-ALCS application programs access them using the MVS VSAM access method.

Although it is possible for both ALCS and non-ALCS application programs to update general files (or GDSs), ALCS does not itself provide services to serialize accesses between ALCS and non-ALCS application programs. This means that you must design your applications so that:

1. Only the non-ALCS application program updates the data set, but ALCS application programs can read the data, or:
2. Only ALCS application programs update the data set, but the non-ALCS application program can read the data, or:
3. The ALCS and non-ALCS application programs use an agreed protocol to avoid conflicting or lost updates.

Even in cases 1 and 2, you may need to establish an agreed protocol to ensure that if one application program updates several records the other application program *sees* a consistent database.

Shared general files (or GDSs) are probably suitable if:

- Only z/OS applications need to share the data.
- Only one application (ALCS or non-ALCS) updates the information.
- A large proportion of the ALCS transactions access the data.
- The ALCS application does not require the sophisticated capabilities of SQL.

An example where a shared general file (or GDS) might be suitable is a non-ALCS application program that stores currency exchange rates that are used by ALCS applications.

Figure 47 shows how an ALCS application can use a general file or general data set (GDS) to share data with a non-ALCS system.

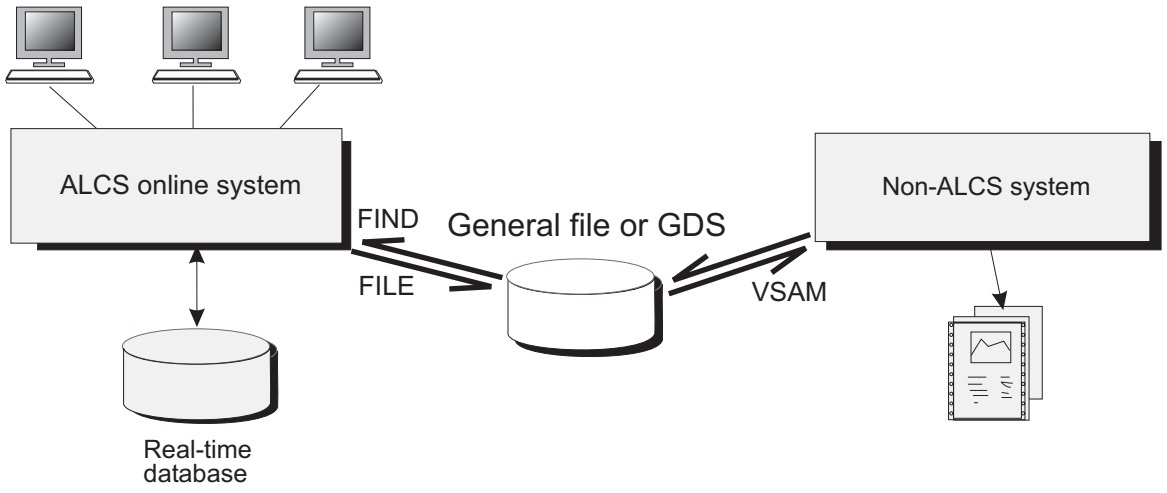


Figure 47. ALCS data sharing: Using a general file or GDS

3.3.4 Batch data export and import

For some application systems, ALCS and non-ALCS application programs need to access the same data but either the ALCS program or the non-ALCS program does not need absolutely up-to-date information. In these cases, the most efficient and simple way to share the data may be for one application program to create a file containing a *snapshot* of the data. Other application programs can then process the file without serializing access to the file.

An example where batch data export might be suitable is a non-ALCS application program that checks for airline passengers with similar or identical names who have reservations for the same flight – helping the airline to identify and delete duplicate reservations for the same passenger.

Many airlines implement this type of application by having the ALCS seat reservation application write details of each reservation to a sequential file. Later, the non-ALCS application program processes this file as a batch process (typically executed once each day).

Figure 48 shows how an ALCS application can use a general sequential file to share data with a non-ALCS system.

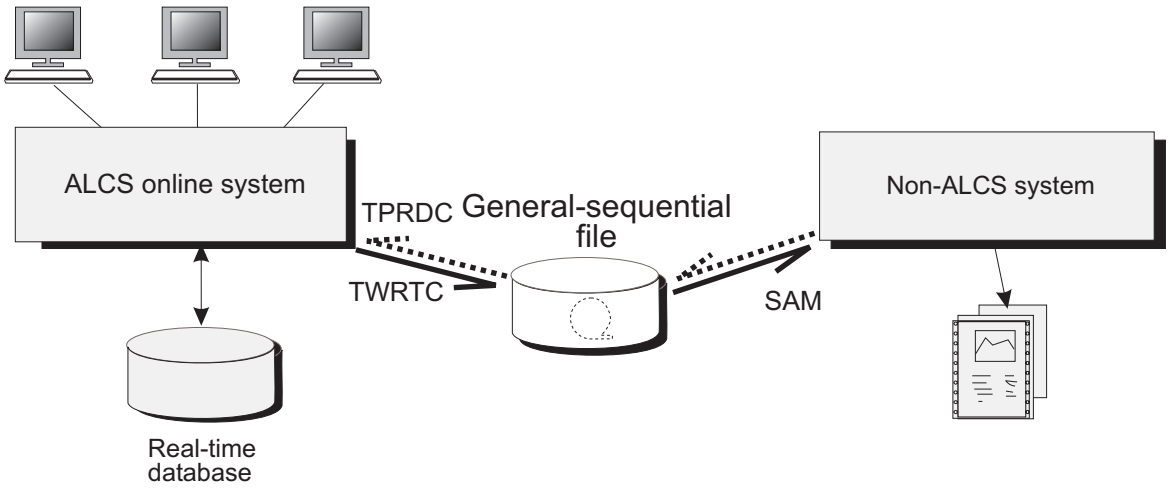


Figure 48. ALCS data sharing: Using a general sequential file

Chapter 4. ALCS database file management

This section describes the ALCS direct-access files. ALCS provides two types of direct-access file for application use:

- ALCS real-time database, for online transactions
- General-file, to transfer data between the online and the offline system. Application programmers further classify general files into:
  - General file
  - General data set (GDS)

Figure 49 shows the basic types of direct-access data sets that ALCS supports.

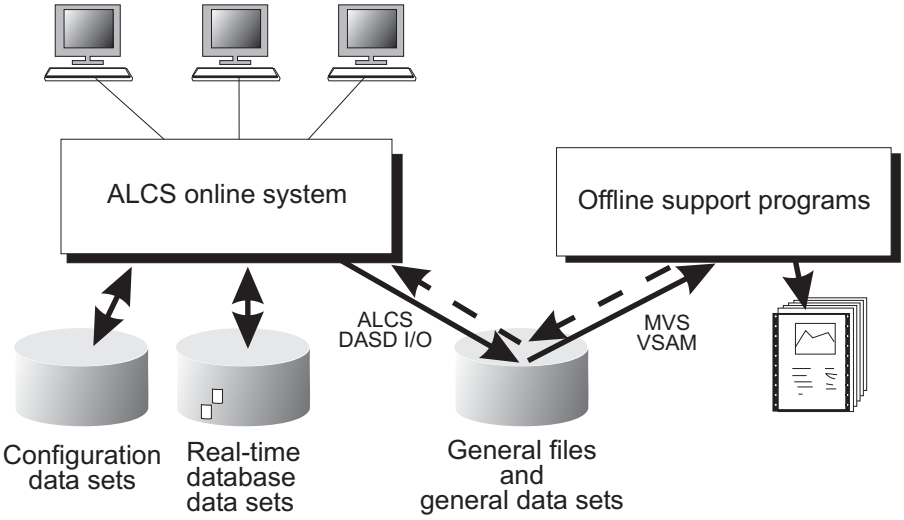


Figure 49. ALCS DASD files: Overview

ALCS uses a third type of direct access file to contain configuration information. Application programs cannot access the configuration data set.

Figure 50 shows where you can find more information about ALCS files and data sets.

Figure 50. Where to find more information about ALCS direct-access file management	
Type.	
Allocatable space	4.4, “Allocatable space overview” on page 75
Fixed files	“Fixed files” on page 63
General files	4.2, “General files and general data sets” on page 69
Pool files	“Pool files” on page 64
Configuration data set	4.9, “The ALCS configuration data sets” on page 89

4.1 The ALCS real-time database

ALCS applications that do not need to share data with non-ALCS applications normally keep data on the ALCS real-time database.

ALCS application programs can access real-time database records at all times. ALCS provides a number of facilities that make the real-time database particularly suitable for transaction processing applications, including:

- Duplication of data to protect against DASD equipment failure (this is sometimes called **mirroring**).
- Distribution of records across DASD actuators to avoid **hot spots** (this is sometimes called **striping**).
- In-memory caching of highly accessed records (VFA).
- Short processor pathlengths (few instructions) to access database records on DASD.

These facilities combine to provide high-speed data access with a high level of data integrity.

The real-time database consists of a number of data sets. There are one or more data sets for each record size, and there can be up to eight record sizes (L1 through L8). The system programmers chose these sizes when they install ALCS.

4.1.1 Organization of the database

The ALCS real-time database organization is designed to optimize DASD performance and avoid **hot spots**. ALCS allows you to have multiple data sets for each record size (and record type within this record size). Records of any one record type can reside on more than one DASD volume.

To help to balance the number of I/O operations (accesses) for any record type, ALCS spreads the records across these data sets. This means that there will be approximately the same number of accesses to each of the data sets for a record size.

4.1.2 Duplicated database

If you wish, you can use the dual copy facility of the IBM 3990 DASD controller to duplicate the ALCS database (and any other data sets) as well as (or instead of) the ALCS facilities.

The ALCS duplicated database facility

ALCS optionally supports a duplicated database. The DUPLEX parameter of the `DBGEN` macro specifies whether the database is duplicated.

A duplicated database has two copies of every real-time database data set, including any spill data sets. The two copies are called **copy 1** and **copy 2**.

A duplicated database provides some protection against the consequences of equipment failures. Without database duplication, ALCS can execute only when **all** real-time database data sets are available. With database duplication, ALCS can execute when **some** database data sets are not available, provided one copy of each data set is available.

You should allocate the copy 1 and copy 2 data sets on different DASD volumes (preferably attached by different paths) so that failure of a single hardware element does not make both copies unavailable. E.1, “How ALCS uses the duplicated database” on page 153 describes this facility in more detail.

4.1.3 Record classes – fixed file, short-term pool, and long-term pool

There are three different **classes** of records on the ALCS real-time database:

- Fixed file
- Short-term pool file
- Long-term pool file

Short-term pool file and long-term pool file are sometimes referred to collectively as **pool file** or just **pool**.

Fixed files

ALCS application programs access fixed files in much the same way that any applications access direct-access files (sometimes called **random-access** files).

ALCS application programs do not use data set names or file names for fixed files. Instead, they use a special token called the:

Fixed-file record type in ALCS (the FACE ID in TPF).

To access a particular record, the application specifies the file (by type) and a relative record number. The relative record number is called the:

Fixed-file record ordinal number in ALCS (the FACE ordinal in TPF).

Before actually reading or writing a fixed-file record, ALCS application programs use an ALCS service to convert the fixed-file record type and ordinal into a 4-byte token called the **file address**. Figure 51 shows the ALCS fixed files and file address.

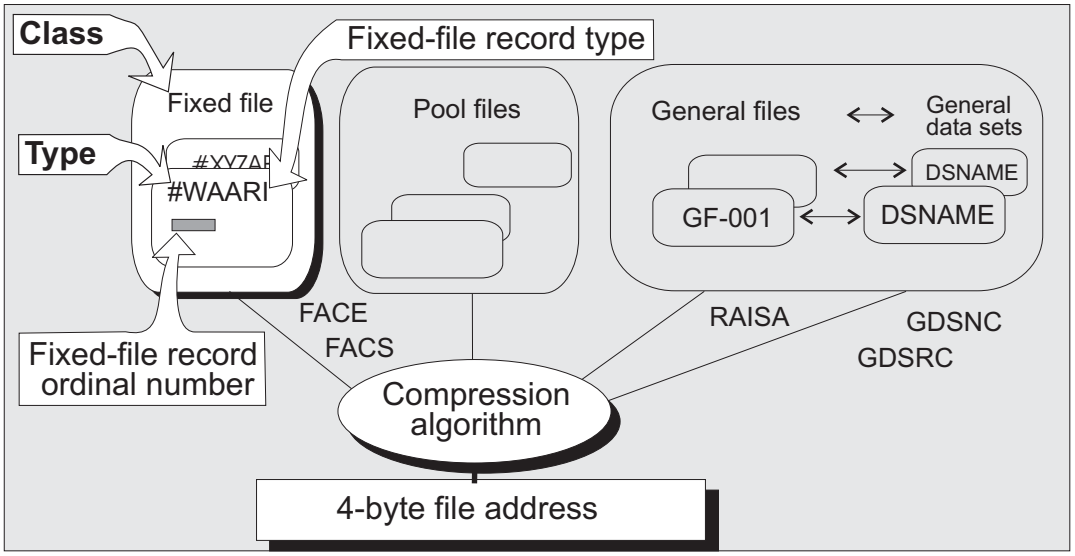


Figure 51. ALCS direct-access files: Fixed files

ALCS application programs cannot create or delete fixed files, and they cannot change the number of records in a fixed file. The system programmer or database administrator makes these changes.

Miscellaneous file: One type of fixed-file record that an installation often defines is a **miscellaneous file**. A miscellaneous file contains fixed-file records that can be made available to applications without system programmer having to define a new fixed-file record type. The system programmer allocates a range of ordinal numbers to one application and other ranges of ordinal numbers to other applications.

Miscellaneous files usually have names (fixed-file record type symbols) such as #MISC1, #MISC2, and so on, where the last digit in the name indicates the record size (L1, L2, and so on).

Pool files

In addition to the fixed files, ALCS supports a number of large pools of records. ALCS application programs cannot specify the file and the relative record number of a particular pool file record. ALCS application programs must use an ALCS monitor service to acquire a record from one of these pools. This service is called **dispense**.

The monitor service returns a file address that the application stores as a pointer in another record. For example, an application program can use a pool-file record as an overflow record by storing its file address in the prime record. 4.1.6, “Overflow and chaining” on page 66 describes this process. Alternatively, 4.1.7, “Lists and indexes” on page 67 describes how an application program can store the file address in a list or index record.

Note: It is not important to an application program **which** pool record ALCS allocates. It only important that the record is not already in use for some other purpose.

Subsequently, application programs access the record using the stored file address.

When the data contained in the record is no longer needed, the application program clears the saved file address to zeros and uses an ALCS monitor service to return the record to the *available* pool. This service is called **release**.

Note: If the file address is saved in more than one place, the application must clear **all** of them.

Short-term pool file: Application programs can use short-term pool-file records to store data for short periods of time (a few seconds or minutes).

An application must release a short-term pool record within (at most) a few hours after the dispense. If the application does not release a short-term pool record within a reasonable time (typically 24 hours) then ALCS assumes that there is a programming error and releases the record itself.

The actual amount of time before ALCS itself releases a short-term pool record depends on the rate at which the short-term pool is dispensed and released.

Your system programmer or database administrator can allocate one short-term pool file for each record size. For example, your installation may have four short-term pool files, one for each of the sizes L1, L2, L3, and L4.

Long-term pool file: Application programs can use long-term pool-file records to store data for long periods of time (days, weeks, or years).

Like short-term pool-file records, the application should release a long-term pool record when the information it contains is no longer required. There is no time limit within which your application must release a long-term pool record. A long-term pool record does not become available immediately after the release, it is available for reuse only when the ALCS space-recovery utility (Recoup) confirms that there are no pointers to the record saved in the ALCS database.

Your system programmer or database administrator can allocate one long-term pool file for each record size. For example, your installation may have four long-term pool files, one for each of the sizes L1, L2, L3, and L4.

Minimum pool-file record requirement for ALCS: ALCS requires a minimum allocation of the L3LTPOOL and L3STPOOL. *ALCS Installation and Customization* lists all the record requirements for ALCS

4.1.4 Long-term pool integrity

Applications must use long-term pool-file records to store data that has a life longer than a few minutes. ALCS protects long-term pool-file records against possible loss or damage. Such loss or damage can arise in a number of ways as follows:

- Equipment failure can prevent the correct update of the DASD copy of a pool file directory record. If this happens, the directory can indicate some records as available when they are actually in use. If ALCS dispenses these records again, the application loses the data they contain.
- Equipment failure or program errors can result in some records never being released. Without some method of recovering these records (lost addresses), the number of available records in the pool could eventually reduce to zero. ALCS does not redispense these records (as it does with short-term pool records).
- Application program errors can result in the release of some records even though the application (or other applications) still requires the data they contain.

4.1.5 Pool dispense rate monitor

The pool file management routines monitor the dispense rates of the long-term pools. If any dispense rate exceeds the appropriate pool dispense rate threshold value, they send an **Attention** message to the RO CRAS.

Each time ALCS calculates a long-term pool dispense rate, it also calculates how long the available file records in the pool can sustain that rate of dispense. If the time until pool depletion drops below the appropriate threshold value, the pool file management routines send an **Attention** message to the RO CRAS, stating the estimated time until pool depletion.

You can use the `ZPOOL` command to review the threshold values for dispense rates and depletion times set by ALCS and, if necessary, to alter them. See *ALCS Operation and Maintenance* for a description of the `ZPOOL` command.

4.1.6 Overflow and chaining

In many cases, the amount of data you want to hold in a record varies dynamically. For example, an airline seat reservation application might keep a list of the passengers booked on each flight. This list will grow dynamically as more and more passengers make reservations for the flight.

Standard forward chaining

To allow for this, it is usual to start building the list in one record (the **prime record**). When this record is full, the application obtains an additional record (an **overflow record**) in which to continue the list. This process can continue for as long as required. As each overflow record fills, the application obtains another overflow record. This process is called **standard forward chaining**.

The prime record and its overflow records are usually chained together by storing a pointer to the first overflow record in the prime record, storing a pointer to the second overflow record in the first overflow record, and so on. The actual pointers are 4-byte tokens called **file addresses** (file addresses are explained in more detail in 4.3, “Record addressing” on page 70) You can conveniently indicate the end of a chain by setting the pointer in the last record to binary zeros (which is an invalid file address).

Figure 52 shows how an application chains overflow records.

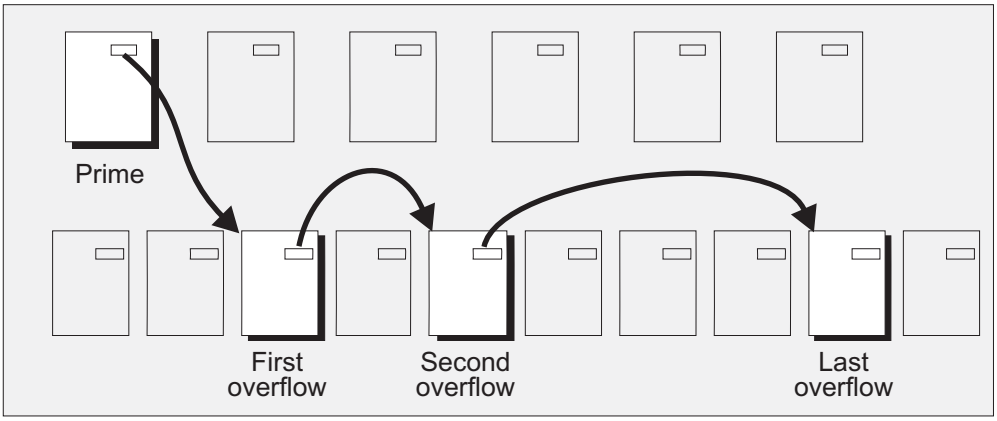


Figure 52. ALCS direct-access files: Standard forward chains

Note: Figure 67 on page 81 shows where the forward-chain pointer is stored in a record header.

Standard backward chaining

There are two potential problems with standard forward chaining:

- To add an item to an existing chain, the application program must read all the records in the chain. For chains which contain many records this can be a substantial overhead and can degrade the performance of the application.
- If any record in the chain is overwritten (for example by program error) it is impossible to find the remaining records in the chain.

A common way to avoid these problems is to use **standard backward chaining** (as well as forward chaining). The prime record holds a pointer to the **last** overflow record. The first overflow record holds a pointer to the **prime** record and so on.

Figure 53 shows how an application creates forward and backward chains.

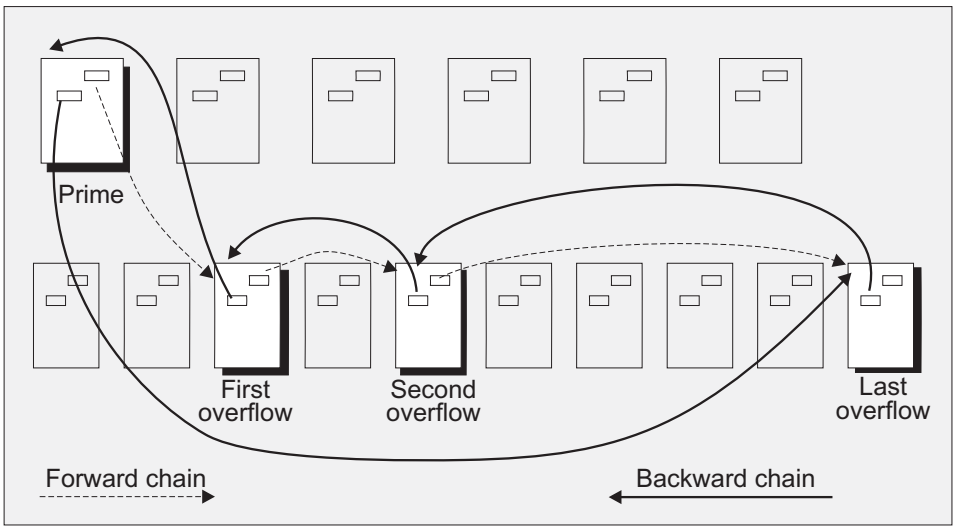


Figure 53. ALCS direct-access files: Backward chains

Note: Figure 67 on page 81 shows where the backward-chain pointer is stored in a record header.

By convention, if there are no overflow records both the forward and backward chain fields in the prime record contain binary zeros.

By using standard backward chaining, an application program can locate the last overflow record using the pointer stored in the prime record – this avoids reading all the records in the chain.

Also, if one record in the chain is overwritten, it is possible to find all the records before the corrupted one by following the forward chain pointers, and all the records after the corrupted one by following the backward chain pointers.

4.1.7 Lists and indexes

You can use chains to create list structures. For example, you can hold a **list** of names in a chain of records (prime and overflow). Each entry in the list points to another record.

Figure 54 on page 68 shows a list structure where names are grouped alphabetically (but not sorted) in chain. Each entry contains a pointer to a record which contains details about that person. One of the records has an overflow record to contain more information. This structure shows standard forward chaining for both the list records and the detail records. You could also use standard backward chaining for either (or both) if required.

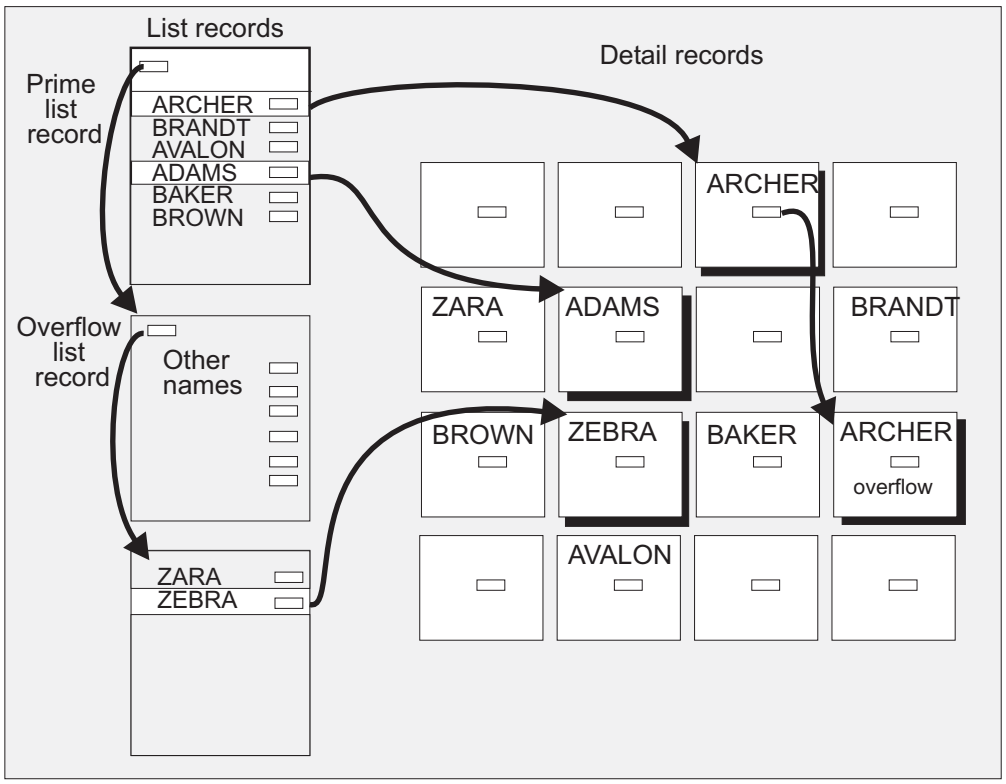


Figure 54. ALCS direct-access files: Lists

If the list contains many names (and pointers) an application program may need to read the prime list record and many overflow list records before it finds the pointer for the required detail record.

You may be able to reduce the number of reads by using an **index** structure. Figure 55 on page 69 shows an index structure based on the 26 letters of the English alphabet. Each of the 26 entries on the index record contains a pointer to a list record which contains all the names that begin with the letter. The list records contain a pointer to the detail records.

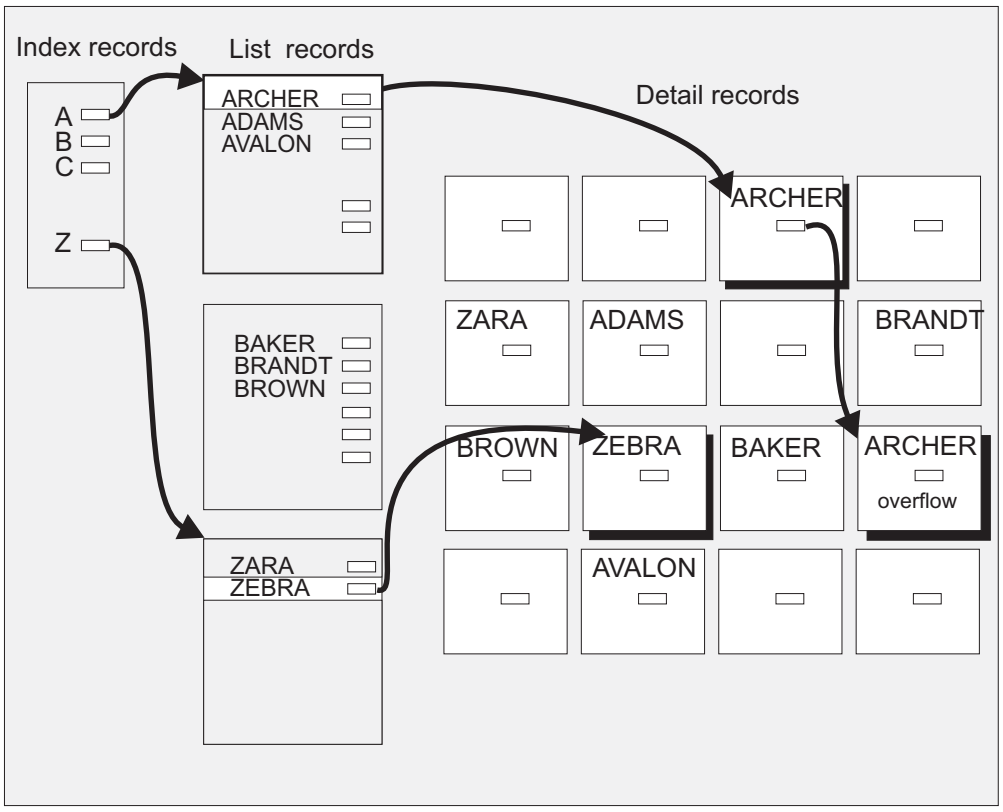


Figure 55. ALCS direct-access files: Indexes

4.2 General files and general data sets

General files can be used to transfer data to or from application programs that do not run under the control of the ALCS monitor. 3.3.3, “Shared general file or GDS” on page 58 gives an overview of sharing files.

General files are **single-extent** VSAM entry-sequenced data sets. There can be a maximum of 256 general files, each identified by a decimal number in the range 0 through 255.

Note: ALCS reserves general file 0 for its own use.

There is one data set for each ALCS general file. Application programs can access a general file only if the data set is allocated to ALCS. In some predecessor systems, general files and general data sets are different and application programs must access a file as either a general file or a GDS. In ALCS they are physically identical, but ALCS supports both methods of access. Application programmers identify:

- General files by the **general file number**
- General data sets by the **data set name**

However to the system programmer general files and general data sets are the same. In this book the term **general file** is used to mean either general file or general data set.

ALCS provides the ZDASD command to allocate and deallocate general files and data sets. *ALCS Operation and Maintenance* describes the ALCS ZDASD command.

4.3 Record addressing

ALCS application programs use a 4-byte **file address** to refer to records in:

- The real-time database
- General files or general data sets (GDS)

This file address is different from ALCS/VSE and TPF file addresses (see 4.3.3, “Multiple file address format support” on page 73).

TPF compatibility

Applications that require compatibility with TPF can use an 8-byte file address, by using a DECB data level. ALCS applications that use a DECB data level must use an 8-byte file address in 4x4 format. You can obtain an 8-byte file address by specifying a DECB address when you request a monitor service to get a file address, or by requesting the FA4X4C monitor-request macro (tpf_fa4x4c C function) to convert an existing 4-byte file address to an 8-byte file address in 4x4 format. For more information about the 8-byte file address in a DECB see *ALCS Application Programming Guide*.

4.3.1 Constructing the file address

ALCS provides monitor services that application programs can use to construct the file address from:

Class Fixed file or general file (or GDS)
Type Fixed file record type or general file number
Ordinal Record ordinal (relative record number within type).

Notes:

1. Application programs do not construct a file address for pool file records. Application programs request a pool-file record and ALCS gives the file address.
2. Application programs do not construct a file address for configuration data sets or system fixed-file records. These records are not intended for application use.

General files and general data sets (GDS) are the same thing. Applications that use RAISA call them general files, applications that use GDSNC and GDSRC call them general data sets. Figure 56 on page 71 summarizes the ALCS class, type, and ordinal to file address compression technique.

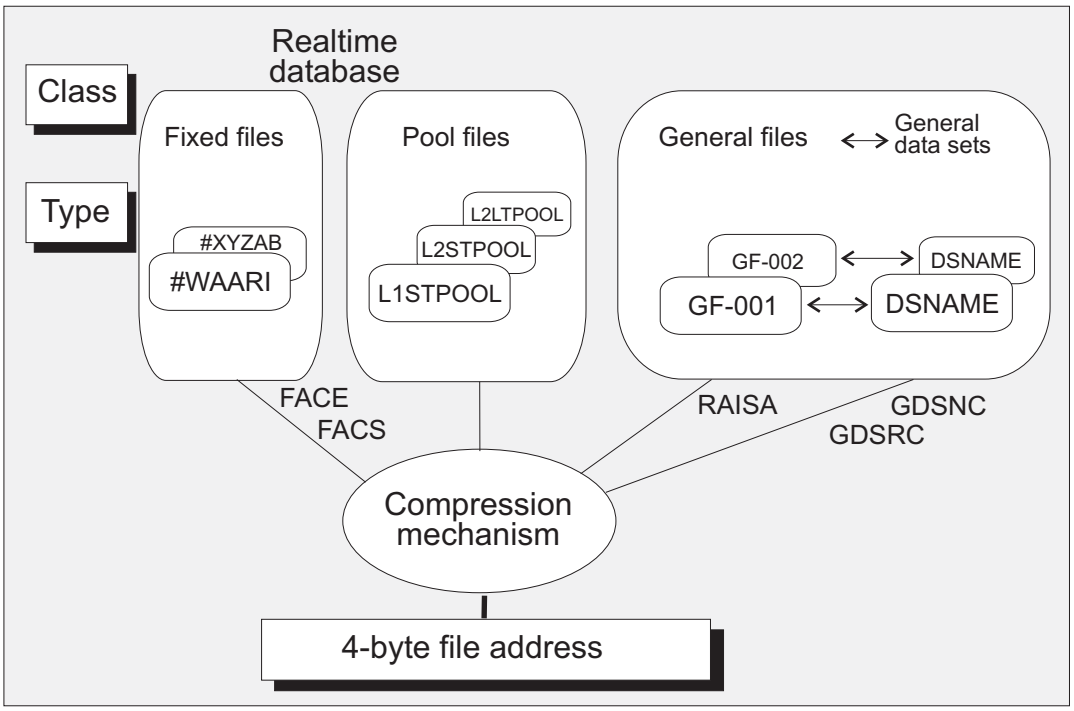


Figure 56. ALCS file address: Compressing the class, type, and ordinal

ALCS provides the `RONIC` monitor-request macro to determine the class, type, and ordinal from the file address.

ALCS also provides C language functions that perform similar services. See *ALCS Application Programming Reference – C Language* for further details.

4.3.2 File address format

The file address format that ALCS uses is called the **band format**. A band format file address consists of two parts (that is, two binary numbers). The two parts are the band and the band ordinal. For any one band, there is a maximum band ordinal. Different bands can have different maximum band ordinals.

The ALCS generation allocates one or more bands for each record type. It never allocates the same band to more than one record type. The ALCS generation must allocate more than one band if the number of records of that type is greater than the maximum band ordinal for the first band.

If there is only one band for a record type, ALCS constructs the file address using the band, with the band ordinal equal to the record ordinal. If there is more than one band for a record type, ALCS uses the first band for record ordinals up to the maximum band ordinal, and it uses the second band with band ordinals starting from 0, and so on.

Suppose, for example, that you decide to allocate band number 4387 (`X'1123'`) to a particular record type. If there are 5000 records of this type, then ALCS forms the file addresses like this:

Record Ordinal	Band	Band ordinal
0	X'1123'	0
1	X'1123'	1
2	X'1123'	2
...	...	...
4999	X'1123'	4999

Note that in this case, band ordinals greater than 5000 are unused – the corresponding file addresses are not valid.

Band number 4387 has a maximum band ordinal number of 65 535 – allowing up to 65 536 records. If there are 80 000 record of the type then you might allocate two band numbers, for example 4387 and 4388 (X'1123' and X'1124'), then ALCS forms the file addresses like this:

Record Ordinal	Band	Band ordinal
0	X'1123'	0
1	X'1123'	1
2	X'1123'	2
...	...	...
65 535	X'1123'	65 535
65 536	X'1124'	0
65 537	X'1124'	1
...	...	...
79 999	X'1124'	14 463

Allocating bands

ALCS Installation and Customization describes how to select and allocate ALCS bands.

How bands and band ordinals appear in the file address

The ALCS file address contains the 1, 2, or 3 bytes of the band *in reverse order* starting at byte 3 of the file address. The remaining bytes of the file address are the band ordinal as an unsigned binary number (see Figure 57 on page 73).

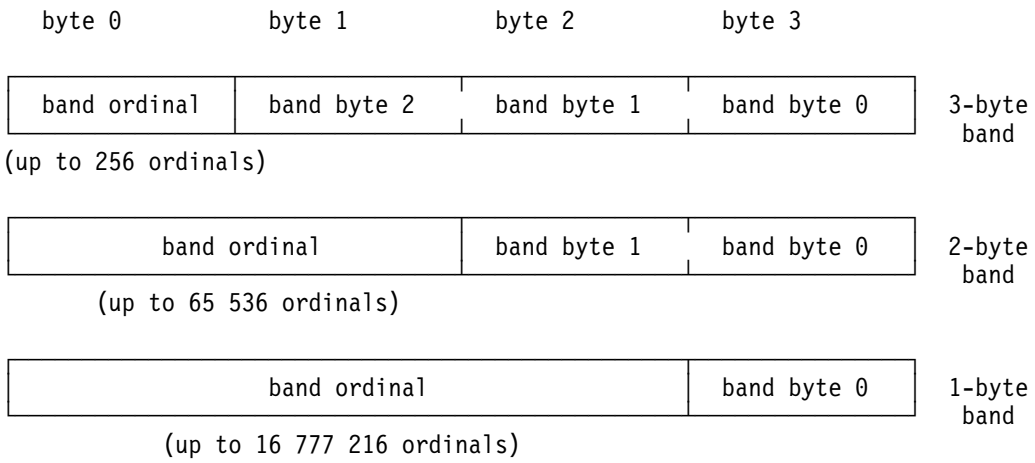


Figure 57. ALCS file address formats

For example, if you decide to allocate band number 4387 (X'1123') to a record type, then ALCS forms the file address for the record ordinal number 500 (X'1F4') as follows:

Band	Band ordinal	File address
X'1123'	X'1F4'	X'01F42311'

4.3.3 Multiple file address format support

Records in an ALCS database can contain references to other records. These references are usually file addresses. Typical examples include chains of pool records, where each pool-file record contains the file address of the next pool-file record in the chain. It is possible, but less common, for a record to contain the file address of another fixed file or general file record.

When a database migrates from TPF or ALCS/VSE, the records contain file addresses that are valid in the *migrate-from* database (TPF or ALCS/VSE).

TPF and ALCS/VSE do not use the ALCS file-address format (band format), they use:

- File-address format 3 (FARF3), 4 (FARF4), and 5 (FARF5) for TPF
- Record-ordinal-number (RON) format file addresses for ALCS/VSE

TPF also supports two obsolete file address formats that can coexist with FARF3, they are:

- Module, cylinder, head, record (MCHR)
- Symbolic ordinal number (SON)

Some TPF users implement other file address formats. Figure 58 on page 74 summarizes the different file addressing schemes that have evolved.

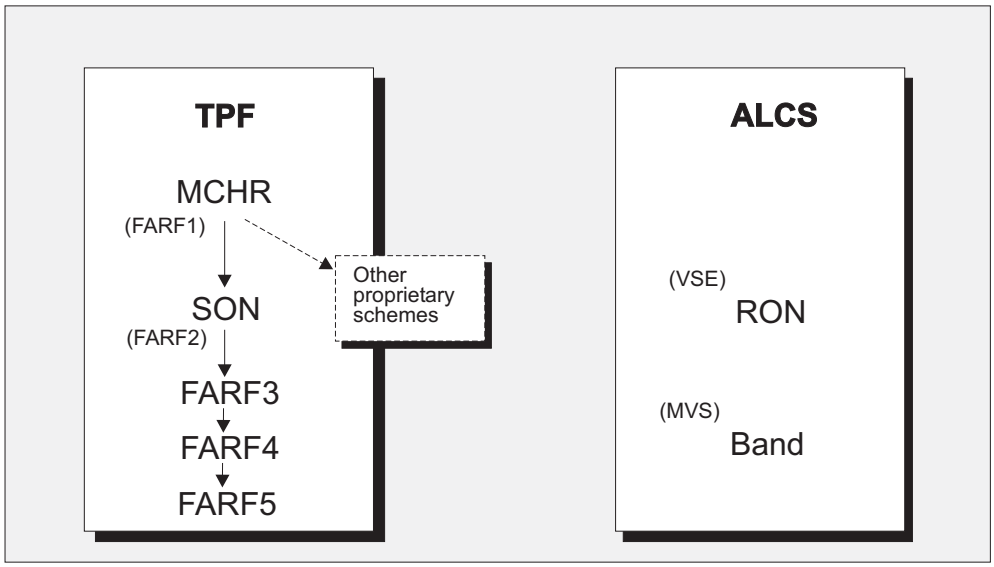


Figure 58. File address: Different formats for TPF and ALCS

To simplify migration from TPF and ALCS/VSE, ALCS can support more than one file address format. That is, at the same time it can support the ALCS band format file address, and **one** of:

RON ALCS supports ALCS/VSE RON format file addresses, provided that the fixed-file type values are the same for ALCS/VSE and ALCS. If the fixed-file type values are different, ALCS can support the file addresses only as user-defined format (see below).

FARF3 ALCS supports TPF FARF3 format file addresses provided that:

- The fixed-file type values are the same for TPF and ALCS.
- Fixed file FARF3 addresses contain the fixed-file type in bits 1 through 12.
- There are no 4KB pool FARF3 addresses

If not, ALCS can support the file addresses only as user-defined format (see below).

user Any user-defined format (or TPF FARF4 and FARF5 file address formats). ALCS can support:

- A file address format that coexists with the ALCS band format.
- A file address format that cannot coexist (for example FARF4).

ALCS supports these by calling an installation-wide exit routine. *ALCS Installation and Customization* describes how to use the USRFAR installation-wide exit routine.

File addresses returned to applications

The `DBGEN` generation macro specifies the file address formats ALCS supports.

Applications normally use band format file addresses. `ENTRC FACE`, `GETFC`, `RAISA`, and so on always return the band format file address. However FIND and FILE macros, `RELFC`, and so on, can use either the band format file address or another file address format (for example, FARF3).

The record hold facility works correctly with multiple file address formats. That is, a record hold using one file address format correctly, holds against another record hold that uses a different file address format for the same record (see E.3, “Record hold facility” on page 155).

Because ALCS monitor-request macros can use both file address formats, almost all ALCS functions can specify either format. For example, the ALCS ZDFIL and ZAFIL commands can specify either file address format. The ALCS ZDATA and ZRSTR commands can load data from magnetic tapes that ALCS/VSE or TPF create (provided that the format is compatible).

TPF duplicated and nonduplicated pools

TPF supports three pool types for each record size: short-term, long-term duplicated, and long-term nonduplicated. ALCS/VSE and ALCS only support two pool types, short-term and long-term.

ALCS multiple file address format support interprets both long-term duplicated and long-term nonduplicated FARF3 addresses as the same type – long-term. To do this, ALCS **partitions** long-term pool into two ranges of ordinal numbers. It then uses one range for long-term duplicated FARF3 ordinals, and the other for long-term nonduplicated FARF3 ordinals. The POOLMIG parameter of the DBGEN macro controls this partitioning.

4.4 Allocatable space overview

Figure 59 on page 76 shows (schematically) the initial allocation for:

- Fixed file
- Short-term pool
- Long-term pool

| For all releases after ALCS Version 2 Release 1.1, this is the way the ALCS generation allocates your database initially (before you add or delete records). If you are migrating from a predecessor ALCS version (or release), this layout is identical to your existing (predecessor) layout.

Figure 59 on page 76 shows some space that is not available (addressable). The extra space exists because the VSAM clusters which contain the ALCS database comprise an integral number of DASD cylinders. The cluster allocation process rounds up the number of records defined in the ALCS generation to the next higher whole number of cylinders.

1.6.1, “How ALCS stores DASD records” on page 32 describes how ALCS uses VSAM control intervals. E.3.3, “Data sets” on page 157 describes how ALCS uses VSAM clusters.

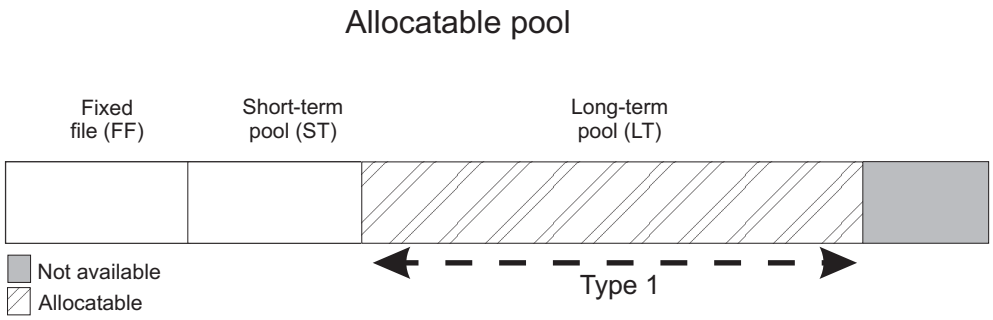


Figure 59. Allocatable pool: Initial allocation

Figure 60 shows the following records dispensed from long-term (LT) pool:

- System fixed-file dispensed for ALCS purposes
- Fixed-file records
- Short-term pool

It also shows some fixed-file records which are marked for deletion. These records cannot be recovered (undeleted) until they are purged.

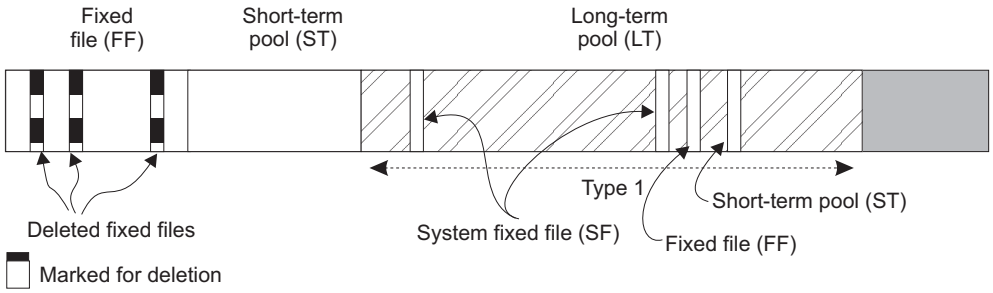


Figure 60. Allocatable pool: Dispensing from LT-pool

Figure 61 shows the situation when the fixed-file records are purged from the database. This database space is not available for re-use until you start to use type-2 long-term pool support.

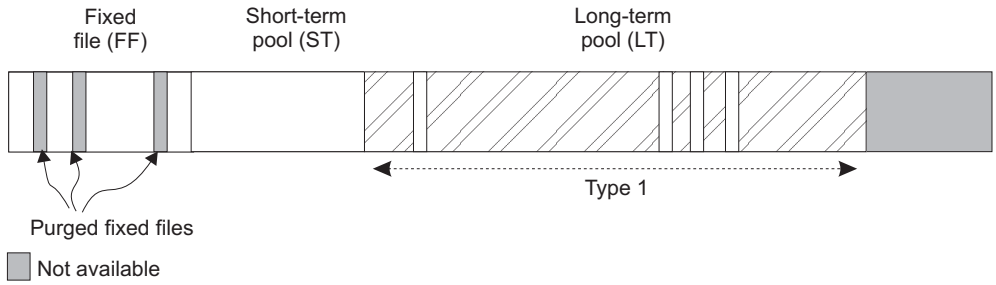


Figure 61. Allocatable pool: After some fixed files are deleted (and purged)

When you change to type-2 dispensing, the previously-unavailable space (including the space occupied by the purged fixed-files) is now available (allocatable). Figure 62 on page 77 shows this new allocatable space.

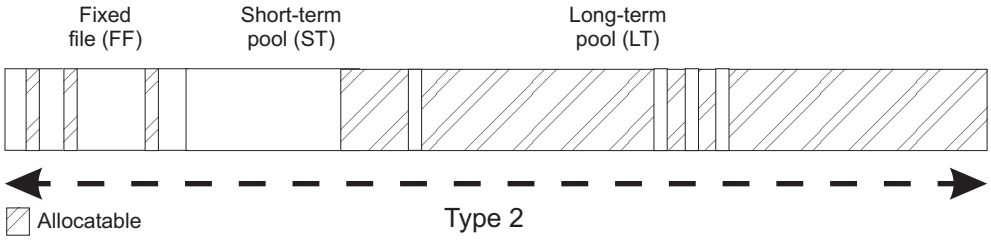


Figure 62. Allocatable pool: After changing to type-2 dispensing

ALCS type-2 long-term pool support uses the same dispense mechanism to obtain records for **any** class of ALCS record on the real-time database. ALCS does not use contiguous space for each class of record, it treats records in allocatable pool as either:

- Allocatable (available for dispensing)
- In use for one of the following:
  - Fixed file (FF)
  - Short-term pool (ST)
  - Long-term pool (LT)
  - System fixed file (SF)

When ALCS dispenses a record from allocatable pool, it marks the record as **in use** for that class.

Note: Figure 63 shows the same database as Figure 62, but from the perspective of records in use.

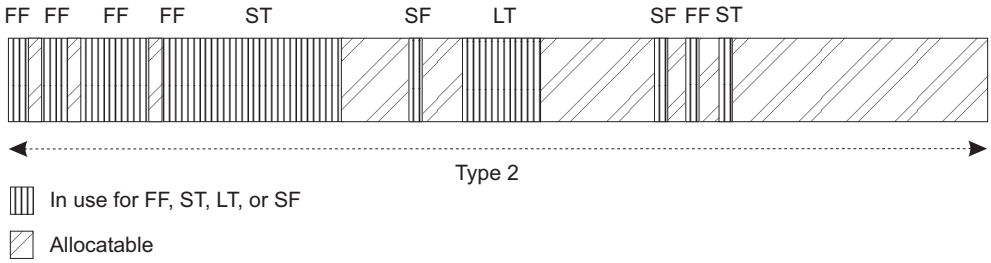


Figure 63. Allocatable pool: Records in use

Figure 64 shows how the record classes are distributed over the allocatable pool space. In particular it shows:

1. Long-term pool records in what were previously fixed-file records.
2. Fixed-file and short-term records in what were previously long-term pool records.
3. Short-term pool records in space that was previously not available.

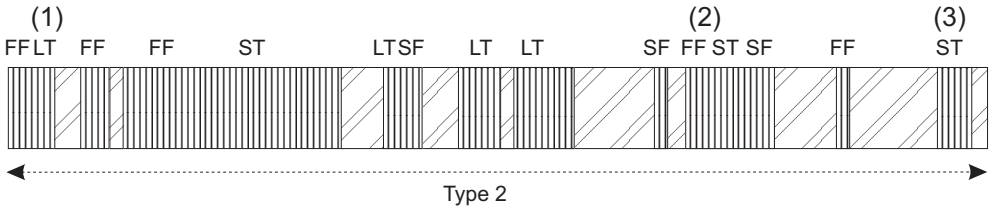


Figure 64. Allocatable pool: Using and reusing allocatable pool

4.4.1 Algorithm-based addressing

Predecessor ALCS systems use an algorithm to convert a file address to a data set name, and a relative byte address (RBA) within the data set to perform an I/O operation.

ALCS continues to support algorithm addressing. The algorithm uses the DASD configuration tables produced by the ALCS DASD generation. Figure 65 shows algorithm addressing.

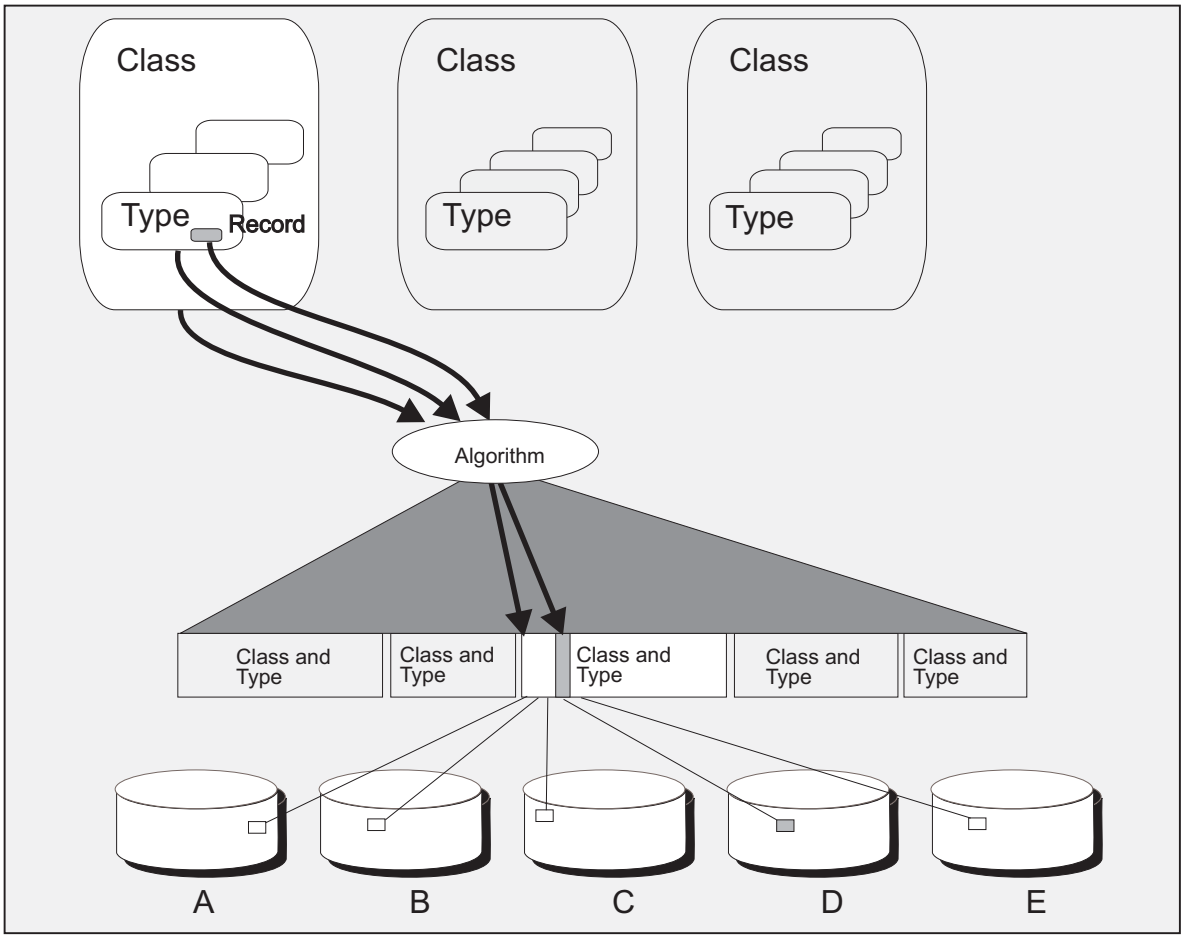


Figure 65. Algorithm file addressing: Class, type, and ordinal to data set and RBA

4.4.2 Table-based addressing

ALCS uses tables to convert file addresses into the corresponding data set and RBA information to locate the physical record.

Note: ALCS still supports algorithm-based addressing, but uses tables for added records. Figure 66 gives an overview of the two types of file addressing.

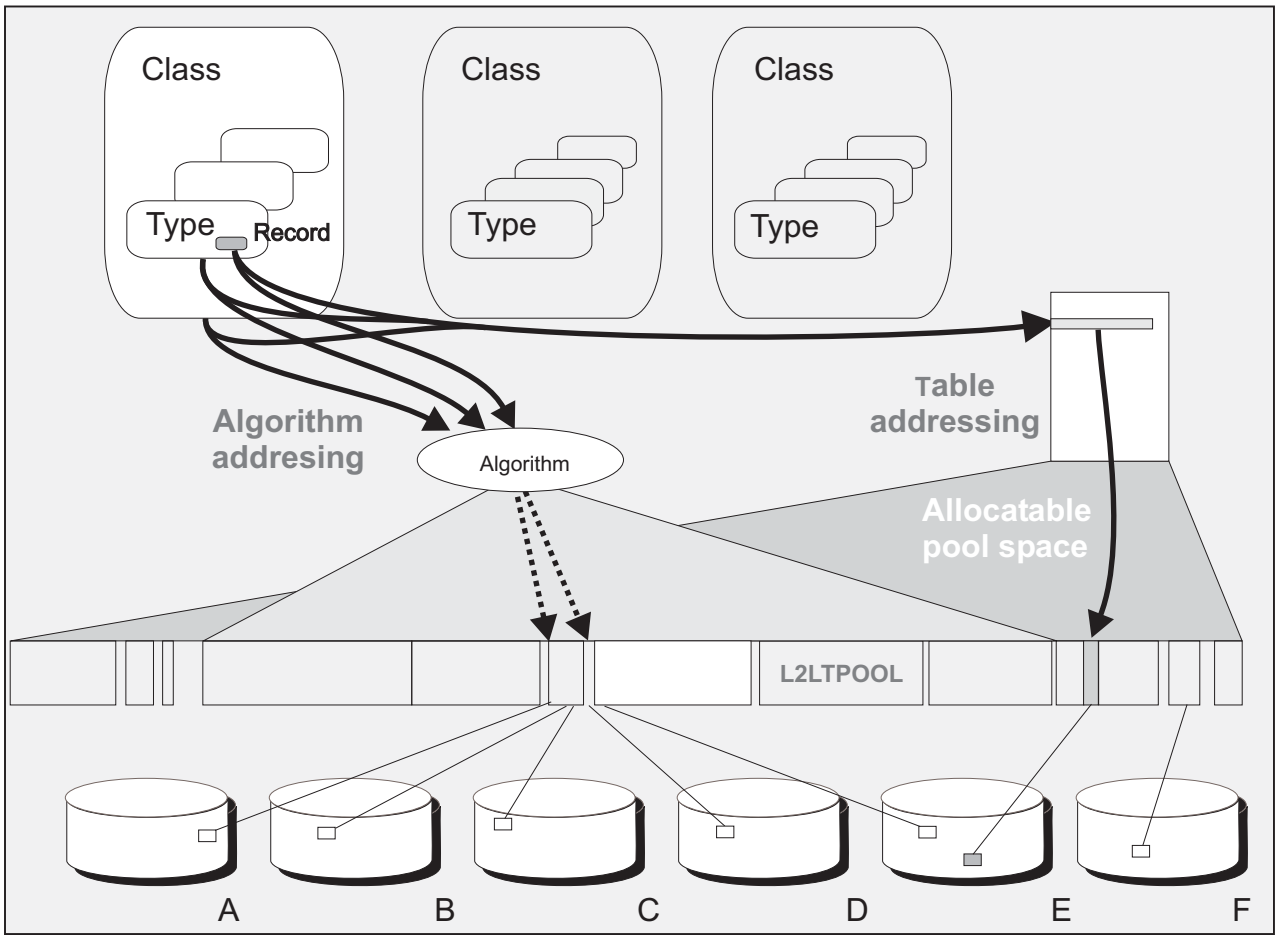


Figure 66. Table-based file addressing: Class, type, and ordinal to data set and RBA

Every record on the real-time database has an **allocatable-pool** file address. These are conventional ALCS band-format file addresses with one or more bands defined for each record size. There is, of course, only one allocatable pool **type** for each record size.

Note: When you are migrating from ALCS/MVS/XA or ALCS Version 2 Release 1.1, you must define additional bands for the allocatable-pool types (sizes).

The index tables exist in memory and are built up using:

- Fixed-file directories
- Segment tables

Fixed-file directories

The fixed-file directories exist on the real-time database as chains of allocatable-pool records. Application programs cannot access these system records. Any attempt is treated as an invalid file address. When ALCS is executing, they also exist as in-memory tables.

Some of the real-time database records are used for fixed-file, short-term pool, and system records. Application programs access these records using the corresponding fixed-file or pool-file address. If an ECB-controlled program attempts to update a fixed-file or short-term pool record using its allocatable-pool file address, ALCS treats this as an invalid file address.

System records cannot be updated by ECB-controlled programs. If an ECB-controlled program attempts to update a system record using its allocatable-pool file address, ALCS treats this as an invalid file address.

The remainder of the real-time database records are available for use as long-term pool records. Application programs can access these records using the allocatable-pool file address. Databases migrated from ALCS/MVS/XA or ALCS Version 2 Release 1.1, require the following special considerations:

- The existing records contain embedded references to long-term pool. ALCS allows application programs to access these records using the algorithm file addresses (ALCS/MVS/XA or ALCS Version 2 Release 1.1).
- Fall back to the predecessor system (ALCS/MVS/XA or ALCS Version 2 Release 1.1). Long-term pool file records can be dispensed with the predecessor system file addresses until fallback capability is no longer required.

Fixed-file directories (which are also used for short-term pool) map each valid fixed-file or short-term pool file address into the corresponding allocatable-pool file address.

Segment tables

In TPF and previous versions and releases of ALCS, addressability of records on the real-time database is a natural consequence of the algorithms used to convert file addresses.

For old fixed-file and pool-file records (ALCS/MVS/XA or ALCS Version 2 Release 1.1), the conversion uses algorithm file addressing. For fixed-file and short-term pool records added since migration, the conversion uses the fixed-file directories.

The allocatable-pool record ordinal numbers are then mapped on to physical record locations using segment tables. The segment tables map **segments of addressability** on to portions of the database data sets. Each segment of addressability comprises 65 536 consecutive allocatable-pool ordinal numbers, and maps onto 65 536 consecutive records within one database data set.

This addressing scheme has the following important effects:

- Addressability can be allocated independent of the actual number of physical records. Normally, at least enough addressability must be allocated to equal or exceed the number of physical records. But excess addressability can be allocated to allow for planned increases in the number of records (any file address that does not correspond to a real physical address is treated as invalid).
- Long-term pool dispense cannot dispense allocatable pool by scanning consecutive ordinal numbers (like ALCS Version 2 Release 1.1 does). Such a mechanism would create unacceptable hot spots because consecutive dispenses would come from the same data set. Instead, it explicitly **strobes** the database data sets so that consecutive dispenses come from different data sets.
- Adding addressability in the form of additional data sets can have the effect of unbalancing the even distribution of records previously provided by striping. To correct this there is a restripe facility (`ZDASD STRIPE`) which redistributes parts of the data base evenly over all the real-time datasets.

4.5 Record header

Each DASD record contains a header which ALCS applications use for the following purposes:

- Record identification
- Audit information (program stamp)
- Chain information (if record chaining is used)

Figure 67 shows the layout of the ALCS standard record header.

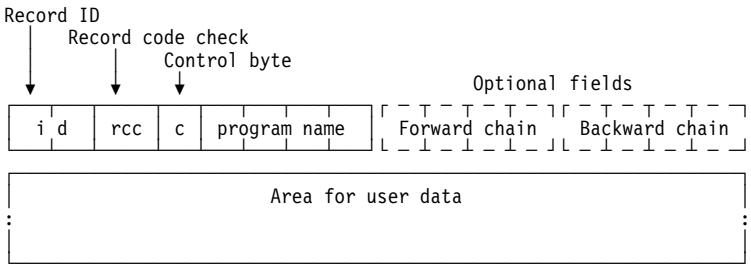


Figure 67. Record header

ALCS defines a standard 16-byte header format for DASD records. The header consists of a mandatory 8-byte part with optional extensions to 12 bytes (if standard forward chaining is used) or 16 bytes (if standard backward chaining is used).

In assembler programs use the `STDHD` DSECT macro to reference the fields in the standard record header. See the *ALCS Application Programming Guide* for details.

In C programs use the header file <c$stdhd.h> to reference the standard record header. See the *ALCS Application Programming Reference – C Language* for details.

Note: The $ character is the national currency symbol (X'B5').

The IBM program TPFDF uses an extended header for DASD records. TPFDF references the `STDHD` DSECT macro and <c$stdhd.h> C header file which describe

an extended record header containing 4-byte file addresses. TPFDF may also reference the ISTD8 DSECT macro and <c$std8.h> C header file which describe an extended record header containing 8-byte file address fields for compatibility with TPF.

The standard header contains the following fields:

Record ID
A 2-byte field that identifies the type of data in the record.

Record code check (RCC)
A 1-byte field that (optionally) distinguishes between records that contain the same type of data (the record ID is the same) but that are on different chains.

Control byte
A 1-byte field that (optionally) contains the characteristics of the record (for example, record size, type of record chaining, and so on). ALCS does not access the record control byte.

Program name
The 4-byte program name of the application program that last filed the record.

Chain fields
By convention, for records that use chaining (forward or backward), a 4-byte forward chain field and a 4-byte backward chain field complete the standard header.

4.1.6, “Overflow and chaining” on page 66 describes record chaining.

The system programmer can change the default attributes of a record type.

4.5.1 Record ID and RCC checking

When an application program reads a DASD record, it can specify the expected record ID (or record ID and RCC). ALCS checks that the actual ID (or ID and RCC) matches the expected ID (or ID and RCC). Application programs can suppress these checks.

When an application program writes a DASD record, it must specify the expected record ID (and optionally the RCC). ALCS checks that the actual ID (or ID and RCC) matches the expected ID (or ID and RCC).

4.6 Virtual file access

ALCS reduces the number of I/O operations by using **virtual file access** (VFA) to process all FIND and FILE macros issued by application programs. VFA holds the most recently used database records in buffers (**caches**) in processor storage. Figure 68 on page 83 gives an overview of VFA.

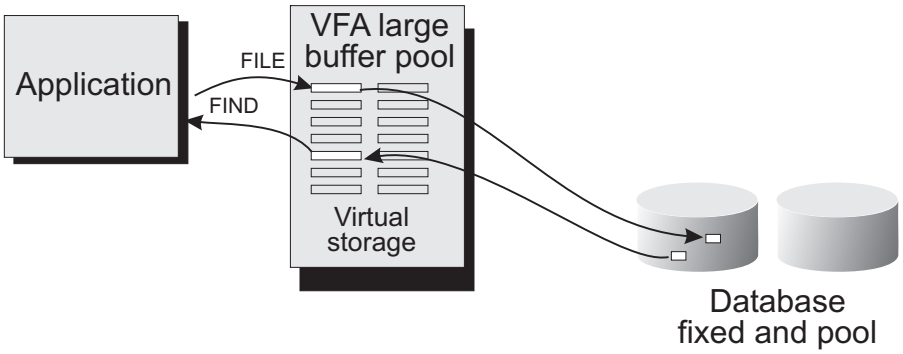


Figure 68. Virtual file access (VFA) overview

The more VFA buffers of each size that are specified at generation time, the greater the benefits in DASD and processor performance.

VFA combines the required level of data integrity and the minimum number of I/O operations by handling FILE macros according to the record VFA option specified at generation time. Different types of record have VFA options to determine whether changed records should be written to DASD at once, deferred for a short time, or held in the buffer.

For example, IPARS AAA records are referred to and perhaps updated several times for every transaction by the associated terminal. Holding these records in VFA buffers provides a faster response and greatly reduces the physical I/O for active terminals. The record is not written to DASD until the buffer is required by more active records. All updated records, regardless of VFA option, are written out when ALCS terminates.

The number of I/O operations to DASD depends on the size of the VFA buffer area. With large numbers of VFA buffers, VFA processes many FIND and FILE macros without doing any DASD I/O. This results in a reduced load on the DASD subsystem, and a lower processor utilization at a given message rate, because of the reduced number of I/O instructions and I/O interruptions. Use the `VFABUF` parameter of the ALCS `SCTGEN` macro to specify the number of VFA buffers during the ALCS generation.

VFA satisfies FILE macros by copying data records from the application's storage to a VFA buffer. It schedules the I/O operation to write the record to DASD from the VFA buffer, depending on the VFA options.

ALCS has the option of using hiperspace to provide additional buffering for DASD I/O.

If specified, ALCS will write the contents of VFA buffers that are about to be reused to hiperspace. This could further reduce DASD I/O if a subsequent FIND request for a record can be satisfied from its hiperspace copy.

ALCS Installation and Customization describes the `VFAOPT` parameter of the ALCS database generation macros to specify the characteristics of VFA. You can specify:

- The filing frequency
  - Immediate
  - Delayed (only when a buffer is required)

  - Time-initiated (flagged records are filed periodically)
- Permanence
  - Permanently resident (the buffer is never re-used)
  - Not permanently resident (VFA re-uses the buffer with the longest time since last referenced)
- New or update record
  - Update mode (VFA ensures that the VFA buffer contains the old record contents). This action is used for update logging; see E.2, “Update logging” on page 154.
  - Not update mode (VFA gets a new buffer and stores the record if necessary).
- Expanded Storage Option
  - Use Hiperspace backing to hold reused VFA buffers
  - No Hiperspace backing
- Online and offline access to the same general file
  - Forced read (always reads the record from DASD to use the latest copy of the record)
  - Not forced read (reads the record from DASD only if it is not in VFA buffers or Hiperspace)

4.7 Spill file on predecessor ALCS systems

Predecessor ALCS systems use **spill data sets** to reduce the need for frequent database reorganizations. These systems use **spill data sets** to extend (add records) to the database without moving any existing records.

For example, to add a new record type that is size L2, or to add more records to an existing record type that is size L2, ALCS uses one or more additional size L2 spill data sets.

ALCS Version 2 Release 4.1 supports any spill data sets (from predecessor ALCS systems) and allows you to add more spill data (using algorithm addressing) until you no longer need to fall back to a previous version of ALCS, or have migrated to type-2 long-term pool support.

You can then start to use the table-based addressing to extend the database.

4.8 The ALCS test database facility

New and changed application programs can contain errors capable of damaging the database. They must, therefore, be tested in such a way as to avoid damage to the existing database, while ensuring realistic testing.

One way of doing this is to copy all or part of the existing database to a test database; but the number of test databases for testing different programs could exceed the hardware available. This is especially true because ALCS allows any number of test systems to be in operation at one time, so that application programmers can test their different programs simultaneously and independently. The ALCS test database facility allows many application programmers (or groups of

programmers) to have their own test system without needing multiple copies of the entire test database.

Note: Do not use the test database facility if your configuration data sets have not been initialized. The note in “Creating configuration data sets” on page 90 explains this in more detail. Also, do not use the communication configuration data set (CDS2) on ALCS systems that use the test database facility. “Using CDS's with the test database facility” on page 91 explains this in more detail.

4.8.1 How the test database facility works

The ALCS test database facility gives a program read-only access to a database.

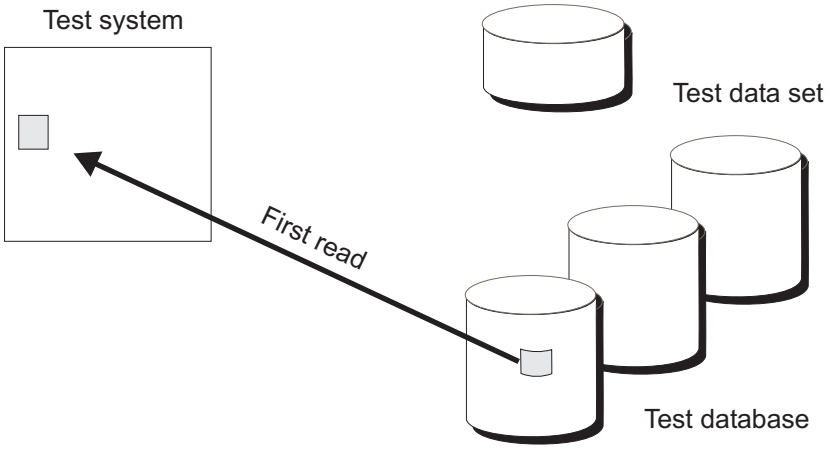


Figure 69. Test database: Read a record from the test database

When the program updates a record, ALCS writes it to a separate data set (the **test data set**) instead of updating the database from which it was read.

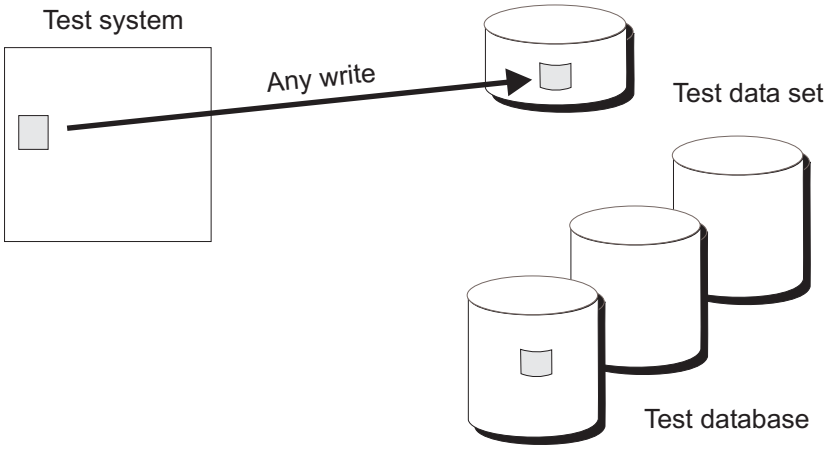


Figure 70. Test database: Write a record to the test data set

When the program next reads the record, ALCS gets it from the test data set. Thus the application seems to have normal read-write access to the database, which however remains unchanged.

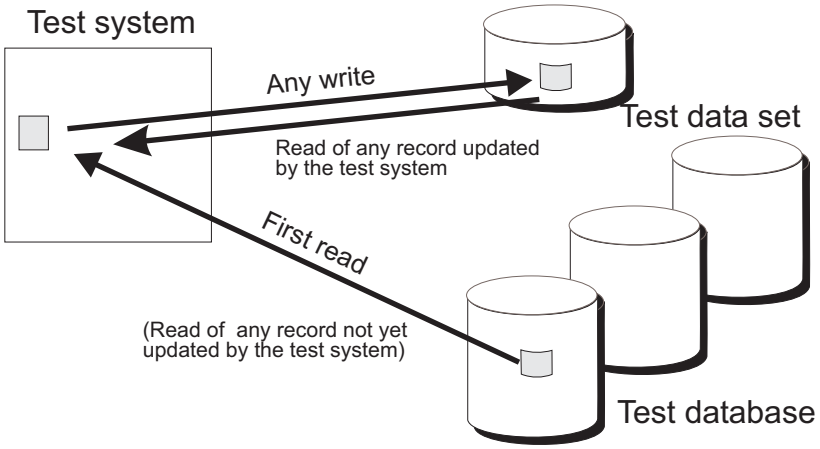


Figure 71. Test database: Read and write on the test data set

The ALCS test database facility makes it safe to test programs using the production database. However, most users prefer to use this facility with a stable test database that meets all the testing requirements of the programmers, with minimum impact on production work and without excessive demands on DASD space.

4.8.2 Benefits

Some of the benefits of the test database facility are:

Recovering from database damage

During a test, an application program can write bad data that makes the database unusable. If this happens, it is not necessary to restore the test database. Deleting all the records in the test data set has the same effect. An easy way of doing this is to delete the test data set and allocate a new (empty) data set.

If a series of tests takes several days or weeks, it is possible to back up the test data set at convenient times. If a particular test writes bad data, then it is easy to recover by restoring a backup of the test data set.

Sharing the test database

Several programmers may need a test database. The test database facility allows them all to share the same copy of the test database. If each programmer (or group of programmers) has a separate test data set, they can run tests simultaneously without interfering with one another.

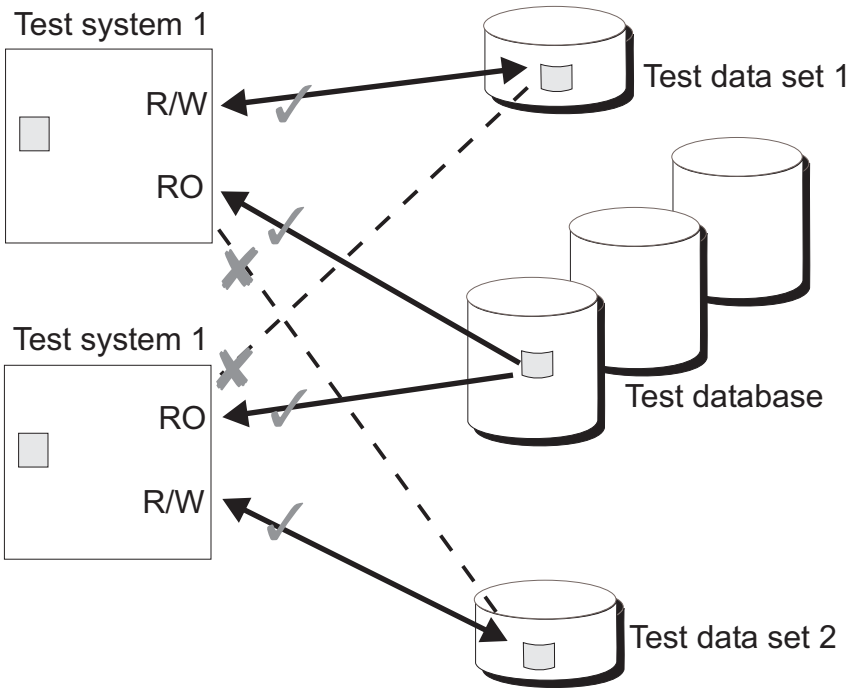


Figure 72. Test database: Shared testdata base, separate test data sets

Also, programmers who are running a series of tests do not put other tests at risk by changing or damaging their data.

A programmer can create a new test data set by copying an existing test data set. In this way, programmers can run tests against a database that already includes updates from previous testing. For example, one programmer might initialize a number of records, then a second programmer might copy the test data set. Both programmers can then run tests that use these records.

You can also use the `ZRELO` command to move records from a sequential file to your test database. This is described in *ALCS Operation and Maintenance*.

4.8.3 Test pool files

If your test database is a copy of all or part of your production database then you may wish to update it from time to time by making a new copy from the production database. But if you do this, you may find that some pool file addresses (both long-term and short-term) that were available when you made your previous copy from the production database have since been dispensed on both the production database and on one or more test data sets. This can cause some strange effects – for example:

Suppose that at the time you made a copy of the production database pool-file record L1LTPOOL(200) was available. A program running on the production system dispenses the record for use as an overflow for the fixed-file record #FIXEDA(20). It saves the address of L1LTPOOL(200) in #FIXEDA(20). Meanwhile, a program running on a test system dispenses the record for use as an overflow for the fixed-file record #FIXEDB(30). It saves the address of L1LTPOOL(200) in #FIXEDB(30). Figure 73 shows the two versions of the pool record.

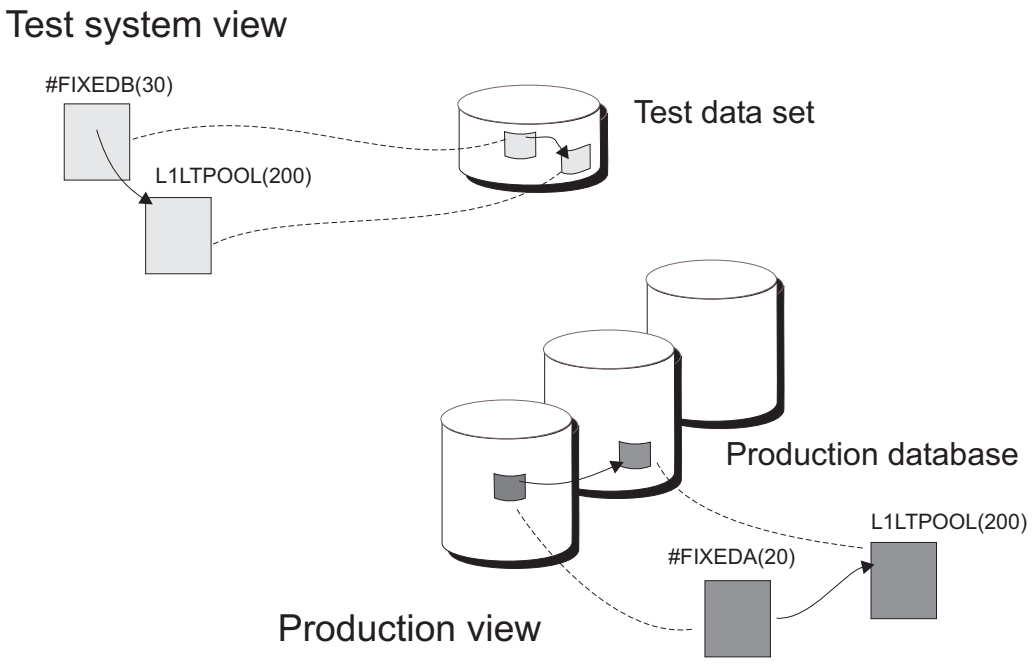


Figure 73. Test database: Copy record to test data set

If you now make a new test database from the production system, there are two "versions" of record L1LTPOOL(200). The one on the test data set is an overflow for the fixed-file record #FIXEDA(20), the one on the test database is an overflow for the fixed-file record #FIXEDB(30).

If a program running on the test system needs to access the overflow record for #FIXEDB(30) then it will read L1LTPOOL(200) and find the data that it expects. But if the program needs to access the overflow for #FIXEDA(20) – assuming that #FIXEDA(20) has not been updated on the test system – then it will read L1LTPOOL(200) and find the overflow for #FIXEDB(30). Figure 74 on page 89 shows the two versions of the pool record.

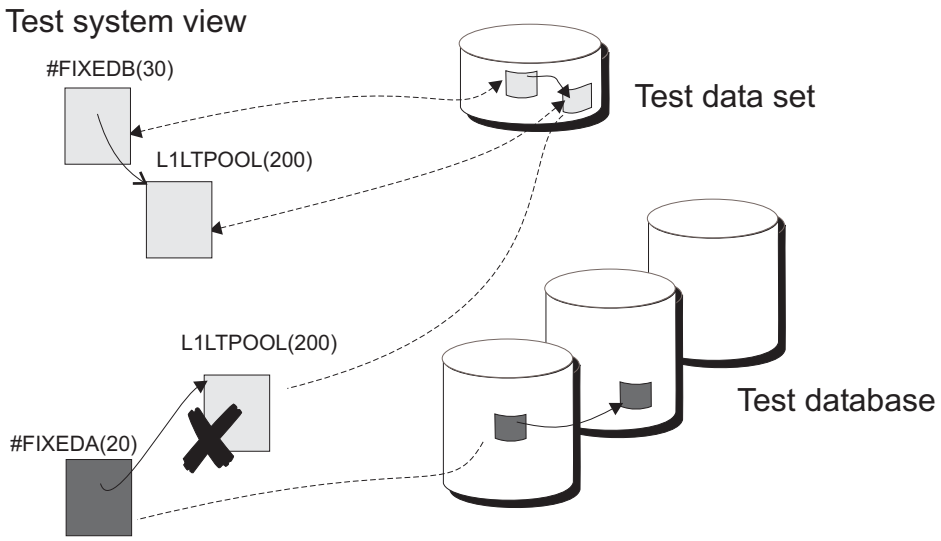


Figure 74. Test database: Incorrect overflow file address

To avoid these strange effects, you can partition pools to reserve pool records for test database use.

For example, you can reserve 1 percent of pool records for test database use, and leave 99 percent for production use.

This avoids the problem of incorrect pool addresses between the test database and the production database, because it is impossible to dispense the same record on both the test system and the production system.

See *ALCS Installation and Customization* for more information about the PARTITION parameter of the `DBGEN` macro.

4.9 The ALCS configuration data sets

ALCS provides three configuration data sets. They contain system configuration information that is used when the ALCS system is restarted. The three configuration data sets are:

- the database configuration data set, CDS0
- the program configuration data set, CDS1
- the communication configuration data set, CDS2

The database CDS is required by all ALCS systems, but the other two CDS's are optional.

ALCS system configuration information is also held in load modules whose names are in the PARM list on the EXEC statement that starts the ALCS job. During ALCS restart, configuration data may be obtained from the CDS's and from the configuration load modules referenced by the PARM list.

Creating configuration data sets

The ALCS database configuration contains the data set names of the CDS's. The CDS's are created using the z/OS IDCAMS utility. Each CDS must contain 4096 records, and the records in each CDS must have a CI size of 4096 and a record size of 4088. When the ALCS system is restarted, it obtains the names of all the CDS's from the database configuration referenced by the PARM list. If the CDS's are newly created, ALCS automatically preformats them. When the preformat is complete, ALCS copies the information held in the configuration load modules (referenced by the PARM list) to the CDS's. For example, the information held in the database configuration load module is copied to the database configuration data set (CDS0). On subsequent ALCS restarts, ALCS will use configuration information in the CDS's.

Note: ALCS cannot preformat your CDS's if you are using the test database facility. This is because the CDS's are opened as read only data sets. In this situation, you need to run your ALCS system once without the test database facility. Once the CDS's have been preformatted you may use them with the test database facility.

Updating the configuration data sets

The ALCS system provides operator commands that update the contents of the CDS's.

- Updating CDS0, the database configuration data set

  This CDS is updated when you load a new database configuration table using the `ZDASD LOAD` command. When you wish to update the ALCS database configuration, you must create a new database configuration table and load it onto CDS0.

- Updating CDS1, the program configuration data set

  This CDS is updated when application programs (for system wide use) or installation-wide monitor exits are loaded using the `ZPCTL LOAD` command. When you wish to consolidate the application programs and monitor exits that have been loaded online, you can create a new program configuration table and load it onto CDS1.

- Updating CDS2, the communication configuration data set

  This CDS is updated when communication configuration load modules are loaded using the `ZACOM LOAD` command. These communication configuration load modules enable you to reconfigure your ALCS communication network. When you wish to consolidate the ALCS communication configuration load modules, you can create a new communication configuration load list and load it onto CDS2.

All updates to the CDS's are performed by operator commands that include a LOAD parameter (`ZDASD LOAD`, `ZPCTL LOAD` and `ZACOM LOAD`). After a successful *load*, if the update is to be retained over a system restart, it must be *confirmed*. This is achieved by entering the relevant command with the CONFIRM parameter. After the *confirm*, the update that has been loaded onto the CDS is used by ALCS during the next system restart. After a successful restart of the ALCS system, the updates on the CDS that are in the *confirmed* status should be *committed*. This is achieved by entering the appropriate command with the COMMIT parameter. While an update is in the *load* or *confirmed* status, it can be backed out from the CDS by entering the relevant command with the BACKOUT parameter.

The ALCS CDS's therefore enable online updates of the database and communication configuration (plus online loads of programs and exits) to be automatically reloaded after a system restart.

Using CDS's with the test database facility

The database CDS is required by all ALCS systems, therefore it must be available to ALCS systems that use the test database facility.

The program CDS is optional, therefore you must decide if it is required on your ALCS systems that use the test database facility.

Although the communications CDS is optional, it should not be used on ALCS systems that use the test database facility. Each ALCS system requires a different communication configuration. For example, each ALCS system requires its own VTAM ACB name, its own prime and RO CRAS terminals, etc. and should not share its communication configuration with other ALCS systems.

Chapter 5. Sequential file management

ALCS uses the MVS sequential access method (SAM) to create and access physical sequential data sets, which are called **sequential files**. The online monitor creates some physical sequential data sets directly. It also supports monitor-request macros that allow application programs to create and access physical sequential data sets.

ALCS supports six types of sequential file as follows:

- Four types of **system sequential files** (sequential files that the ALCS online monitor creates). Application programs cannot write records to system sequential files.
- Two types of **application sequential files** (sequential files that ALCS application programs create or access).

Figure 75 shows the ALCS sequential data sets, this includes:

- The six types of sequential file
- The ALCS direct-access data sets (these are only included to differentiate between the sequential data sets and the direct-access data sets)

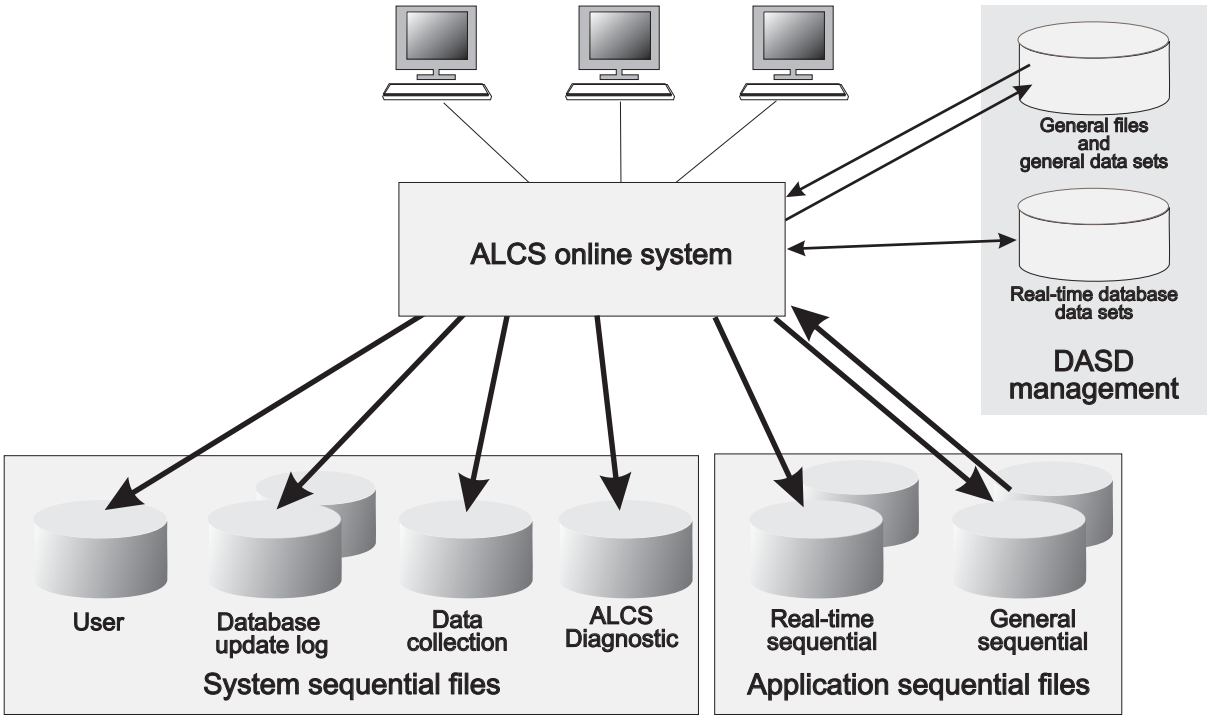


Figure 75. ALCS sequential files: Overview

Figure 76 gives an overview of how each type is used.

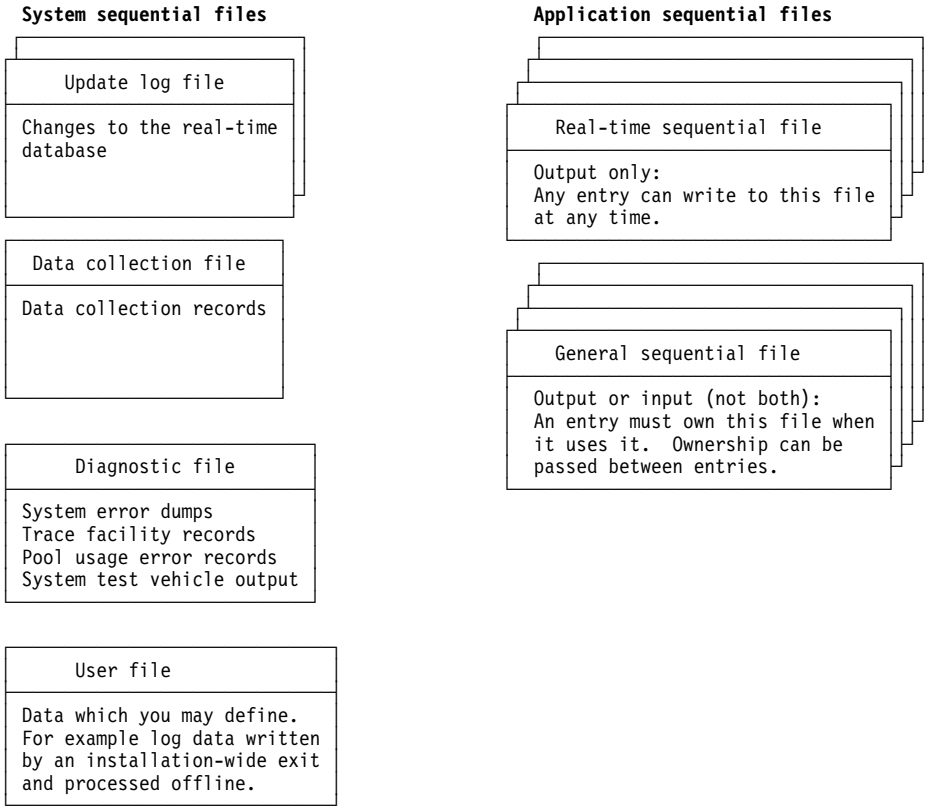


Figure 76. Sequential files: Types and contents

For sequential files, ALCS supports the device types that MVS SAM supports. However some device types can be unsuitable for some sequential files. For example, to dump records from the ALCS real-time database, DASD or magnetic tape can be suitable; punched cards or paper listings are probably unsuitable.

5.1 System sequential files

The ALCS online monitor creates the following system sequential files:

- ALCS diagnostic file
- ALCS update-log file or files
- ALCS data-collection file
- ALCS user file

5.1.1 ALCS diagnostic file

Every sequential-file generation must define an ALCS diagnostic file. ALCS writes diagnostic information to the ALCS diagnostic file. This data includes:

- System error dumps
- ALCS trace facility records
- Pool usage error records
- ALCS system test vehicle output
- Data-collection output (if you do not define a separate ALCS data-collection file)
- User output (if you do not define a separate ALCS user file)

The ALCS diagnostic file processor formats and prints the diagnostic information contained by the ALCS diagnostic file. The ALCS statistical report generator formats and prints the performance statistics contained in the ALCS diagnostic file.

5.1.2 ALCS update-log file(s)

Every sequential file generation must define an ALCS update-log file. The ALCS monitor logs changes to the real-time database in the ALCS update-log file. When ALCS writes a record to the real-time database, it can also write a copy of the record to the ALCS update-log file. ALCS uses the ALCS update-log file to restore the real-time database after an equipment or program error destroys it.

ALCS supports two types of logging:

Forward ALCS logs the record contents after they are updated
Backward ALCS logs the record contents before they are updated

You can specify the following combinations of logging in the ALCS generation process:

- A forward log, or
- A backward log, or
- Both a forward and a backward log
- A single merged file for both a forward and a backward log

5.1.3 ALCS data-collection file

When data collection is active, ALCS writes data-collection records to a system-sequential file. This file can be either:

- The ALCS **data-collection** file. To do this, you simply define an ALCS data-collection file.
- The ALCS **diagnostic** file. If you do not define an ALCS data-collection file then ALCS writes the records to the ALCS diagnostic file (the data-collection records are intermixed with other information).

Subsequently, you can process this sequential file to produce performance reports using:

- The ALCS statistics report generator (SRG)
- The IBM Service Level Reporter (SLR)
- Any similar utility

5.1.4 ALCS user file

ALCS itself does not write to this system sequential file. It is provided for you to write data from an installation-wide monitor exit. To do this you must use the callable service `UWSEQ` (see *ALCS Installation and Customization*).

To process the data written to the user sequential file you must write your own offline program.

If you do not define an ALCS user sequential file then ALCS writes the records to the ALCS diagnostic file (the user records are intermixed with other information).

5.2 Application sequential files

Application programs can use sequential files.

ALCS supports the following types of **application sequential** file:

- Real-time sequential files
- General sequential files.

Real-time sequential files

Real-time sequential files are output data sets.

Application programs do not open and close real-time sequential files. Instead, ALCS initialization allocates and opens them. They remain open all the time that ALCS is executing.

Real-time sequential files are shared resources. An entry does not own a real-time sequential file. Instead, any entry can write a record to it at any time.

General sequential files

ALCS supports both input and output general sequential files. A general sequential file can be either an input or an output data set; it cannot be both. However, "Multiple sequential files on one MVS data set" on page 97 explains how to use a data set for both input and output.

Application programs must open and close general sequential files. ALCS supports monitor-request macros and C language functions to do this. When an application program issues `TOPNC` or `topnc`, the ALCS online monitor allocates and opens the data set. When an application program issues `TCLSC` or `tclsc`, the ALCS online monitor closes and deallocates the data set.

General sequential files are not shared resources. An entry must own a general sequential file before the entry can use it. A general sequential file is **assigned** when an entry owns it. In additon to `TOPNC` (`topnc`) and `TCLSC` (`tclsc`), ALCS supports monitor-request macros and C language functions that allow application programs to pass ownership from one entry to another.

5.3 Symbolic names for sequential files

Every ALCS sequential file has a unique 3-character symbolic name. Application programs use this symbolic name to identify the sequential file. For example, the `TOPNC` and `TCLSC` monitor-request macros include the symbolic name as a parameter. ALCS commands use this symbolic name to identify the sequential file. For example, the `ZDSEQ` command has a parameter that limits the display to a single sequential file; the parameter is the 3-character symbolic name.

The ALCS sequential file generation must include one `SEQGEN` macro for every valid 3-character symbolic name. The `SEQGEN` macro associates the 3-character symbolic name with an MVS data set.

Multiple names for the same file

ALCS allows more than one 3-character symbolic name to refer to the same sequential file (the 3-character names are synonyms). This can be useful with real-time sequential files. Application programs can use different symbolic names for different types of data. The sequential file generation can specify that all the names refer to the same real-time sequential file. Later, a new sequential file generation can specify that the names refer to different sequential files. In this way, a system can generate a small number of sequential files at first. Later, if the volume of output increases, the system can write the data to a larger number of sequential files. Figure 77 gives an overview of this particular mapping.

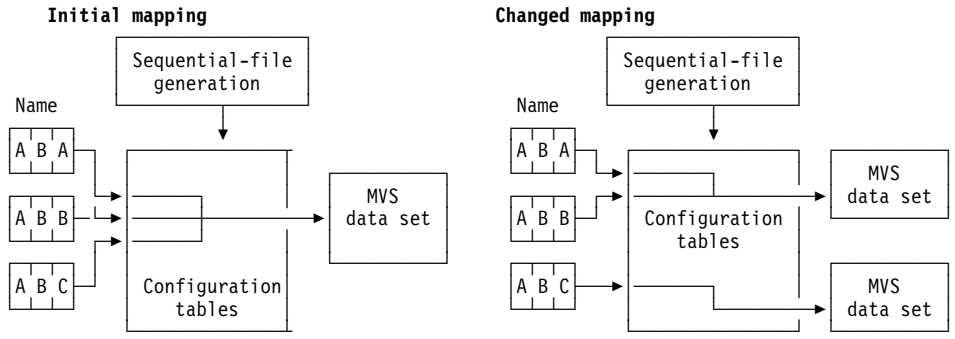


Figure 77. Sequential files: Changing the mapping of symbolic names

Multiple sequential files on one MVS data set

The ALCS sequential file generation can define more than one sequential file with the same data set name. In this case, the different 3-character names are not synonyms. Although they refer to different ALCS sequential files they refer to the same MVS data set. This can be useful if one ALCS application creates a sequential file, and another ALCS application reads the same data set. The first application must use an output sequential file; the second application must use an input sequential file. To do this, the application programs must use different symbolic names. Figure 78 gives an example of this type of mapping.

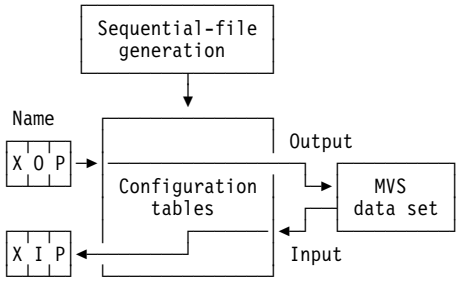


Figure 78. Sequential files: Mapping input and output to a data set

The symbolic names refer to different ALCS sequential files; the first is output, disposition NEW, and so on; the second is input, disposition OLD, and so on. But the sequential files have the same data set name – they are the same data set.

5.4 Cataloged sequential file data sets

When a data set is cataloged, the catalog entry specifies the volume serial number. Before an application program can use an input data set that is not cataloged, you must tell ALCS the volume serial number: use either the SEQGEN macro or the ZASEQ command (see *ALCS Operation and Maintenance*) to do this.

5.5 Sequential file data set switch

ALCS initialization opens system sequential files and real-time sequential files. That is, it allocates and opens MVS data sets. ALCS continues to write data to the system sequential files all the time that ALCS is executing. Application programs continue to write to real-time sequential files all the time that ALCS is executing. But the ALCS diagnostic file processor cannot process (for example) an ALCS diagnostic file data set until ALCS closes and deallocates the data set.

To satisfy these requirements, an ALCS system sequential file (or an ALCS real-time sequential file) is not a single MVS data set. Instead, it is a sequence of MVS data sets. The data set names are the same except for the last qualifier; the last qualifier is a sequence number. For example, a sequence of ALCS diagnostic file data sets might be:

```
ALCS.DIAG.A0000000
ALCS.DIAG.A0000001
ALCS.DIAG.A0000002
```

The ALCS sequential file generation specifies how much data (the number of blocks) ALCS writes to one data set. ALCS then closes and deallocates the data set and starts to use the next data set in the sequence. This process is called **sequential file switch**. After a sequential file switch, other programs can use the previous data set in the sequence.

ALCS automatically performs a sequential file switch when the data set is full (that is, after it writes the specified number of blocks). The ALCS command, ZSSEQ, also performs a sequential file switch. Use ZSSEQ, for example, to switch to a new data set so that the ALCS diagnostic file processor can print a system error dump from the current ALCS diagnostic file data set; otherwise you would have to wait until ALCS fills up the data set.

To reduce the time for a sequential file switch, ALCS allocates and opens in advance the next data set in the sequence. As soon as ALCS starts to write data to one data set, it allocates and opens the next data set in the sequence. In this way, ALCS does not need to wait for volume mounts, and so on, when it performs a sequential file switch.

If sequential file data sets are on magnetic tape, you can set up the data sets as single volume data sets so that ALCS switches data sets at the end of the volume. To do this, specify the SPACE parameter on the ALCS generation SEQGEN macro to reflect the capacity of a reel of tape or tape cartridge.

Chapter 6. Entry management

An entry is an ALCS unit of work. ALCS can create and dispatch entries. This section describes how ALCS creates and dispatches entries. See also 1.7, “Multiprogramming and multiprocessing” on page 34.

6.1 How ALCS creates entries

To create entries, ALCS uses the following:

- Input messages
- Create-type monitor-request macros (CREMC, CREDC, CRETC, CREXC)
- Monitor-created entries
- The time available supervisor (TAS)

Input messages: ALCS creates a new entry for every input message; that is, whenever ALCS receives an input message. To create this new entry the online monitor does the following (not necessarily in this order):

- Initializes storage to contain an ECB and an attached storage block. The attached storage block contains the input message on ECB level 0.
- Updates internal counters and control fields to indicate that there is one more entry.
- Indicates that the entry is an input message entry and saves origin information in the ECB and the ECB descriptor. This origin information includes the **communication resource identifier** (CRI) of the originating communication resource.
- Adds the entry to the input list. The input list is a queue of entries that are new input messages. It is one of the ALCS entry dispatcher work lists. Figure 79 on page 102 shows the structure of an ALCS entry-dispatcher work list.

Create-type services: ALCS provides services that allow application programs to create new entries. When an entry requests one of these services, ALCS creates a new entry as follows (not necessarily in this order):

- Initializes storage to contain the new ECB. If the create-type macro passes storage block contents, ALCS initializes attached storage blocks as required.
- Updates internal counters and control fields to indicate that there is one more entry.
- Indicates that the entry is a created entry and copies origin information from the creating entry to the ECB and the ECB descriptor.
- Adds the entry to one of the ALCS entry dispatcher work lists described in 6.2, “How ALCS dispatches entries” on page 101.

In the same way, ALCS creates new entries for other services that can create new entries. These include:

- System error macros (SYSRA and SERRC). These services can create a new entry that sends a system error dump message to RO CRAS.
- Send-type monitor-request macros. Some send services create a new entry that processes the output message. SENDC T (that is, SENDC with the T parameter) is an example.

Monitor-created entries: ALCS sometimes creates new entries that do not arise from input messages or create-type services. For example, the monitor timer routines create a new entry every minute. The new entry is for application-dependent timer functions.

These new entries are called **monitor-created entries**. ALCS creates them as follows (not necessarily in this order):

- Initializes storage to contain the new ECB and attaches storage blocks as required.
- Updates internal counters and control fields to indicate that there is one more entry.
- Indicates that the entry is a monitor-created entry and clears origin information in the ECB and the ECB descriptor.
- Adds the entry to one of the ALCS entry dispatcher work lists described in 6.2, “How ALCS dispatches entries” on page 101.

Time available supervisor: ALCS supports a facility called **time available supervisor (TAS)**. TAS allows an entry to create other entries with very low priority. To use TAS, entries issue the TASTC and TASBC monitor services TASTC starts TAS (and makes TAS active) and TASBC stops it (makes it inactive).

Unlike the create-type services, TASTC does not create a new entry and add it to an ALCS entry dispatcher work list. TASTC sets a switch to indicate that there is low priority work waiting. The entry dispatcher checks the switch when there is no other work that ALCS can dispatch. If the switch is set, ALCS creates a new entry and dispatches it immediately. See 6.2, “How ALCS dispatches entries” on page 101 for more information. ALCS does this as follows (not necessarily in this order):

- Initializes storage to contain an ECB.
- Updates internal counters and control fields to indicate that there is one more entry.
- Indicates that the entry is a TAS-created entry and clears origin information in the ECB and the ECB descriptor.
- Dispatches the new entry by transferring control to the application program TIA1.

ALCS continues to create and dispatch new entries in this way while TAS is active (that is, until an entry requests TASBC monitor service). In this way, an entry can create a number of low priority entries.

6.2 How ALCS dispatches entries

This section contains Diagnosis, Modification, and Tuning Information. Do not use this information as a programming interface.

This section summarizes the normal way in which ALCS dispatches entries. Do not develop application programs that depend on the way that ALCS dispatches entries. Programs that depend on the ALCS dispatching method can fail unpredictably.

For example, do not develop application programs that depend on the priority order of the ALCS entry dispatcher work lists. In some situations the ALCS entry dispatcher checks work lists in a different order.

To dispatch an entry, the ALCS entry dispatcher (often called the **CPU loop**) checks lists of entries, called **ALCS entry dispatcher work lists**. Each list contains only entries that are dispatchable. For example, if an entry waits for an I/O operation to complete, ALCS does not add the entry to an ALCS entry dispatcher work list until the I/O completes.

All the entry dispatcher tasks check the same set of work lists.

The ALCS entry dispatcher checks its work lists in sequence. In this way, each list corresponds to a priority. The ALCS entry dispatcher work lists are, in order of priority:

IOCB list ALCS uses the IOCB list internally to dispatch system functions.

Ready list ALCS uses this list mainly for entries that can resume processing after I/O completion. For example, if an entry requests a `FINWC` monitor service to read a DASD record, it loses control until I/O completes. When I/O completes, ALCS adds the entry to the ready list.

Delay list ALCS uses this list mainly for entries that request `DEFRC` or `DLAYC` monitor services. Entries request monitor services to avoid monopolizing resources. Refer to 6.5, “Delay and defer processing” on page 104.

Input list ALCS uses this list mainly for new entries. For example, when ALCS receives an input message, it creates a new entry and adds the new entry to the input list.

Defer list ALCS uses this list mainly for entries that request `DEFRC` or `DLAYC` monitor services. Entries request monitor services to avoid monopolizing resources. Refer to 6.5, “Delay and defer processing” on page 104.

Deferred IOCB list ALCS uses the deferred IOCB list internally to dispatch system functions.

If the ALCS entry dispatcher can dispatch an entry from the first list, it does so, and does not check subsequent lists. If it cannot dispatch an entry from the first list, it normally checks the second list, and so on. However, if the system load is heavy the ALCS entry dispatcher does not service the delay, input, or defer lists. Regardless of load, the ALCS entry dispatcher does not service the delay and defer lists in certain situations described in 6.5, “Delay and defer processing” on page 104.

Figure 79 shows the format of an ALCS entry dispatcher work list.

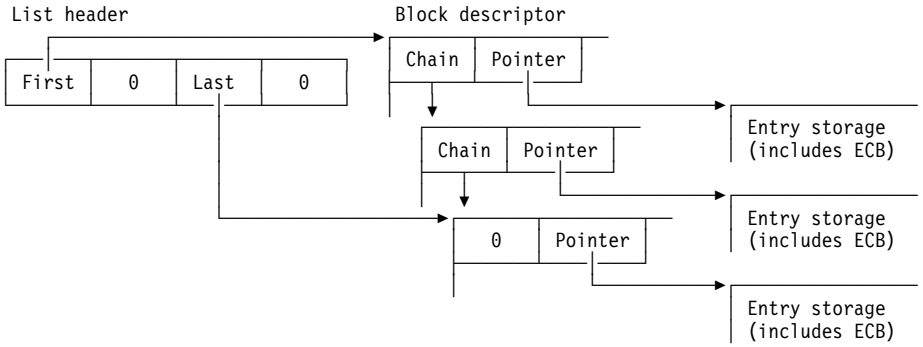


Figure 79. Entry dispatcher work list (schematic)

An ALCS entry dispatcher work list is a first-in-first-out (FIFO) queue. That is, ALCS adds an entry to the bottom of an ALCS entry dispatcher work list; the ALCS entry dispatcher starts checking at the top of the list.

However on a multiprocessor, the ALCS entry dispatcher cannot always dispatch the first entry on a list. For example, the first entry might need exclusive control of a global area field that is already controlled by another entry in shared or exclusive mode. If the ALCS entry dispatcher cannot dispatch the first entry on a list, then it checks the next entry, and so on.

When ALCS adds an entry to an ALCS entry dispatcher work list, it saves the address of the **post-interrupt routine** in the ECB descriptor. To dispatch an entry, the ALCS entry dispatcher removes the entry from the work list and branches to the post-interrupt routine. Generally, the post-interrupt routine does one of the following:

- Add the entry to another ALCS entry dispatcher work list and branch to the ALCS entry dispatcher.
- Exit the entry and branch to the ALCS entry dispatcher.
- Transfer control to an application program.

In the last case, eventually the application program must request a monitor service that loses control. The macro service routine branches to the ALCS entry dispatcher.

If the ALCS entry dispatcher cannot dispatch any entry, it enters the MVS task wait state. When an event makes an entry dispatchable, ALCS posts the ALCS entry dispatcher tasks. The ALCS entry dispatcher then checks the work lists again.

6.2.1 Entry processing limits

ALCS can detect and cancel (exit) entries that monopolize or misuse resources. For example, if an entry dispenses too many pool file record addresses, then ALCS cancels the entry with a system error dump.

To do this, ALCS uses entry processing limits specified by the generation parameters ENT*xxxx* of the `SCTGEN` macro.

ALCS Installation and Customization describes the `SCTGEN` macro.

ALCS can also detect application programs that loop without losing control, see 6.3.1, “Application loop timeout”.

6.3 How entries lose control

Many multiprogramming systems invoke the dispatcher at every interrupt. In these systems, a task can lose control at any time (unless the task can mask the processor against interrupts). This is because an interrupt can make a higher priority task dispatchable. If it does, the multiprocessing system stops processing the active task and dispatches the higher priority task instead. This is called **preemptive dispatching**.

ALCS does not use preemptive dispatching. Instead, ALCS invokes its entry dispatcher only when an application program requests a monitor service. If an interrupt makes an entry dispatchable, ALCS adds it to an ALCS entry dispatcher work list; the active entry continues to execute. This is called **non-preemptive dispatching**. Whether or not ALCS invokes the entry dispatcher depends on which service the entry requests. Each service does one of the following:

- Always loses control; that is, invokes the entry dispatcher (for example, the `DEFRC` monitor service).
- Never loses control (for example, the `TIMEC` monitor service).
- Sometimes loses control (for example, the `ENTRC` monitor service loses control only when the global serialization requirements for the entering program and for the entered program differ).

ALCS Application Programming Reference – Assembler and *ALCS Application Programming Reference – C Language* contain descriptions of all assembler and C language services respectively. These descriptions indicate whether the service causes the entry to lose control (that is, whether or not the routine invokes the entry dispatcher).

6.3.1 Application loop timeout

An ALCS entry does not normally lose control until it requests a monitor service. However, an entry can monopolize a processor when it goes into a loop and does not request any monitor service (or requests only monitor services that do not lose control).

To prevent this, ALCS imposes a maximum time for an entry to execute before it must lose control. If an entry does not lose control within this time limit, then ALCS cancels (exits) the entry with a system error dump. This procedure is called **application loop timeout**.

Do not develop application programs that depend on the exact size of the application loop timeout. Instead, ensure that entries lose control at least after every 10 000 instructions.

6.4 Input/output counter and wait service

ALCS provides monitor services that request input/output (I/O) operations. For example, `FILEC` requests a write to a DASD data set, `TPRDC` requests a read from a sequential file (physical sequential data set), and so on. Generally the I/O can proceed independently; the entry can continue processing while the I/O takes place. An entry that requests an input operation must ensure that the data input completes successfully before it uses the data. To do this, the entry uses the ALCS wait service. The wait service:

- Suspends the entry until the I/O completes
- Tests for I/O errors

If an entry requests input when a buffer already contains the data, the wait service does not suspend the entry. Even so, the entry must always request the wait service. The entry cannot know that a buffer already contains the data.

The ALCS `WAITC` monitor service and the `waitc` C language function request the wait function. Some other monitor services and C language functions request both I/O and the wait service. For example, `FINWC` requests a DASD read followed by the wait service.

ALCS does not support the wait service for all monitor services that request I/O. For example, `FILEC` requests a DASD write. The wait service does not wait for I/O that `FILEC` requests. Also, the wait service does not test for `FILEC` I/O errors. So there are two types of monitor service or C language function that request I/O:

- The I/O request requires the wait service. The entry *must* request the wait service to check that the I/O completes successfully.
- ALCS does not support the wait service for the I/O request (to reduce response time). The entry *cannot* check that the I/O completes successfully.

The description of each service in *ALCS Application Programming Reference – Assembler* and *ALCS Application Programming Reference – C Language* indicates whether or not it requires the wait function.

6.5 Delay and defer processing

Delay and defer list service: 6.2, “How ALCS dispatches entries” on page 101 describes how the ALCS entry dispatcher services the ALCS entry dispatcher work lists in the following order of priority:

1. Ready list
2. Delay list
3. Input list
4. Defer list

It only checks the input list when it cannot dispatch any entry on the ready list, and so on. Also, the ALCS entry dispatcher does not service the input, delay, and defer lists if ALCS is too busy.

Sometimes the ALCS entry dispatcher does not service the delay and defer lists even when there are no other dispatchable entries and ALCS is not too busy. This allows other MVS tasks to have priority over ALCS entries that are on the delay and defer lists. It also allows time for write operations to complete before the ALCS entry dispatcher redispatches an entry that requests a DEFRC monitor service.

In particular, once the ALCS entry dispatcher has serviced the defer list, it does not do so again until 0.05 seconds later.

Attention

You should be aware of the dangers of defer/delay loops waiting for some external event that never happens. You can overcome these dangers either by using EVNTC, EVNWC, and POSTC, or by incorporating a check to limit the amount of time spent in the defer/delay loop.

Entry write limits: Some ALCS monitor services request output operations that proceed independently of the entry. See 6.4, “Input/output counter and wait service” on page 104. For example, FILEC requests a DASD output operation (write) but the entry does not wait for the output to complete. However an entry can use all the available DASD I/O buffers if it goes into a loop that requests a FILEC monitor service.

To prevent this, ALCS requires entries to request DEFRC monitor service (or DLAYC). DEFRC and DLAYC suspend processing of an entry to allow other entries to proceed. The ENTWRT= parameter of the SCTGEN macro specifes two limits (*ew1* and *ew2*) to regulate this.

The *ew1* value is the maximum number of writes that an entry can request before it must defer. When an entry requests more than *ew1* writes without issuing a DLAYC or DEFRC macro, ALCS forces the entry to defer. That is, for one write in every *ew1* writes, ALCS adds the entry to the defer list during the write monitor service. This gives a time delay (on average 0.025 seconds) that allows the writes to complete.

When the entry continues to request writes, ALCS forces it to defer again after the next *ew1* writes, and so on.

An entry can prevent these forced defers by issuing DLAYC or DEFRC more often than every *ew1* writes.

When an entry requests DLAYC or DEFRC monitor service, the other limit of the pair, *ew2*, decides whether ALCS puts the entry on the delay list or the defer list. If there have been *ew2* (or more) writes since the entry lost control, ALCS adds the entry to the defer list during the DLAYC or DEFRC monitor service. This gives a time delay (approximately 0.025 seconds) to allow any writes to complete.

When an entry requests a write and requests a DEFRC monitor service every time round a loop, for example:

```
LOOP   EQU   *
⋮
       FILEC D1
       DEFRC ,
⋮
       B     LOOP
```

then ALCS puts the entry on the delay list each time round the loop, except that every *ew2* times round the loop ALCS puts the entry on the defer list.

TPF compatibility

TPF implements DLAYC and DEFRC slightly differently from ALCS. The differences are not normally significant to application programmers, but do not request DEFRC monitor service with records held if you wish to maintain compatibility with TPF.

6.6 Communication between entries

Sometimes entries need to communicate with other entries. That is, they must pass data to or share data with other entries. But an ALCS entry must not access ECB fields or storage blocks that belong to another entry.

6.6.1 How entries pass data

ALCS provides monitor services that allow an entry (the creating entry) to create a new entry (the created entry). These are the create-type services. Create-type services have parameters that allow the creating entry to pass data to the created entry. The creating entry can specify:

- The address and length of a parameter area. ALCS copies the contents of the parameter area into the ECB work area of the created entry.

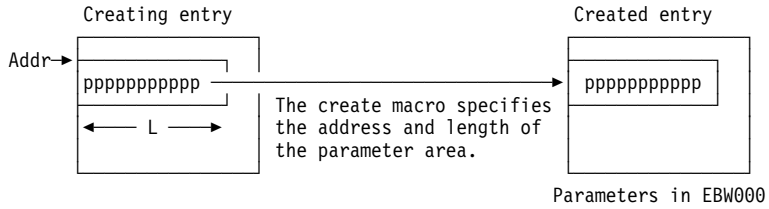


Figure 80. Passing data between entries: parameter areas

- One or more storage blocks attached to the ECB of the creating entry. ALCS copies the contents of the storage blocks into the new storage blocks attached to the ECB of the created entry.

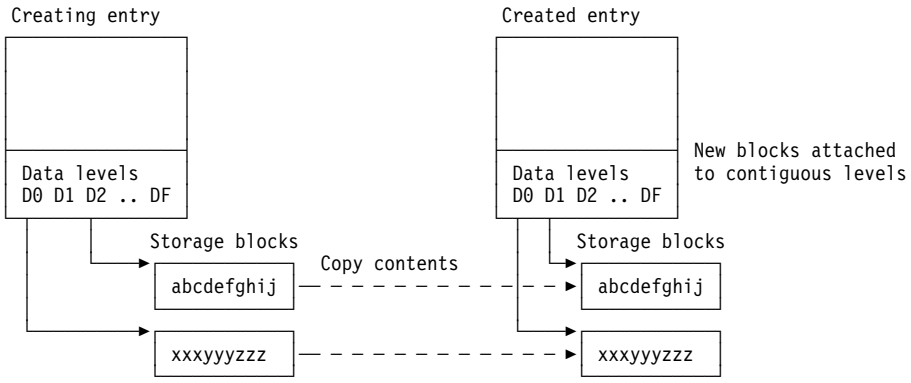


Figure 81. Passing data between entries: storage blocks

ALCS Application Programming Reference – Assembler and *ALCS Application Programming Reference – C Language* explain how to use these services and their parameters.

6.6.2 How entries share data

Creating entries can share data with created entries.

The main types of data that ALCS entries can share are:

- Application global area
- DASD records

ALCS supports facilities designed to help entries to share data.

ALCS supports an area of storage that entries can share. This is called the **application global area**. This area is not protected; any entry can store information in the global area.

It can be complex to use the application global area in a multiprocessor environment for anything except constants. It is much simpler to store variable data in DASD records (appropriate use of virtual file access (VFA) makes physical I/O unnecessary).

However, if you cannot avoid using the application global area, *ALCS Installation and Customization* describes how to use the application global area.

E.3, “Record hold facility” on page 155 describes facilities that help entries to share DASD records.

6.7 Transactions that create multiple entries

Many transactions require only one entry each for their processing – that is, they do not need to request create-type services. But you may find it more efficient to implement some complex transactions by creating a number of entries that can process in parallel.

For these transactions, the original entry (called the parent entry) creates a number of child entries, which can run parallel with each other and with the parent entry.

The child entries can in turn create more grandchild entries, and so on.

Application programs that create child and grandchild entries in this way can efficiently exploit ALCS's multiprogramming and multiprocessing capabilities – but these types of program can cause serious problems unless they are carefully designed. When designing new programs or installing purchased programs of this type, you should observe the following points:

Avoiding too many create-type services

In most ALCS configurations, ALCS services create-type services in preference to new input transactions. This is to ensure that in-flight transactions complete even when there is a high input message rate. Otherwise an in-flight transaction that requests a create-type monitor service might have to wait while ALCS processes new input transactions, which might in turn request create monitor services, and so on. This could cause a deadlock in ALCS.

But because of this, a parent entry that creates many child and grandchild entries can flood ALCS with entries, and so lock out new transactions. Application programs can avoid this by regulating their use of create-type services with the `LODIC` monitor service (or the `lodic` C language function). But note that transactions that create only one or two entries should *not* use `LODIC` in this way.

Avoiding create-type services in loops

Parent entries that create many child entries typically do this by issuing a create-type macro in a loop. The parent entry can lose control if there are too many entries in the system. There may be too many entries because the parent entry is issuing many create-type services. This situation is compounded if the child entries themselves loop creating grandchild entries.

Other, unrelated, entries that request create monitor services may also be forced to wait in this situation.

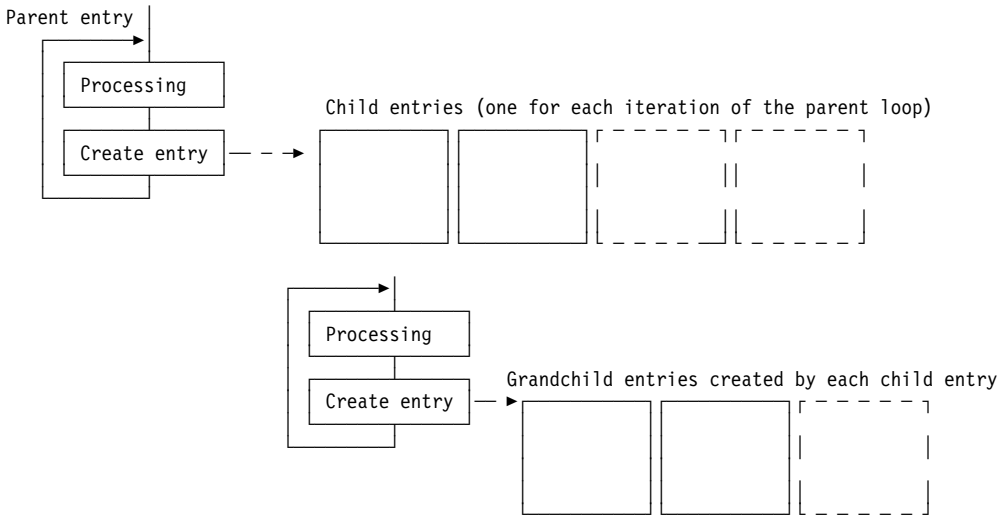


Figure 82. Multiple entries: create-type services in loops

In this way, the ALCS system can be flooded with entries, all of which are trying to request create-type services – but they cannot do so because there are already too many entries in the system.

You can avoid this problem by ensuring that the parent entry regulates its use of create-type services with `LODIC`, and the child entries do not request more than one or two create-type monitor services. Grandchild entries should not request create monitor services at all.

Holding resources for too long

If a parent entry is holding a record or other resource (or has a general sequential file assigned) while it is looping, creating child entries, it may lose control because there are already too many entries in the system. This is similar to, but worse than, the situation shown in Figure 82.

In additon to the danger that other entries might also be waiting to request create-type services, a queue of entries can build up, all waiting to hold the record or resource or to assign the general sequential file.

This is particularly likely to occur if child or grandchild entries need to hold the record or resource, or to assign the general sequential file.

Generally, it is wise for application programs to unhold records or other resources (or reserve general sequential files) before requesting create-type services – especially large numbers of create-type services.

6.8 Entries using SQL, CPI-C, APPC, MQI and TCP/IP

6.8.1 Normal and abnormal termination

SAA defines some automatic processes that occur as part of normal or abnormal termination when application programs use SAA common programming interfaces such as SQL and CPI-C.

ALCS performs these processes when an entry terminates.

Normal termination: ALCS defines normal termination as any of:

- Assembler language `EXITC` monitor-request macro
- Assembler language `BACKC` monitor-request macro when there is no calling program.
- C `exit` function
- C `return` function when there is no calling program.

Abnormal termination: ALCS defines abnormal termination as any system error with exit, including:

- Monitor detected errors that exit the entry.
- Assembler language `SERRC`, `SYSRA`, or `SNAPC` macro, with exit option.
- C `serrc_op` or `snapc` function, with exit option.

6.8.2 SQL threads and application processes

DB2 for z/OS publications refer to **threads**. Corresponding SAA publications refer to **application processes**. Threads and application processes are the same, and correspond to ALCS entries.

For example, SQL associates a **cursor** with a thread or application process. Similarly, it associates a **unit of recovery** with a thread or application process.

In ALCS, SQL cursors and units of recovery are associated with entries. An SQL cursor or unit of recovery is preserved across ENTER/BACK linkages for one entry, but cannot be passed between entries.

The first SQL statement that an entry executes identifies the entry as an SQL thread. When the entry terminates, that SQL thread ceases to exist. A normal termination implies a COMMIT for any unit of recovery not yet terminated; an abnormal termination implies a ROLLBACK for any unit of recovery not yet terminated (see 6.8.1, “Normal and abnormal termination”).

ALCS and DB2 for z/OS restrict the number of SQL threads that can exist at the same time (within one ALCS).

See the description of the DB2 parameter in SCTGEN macro in *ALCS Installation and Customization*.

6.8.3 CPI-C and APPC transaction programs

CPI-C and APPC/MVS publications refer to LU 6.2 conversations between **transaction programs**, **application programs**, or **programs**. These are all the same, and correspond to ALCS entries (**not** to ALCS application programs). An LU 6.2 conversation is preserved across ENTER/BACK linkages for one entry, but cannot be passed between entries.

Entries that initiate or accept LU 6.2 conversations should deallocate them explicitly before exiting. If they do not, ALCS takes the following action:

- When an entry terminates normally, ALCS deallocates any conversations that are still allocated (provided they are in the correct state).
- When an entry terminates abnormally, ALCS deallocates with an abend condition (Deallocate_abend) any conversations that are still allocated.

6.8.4 MQI transaction programs

In WebSphere MQ for z/OS, an open object is identified by its object handle. In ALCS, object handles are associated with entries. An object handle is preserved across ENTER/BACK linkages for one entry, but cannot be passed between entries.

Entries that open MQ objects should close them explicitly before exiting. If they do not, ALCS closes any objects that are still open.

6.8.5 TCP/IP Sockets transaction programs

In TCP/IP for MVS or Communication Server, a socket is identified by its socket descriptor. In ALCS, socket descriptors are associated with entries. A socket descriptor is preserved across ENTER/BACK linkages for one entry, but cannot be passed between entries.

Entries that create TCP/IP sockets should close them explicitely before exiting. If they do not, ALCS closes any sockets that are still open.

Chapter 7. Storage management

This section describes ALCS entry and high level language storage management.

7.1 Storage layout

ALCS runs as a single MVS region. The storage associated with the region includes:

Home-address space Contains:

- MVS
- ALCS
- ALCS application programs
- All storage accessible by ALCS application programs
- Most of the ALCS tables (including VFA)

Dataspaces ALCS uses dataspaces to hold tables that it cannot conveniently keep in the home address space. Application programs cannot access these tables.

Hiperspaces ALCS can (optionally) use Hiperspace to provide additional buffering for DASD I/O. This Hiperspace buffering is in addition to the buffering provided by VFA.

The remainder of this section describes the storage in the home address space.

ALCS requests MVS to make the address space non-swappable. After obtaining storage with the MVS `GETMAIN` macro, or loading a module, ALCS requests the MVS `PGSER` macro to fix the storage. For storage above the bar, ALCS issues the MVS `IARV64` macro to obtain and fix the storage if required

The `PAGE` parameter of the ALCS generation macro `SCTGEN` can override this action for some storage areas. Figure 83 shows where the `PAGE` parameter can be used, and how the ALCS initialization process formats the user area of the MVS address space.

Figure 83 (Page 1 of 2). Protected storage locations and characteristics

	Below 16MB	**Page fixed**	`PAGE=` **override**
Monitor			
ALCS monitor program	Yes	Yes	
Monitor work areas	Yes	Yes	
Monitor tables area	Yes	Yes	
System configuration table	No	Yes	
Area for I/O control blocks	No	Yes	
DASD I/O			
DASD configuration table	No	Yes	
Data set descriptor blocks	No	Yes	
VFA record locator table	No	Yes	
VFA buffer headers	No	Yes	
VFA data buffers	Note 5	Yes	`ALL` or `VFA`
Sequential file configuration table	Yes	Yes	
Sequential file buffers	No	Yes	

Figure 83 (Page 2 of 2). Protected storage locations and characteristics

	Below 16MB	Page fixed	PAGE= override
Communication			
Communication configuration table	No	No	
Communication hash tables	No	No	
SLC control blocks, including I/O buffers (Note 1 on page 112)	Yes	Yes	
SLC I/O message buffers (Note 1 on page 112)	No	No	
SLC link and channel keypoints (see Figure 84 on page 112)			
APPC data and vector table	No	Yes	
TCP/IP extended buffer	Note 6	No	
Application programs			
Program configuration table	No	Yes	
Module load table	No	Yes	
Program hash table	No	Yes	
Program control table	No	Yes	
Application program load modules	Note 2 on page 112	Yes	ALL or PROGRAM

Figure 84. Unprotected storage locations and characteristics

	Below 16MB	Page fixed	PAGE= override
Application storage			
Application global area 1	Note 3	Yes	ALL or GLOBAL
Application global area 2	Note 3	Yes	ALL or GLOBAL
Application global area 3	No	Yes	ALL or GLOBAL
Storage units	Note 4	Yes	ALL or STORE
Communication			
SLC link and channel keypoints (Note 1)	No	No	

Notes:

1. Storage is allocated only if there is an SLC network.
2. Specify AMODE*nn* at link-edit time for ALCS to load programs:

   Below 16MB, AMODE24 (24-bit addressing)
   Above 16MB, AMODE31 (31-bit addressing).
3. Global areas 1 and 2 are below 16MB if you specify AMODE31=(FORCE,NOTGLOBAL) in SCTGEN, or omit AMODE31. They can be anywhere if you specify:

   AMODE31=(FORCE,GLOBAL) or
   AMODE31=FORCE

   in SCTGEN.
4. Storage units can be anywhere if you specify AMODE31=FORCE.
5. If you specify SCTGEN AMODE64=VFA in your system configuration, then ALCS will obtain page-fixed virtual storage needed for the VFA buffers above the bar.

If you do not specify SCTGEN AMODE64=VFA in your system configuration, then ALCS will obtain virtual storage needed for the VFA buffers above the line, but below the bar.

6. Storage is above the bar.

7.2 Real and virtual storage

z/OS is a Multiple Virtual Storage system where each batch job or started task has its own virtual address space. ALCS makes use of virtual addresses which are transformed to reference real storage, by hardware use of the Page and Segment tables. The availability of 64-bit addressing allows ALCS to use real and virtual memory above the 2GB bar, thus removing memory constraints and freeing memory for use by applications and other subsystems.

7.3 Entry storage

1.7.2, “Entry control block” on page 35 describes how each entry requires an area of storage for its ECB. During processing, the entry requires additional areas of storage; for example, to contain an input message, to read or write a DASD record, to contain DECBs, to construct and send a response message, and so on. The storage for the ECB, together with these additional areas of storage, is called **entry storage**.

The ALCS online monitor allocates additional storage to the entry in blocks of a predetermined length, called **storage blocks**. ALCS allows up to nine different storage block sizes (Figure 31 on page 33 shows these sizes). For size L0, the block size is the same as the block size for size L1, but only 127 bytes are available to the application program.

7.4 Heap storage used by assembler programs

Assembler programs may use the CALOC, MALOC, RALOC, and FREEC monitor-request macros to obtain or release heap storage.

Whenever an assembler program obtains heap storage, a type 3 storage unit is allocated, in addition to any other storage units that are already in use.

If the program requires more heap storage than fits in the first type 3 storage unit, ALCS allocates additional type 3 storage units.

You need to be aware that the largest contiguous amount of heap storage that a application can use is a whole type 3 storage unit. You must choose the size of your type 3 storage units so that they are large enough to hold the largest variable (which can be an array or structure) that your application uses.

Note: The system programmer allocates the number and size of type 3 storage units using the NBRSU= and SUSIZE= parameters of the SCTGEN system generation macro, as described in *ALCS Installation and Customization*.

7.5 High-level language storage

ALCS provides services that allow C language application programs to obtain, use, and release standard size storage blocks (L0, L1, and so on). C language application programs can also access fields in the ECB. But ALCS does not provide these services for other high-level languages.

The usual way for high-level language (including C) programs to obtain and use storage is by using dynamic variables. C language programs also obtain storage using the `malloc`, `calloc`, and related functions.

At execution time, high-level language programs obtain and manage storage to contain dynamic variables and to satisfy C language `malloc`, `calloc`, and related function calls. They also obtain and manage storage for register save areas, for library routines to use as work areas, and so on.

Application programmers do not need to understand in detail how their programs obtain and manage this storage at execution time. The compiler generates code that obtains the storage, as required, by requesting ALCS services. The services allocate *entry* storage so that different transactions use different storage even when the same program is executing. Note that these services are intended for use *only* by compiler-generated code – application programmers should not attempt to invoke the services directly.

7.5.1 Initial storage allocation

When an ALCS ENTER-type service transfers control to a high-level language program, the high-level language application program obtains a storage area called the **initial storage allocation** (ISA).

The size of the ISA depends on the version and release of the compiler and library environment, but is on the order of 100KB.

The ISA storage is released when the high-level language program returns to the calling program, or if it exits without returning control.

Note that the compiler automatically generates code to obtain and release the ISA. The high-level language source code does not include any instructions to do these things.

7.5.2 Stack and heap storage

High-level language programs use **stack** and **heap** storage to contain dynamic variables. C language programs can also obtain heap storage using the `malloc`, `calloc`, and related functions.

During execution, the programs obtain and release stack and heap storage when required. The compiler generates code that manages this storage. This code does not request storage separately for each variable. Instead, it requests storage in amounts that are a multiple of 64KB for stack storage and 256KB for heap storage. (In other environments the amounts can be multiples of a different constant.) Typically a program can store many variables within 256KB, but it is possible that a single variable (for example a large array) can require more than 256KB. For a description of how stack and heap are used, see *z/OS Language Environment Programming Guide*.

When the program requires stack or heap storage, it requests the required amount (in multiples of 64KB and 256KB respectively) from ALCS. ALCS allocates stack storage in **type 2 storage units** (also called **HLL storage units**). ALCS allocates heap storage in **type 3 storage units**.

Whenever an ALCS ENTER-type service transfers control to an HLL program, the entry needs at least one **type 2 storage unit** (to hold the ISA) additional to any storage units it is already using. The type 2 storage unit size must be at least large enough to hold the ISA – any remaining space in the first type 2 storage unit is available for use as stack storage.

If the program requires more stack storage than fits in the first type 2 storage unit, ALCS can allocate additional type 2 storage units. If the program requires more heap storage than fits in the first type 3 storage unit, ALCS can allocate additional type 3 storage units.

You need to be aware that the largest contiguous amount of stack storage that an application can use is a whole type 2 storage unit. You must choose the size for type 2 storage units so that they are large enough to hold the largest stack frame that any HLL program requires (the stack frame includes storage for all function-scoped variables defined in the function, plus the save area and other work areas used by the compiled code). Similarly, the largest contiguous amount of heap storage that an application can use is a whole type 3 storage unit. You must choose the size for type 3 storage units so that they are large enough to hold the largest area requested by `malloc` or a similar C language function.

Note: The system programmer allocates the number and size of type 2 and 3 storage units using the NBRSU= and SUSIZE= parameters of the `SCTGEN` macro, as described in *ALCS Installation and Customization*

7.6 Storage units

The ALCS online monitor satisfies all requests for entry storage from areas of storage called **storage units**. Each entry needs at least one storage unit. This is called the **prime storage unit**. Both the ECB and the ECB prefix are in the prime storage unit. The ALCS online monitor uses the ECB prefix to hold information about the entry (for example, the entry macro trace details). Application programs must not alter any part of the ECB prefix. The ALCS online monitor keeps critical details of the entry in the ECB descriptor for the prime storage unit. Application programs cannot modify these details because the ECB descriptors are in protected storage.

The remainder of the prime storage unit is available to satisfy storage block requests. If the ALCS online monitor cannot service a storage block request from the prime storage unit, it allocates an **overflow storage unit** to the entry. The ALCS online monitor then allocates storage requests from the overflow storage unit. The entry can overflow into further overflow storage units until its total storage allocation exceeds its entry storage limit. Overflow storage units do not contain an ECB prefix or an ECB.

ALCS uses a separate storage unit to contain DECB information. The first time the ALCS online monitor receives a request to create a DECB for the entry, it allocates a new overflow storage unit for the entry and reserves the first section for use as part of a DECB frame. The remainder of this overflow storage unit is available to

satisfy storage block requests. The ALCS online monitor keeps critical DECB data in another part of the DECB frame, referred to as the DECB descriptor, which is in protected storage. A DECB frame contains multiple DECBs. If the ALCS monitor cannot service a create DECB request from this DECB frame it will allocate a new overflow storage unit to contain a new DECB frame.

ALCS uses different types of storage unit for allocating heap storage (for assembler and high-level language application programs) and stack storage (for high-level language application programs). Figure 85 shows entry storage using a prime and overflow storage units, an HLL storage unit, and a heap storage unit.

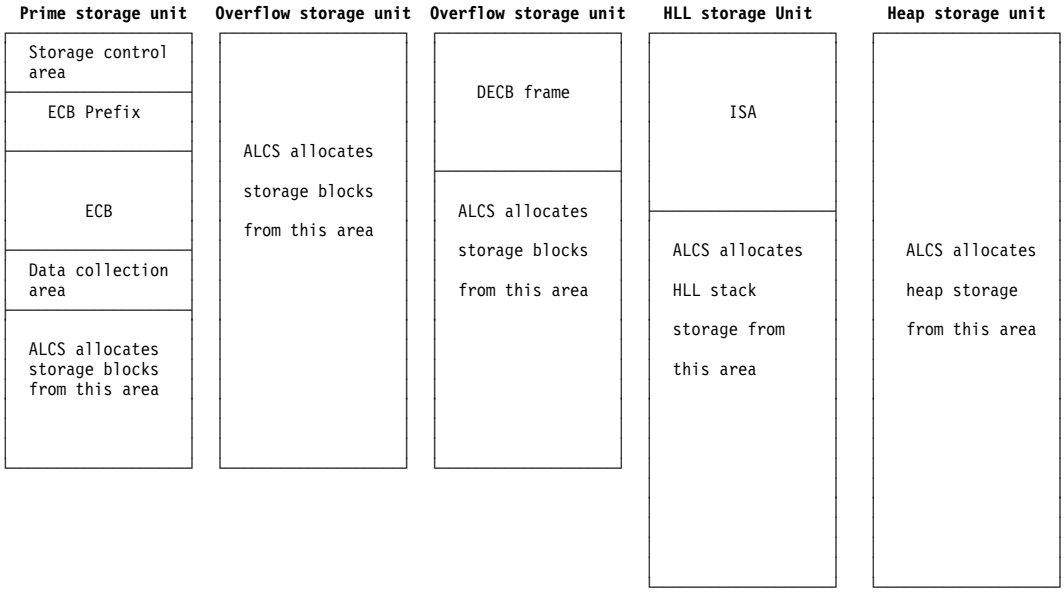


Figure 85. Entry storage: Prime and overflow storage units

Use the NBRSU= and SUSIZE= parameters of the SCTGEN macro (in the ALCS generation) to specify the number of storage units and the storage unit size. The storage unit size must be larger than the largest storage block size. When specifying the storage unit size, consider the following:

- Too large a storage unit size is economical on time but extravagant on storage. That is, although few entries need an overflow storage unit, a lot of storage is unused.
- Too small a storage unit size is economical on storage but extravagant on time. That is, although little storage is unused, most entries need an overflow storage unit.
- The maximum amount of stack increment that ALCS can allocate for an HLL program is one HLL storage unit. Therefore the HLL storage unit must be large enough to hold the largest stack frame that any HLL program requires, where the stack frame includes storage for all function scoped variables defined in the function plus the save area and other work areas used by the compiled code.
- The type 3 storage unit size must be large enough to contain the largest single area requested by MALOC or a similar assembler macro, or by malloc or a similar C language function.

Specify the storage unit size so that the frequently occurring types of entry need only one storage unit (or two if using DECBs).

Note: High-level language application programs need at least one HLL storage unit as well as the prime storage unit.

Use the ENTSTOR1, ENTSTOR2, and ENTSTOR3 parameters of the SCTGEN macro to set the initial entry storage limit and the maximum entry storage limit for different storage unit types. The entry storage limit for each entry is this initial value, unless the entry requests a SLIMC monitor service to alter it. SLIMC can decrease the limit, or increase it up to the maximum entry storage limit. The ALCS online monitor terminates the entry with a dump when the total size of the storage units required by the entry exceeds the storage limit. Due to fragmentation of storage within the storage units, the total amount of storage actually in use by the entry can be less than the entry storage limit when the entry is terminated. Be sure to consider this fragmentation when setting the initial and maximum entry storage limits.

When ALCS initialization allocates storage for the storage units, it allocates a page as a trap between each storage unit. The monitor sets the key of these trap pages to 7; it sets the storage unit key to 8. This provides some protection from corruption of storage due to application program errors.

ALCS uses these **trap pages** to contain control information (for example the ECB and DECB descriptors).

Chapter 8. Automated operations

NetView offers a variety of system and network management capabilities for the MVS environment including support for automated operations. The NetView Program-to-Program Interface (PPI) allows application programs (for example ALCS) to exchange data with NetView or with another application running in the same MVS image. NetView acts as a server and manages the queues and message forwarding services to enable CRAS operation from NetView operator terminals.

With the appropriate changes defined, ALCS uses the PPI to forward messages to the ALCS Prime, RO and Alternate CRAS terminals.

Messages sent to these CRAS devices may be selected for automation and thereby initiate a programmed response. A typical programmed response will be controlled by a NetView command list (CLIST) program written in REXX*.

ALCS events that require a programmed response are identified by the entry of specific messages in the NetView message automation table. For example the programmed response can be:

- Responding to ALCS with an operator command
- Highlighting the message on NetView
- Producing a Network Problem Determination Application (NPDA) alert

Use of automated operations techniques provide several benefits, these include:

- Improvement in system and network availability
- The ability to respond quickly to any system message
- Accurate and consistent message response 24 hours a day
- Reduced levels of human intervention
- Unmanned management of remote systems

Appendix A. ALCS pool file support

This section describes the pool file support that ALCS Version 2 Release 4.1 provides.

A.1 Recoup and emergency pool recovery

A short-term pool-file record can be reused immediately after the application program releases it.

ALCS does not automatically reuse long-term pool-file records. These are only made available for reuse by emergency pool recovery (PDU) or by running Recoup. Recoup is a database validation routine that runs under the control of the ALCS online monitor, without interrupting normal message processing. Based on parameters provided by the database administrator, Recoup checks all chains of long-term pool file records and identifies the records that can safely be used.

The emergency pool recovery (PDU) function is a process that automatically reuses released records when a pool is exhausted. See *ALCS Installation and Customization* for a description of PDU.

"Pool files" on page 64 gives an overview of pool file usage.

A.2 Long-term pool support – type 1 and type 2

The long-term (LT) pool support supplied in ALCS/MVS/XA and ALCS Version 2 Release 1.1 is called **type 1 LT-pool support**.

ALCS Version 2 Release 1.3 and later releases provide type 1 LT-pool support and also an improved support called **type 2 LT-pool support**, which can be used instead.

For both type 1 and type 2 LT-pool support ALCS provides integrity for long-term pool records by the use of:

- Pool directory records
- Record control fields in the long-term pool records
- The ALCS database-validation utility called Recoup

LT-pool directories are built and written to the database by Recoup. They are not keypointed at other times. Recoup comprises two phases as follows:

- In the first phase, called **chain chase**, Recoup identifies all the records that are in use and stores the information in the Recoup general file.
- In the second phase, called **directory build**, Recoup builds the directories using the information in the Recoup general file (placed in the file during the first phase).

The directories contain 1 bit for each long-term pool record to show if that record is available for dispense or is already in use. However, the pool-file management routines use the long-term pool directories only as a preliminary indication of the status of the record.

The routines determine the exact status of the record from control information held in the record itself. ALCS places this control information in an area of the record that application programs do not normally reference.

Whenever the pool file management routines dispense, write, or release a long-term pool record they check and update this control information. Pool file management routines also check the control information when application programs read the record.

Each time these routines detect an error (for example, a program writing an undispensed record) they write a pool-record usage error message to the ALCS diagnostic file. You can print these pool record usage error messages by running the ALCS diagnostic file processor with `POOLERR=YES`.

ALCS Operation and Maintenance gives details of the error reports and diagnostic actions.

ALCS does not make long-term pool records available for reuse immediately when application programs release them. Instead, the pool file management routines mark the release in the control information in the DASD record. ALCS will redispense the record when:

- Recoup is run to confirm that there are no data structures referencing the record.
- The ALCS emergency pool recovery function is activated for record dispensing.

Long-term pool control information

The control information in the long-term pool records consists of four fields. Each field contains a 4-character program name and a time (the high-order 4 bytes of the time-of-day (TOD) clock). The four fields are:

Dispense Set when ALCS dispenses the record to an application. ALCS clears the other 3 fields.
Write Set when an application writes the record.
Recoup Set when Recoup chain-chase reads the record.
Release Set when an application releases a record.

ALCS checks these fields against information in the control program keypoint (CTKB). The pool-file management routines update the information in CTKB which contains, (among other items), the start time of the last completed Recoup run.

A.2.1 Type 1 long-term pool support

With type 1 LT-pool support it is not possible to increase the number of records in an LT-pool, or to add LT-pools of a different record size, without an ALCS outage and a time consuming database reorganization.

With type 1 LT-pool support only a portion of the LT-pool directories is held in main storage at any one time. This means that a number of scans of the Recoup general file are required during directory build. Directory build can take several hours on a large system.

A.2.2 Type 2 long-term pool support

Adding LT-pool records

With type 2 LT-pool support it is possible to increase the number of pool records in the LT-pool, and to add LT-pools of a new record size (in NORM state without a restart of ALCS). You can do this by:

- Adding one or more data sets
- Increasing the size of one or more data sets

ALCS Installation and Customization gives more information about specifying database requirements.

A.2.3 Migration from type 1 LT to type 2 LT-pool support

Migration from type 1 LT-pool support to type 2 LT-pool support is controlled by means of the `DBHIST` macro in the DASD generation. *ALCS Installation and Customization* gives more information about `DBHIST`.

This migration can be done in NORM state without a restart of ALCS.

You can fall back from type 2 LT-pool support to type 1 LT-pool support by **backing out** the DASD generation which specified type 2 LT-pool support.

You cannot do this after the load of the DASD generation is **committed**. *ALCS Operation and Maintenance* describes this procedure.

When you load a new DASD generation which calls for a migration (from type 1 LT-pool support to type 2 LT-pool support) the migration takes effect at the completion of the next Recoup run. Similarly fallback (from type 2 LT-pool support to type 1 LT-pool support) takes effect at the completion of the next Recoup run.

Note: Once you run Recoup to activate type 2 LT-pool support you **cannot** safely fall back to ALCS/MVS/XA or ALCS Version 2 Release 1.1 (even if you fall back to type 1 LT-pool support first). This is because your database may contain embedded references to type 2 LT-pool file addresses. ALCS/MVS/XA and ALCS Version 2 Release 1.1 cannot support these file addresses.

A.2.4 Performance

In type 2 LT-pool support the pool directories are held in main storage all the time. The directory build of all LT-pool directories can be performed with one scan of the Recoup general file. This is considerably faster than the directory build for type 1 LT-pool support.

A.3 Short-term pool support – type 1 and type 2

The short-term pool support supplied in ALCS/MVS/XA and ALCS Version 2 Release 1.1 is called **type 1 ST-pool support**.

ALCS Version 2 Release 4.1 provides type 1 ST-pool support and also an improved support called **type 2 ST-pool support**, which can be used instead.

A.3.1 Type 1 short-term pool support

Type 1 ST-pool support controls the use of pool by a directory for each pool. The directory contains 1 bit for each pool record which indicates whether the pool record is available for dispense. Logically there is 1 bit for every record in the pool. Physically, however, only 8256 bit switches for each ST-pool are held in main storage at one time so that only a maximum of 8256 records are properly controlled.

Because of the small amount of ST-pool information held in the directory it is not possible for ALCS to detect most ST-pool usage errors. Errors in application programs, such as writing a record after it has been released, are not detected by ALCS and can cause corruption of the database. Once a record is released by an application it can be dispensed again immediately and this increases the probability of application program errors resulting in database corruption.

With type 1 ST-pool support it is not possible to increase the number of records in the ST-pool, nor is it possible to add ST-pools of a new record size, without an ALCS outage and a time-consuming database reorganization.

A.3.2 Type 2 short-term pool support

Type 2 ST-pool support also controls the use of pool by a directory for each pool. The directory contains 1 byte of information for each record in the pool. The entire directory is held in main storage all the time ALCS is running (except for a short time after ALCS restart). The directory is keypointed to the database at appropriate intervals and reloaded into main storage from the data base if ALCS is restarted.

Contents of directory byte

The contents of the directory byte for a short-term pool record show the status of the record. The status is usually one of:

- **Never dispensed**

  The contents of the directory byte is 0. The record is available for dispense.

- **Record dispensed and not yet released**

  A range of values in the directory byte is reserved for records which are dispensed and not yet released. This range is 50 through 239.

  When a record is dispensed, ALCS sets the corresponding directory byte to a number in this range. Over a period of time ALCS decrements the directory byte for a record which is dispensed but not released to the low end of the range (50).

  If the directory byte reaches this low value it usually indicates an error in the application, because either:

  – The application has finished using the record and failed to release it.
  – The application used a short-term record when it should have used a long-term record.

  ALCS redispenses the record and places a **timed out** attention message on the diagnostic file.

  ALCS searches the directory to find records which are available for dispense. The length of time between dispense and redispense of a timed-out record depends on the time to search a directory. This is a function of both:

- System activity,
- The number of directory searches between dispense and timed-out status.

The number of searches is called the *N1* value. The default is 100, but you can use the ZPOOL operator command to set any value in the range 1 thru 189.

ALCS Operation and Maintenance describes the ZPOOL command.

Alterations by the ZPOOL command are kept across a restart. You can use the USRSTD installation-wide monitor exit to alter the *N1* value for individual records.

ALCS Installation and Customization describes the USRSTD installation-wide monitor exit.

Note: If you set *N1* to a low value the record may be treated as **timed out** when the application still needs it.

- **Record released and not yet redispensed**

  A range of values in the directory byte is reserved for records which have been released and not yet redispensed. This range is 1 through 25.

  When a record is released ALCS sets the corresponding directory byte to a number in this range. Over a period of time ALCS decrements the directory byte for a record which has been released but not redispensed to the low end of the range. This is 1.

  When the directory byte reaches this low value ALCS assumes the record is available for redispense.

  The length of time between the release and the redispense of a record depends on the time to search a directory. This is a function of both:

  - System activity
  - The number of directory searches between release and redispense

  The number of searches is called the *N2* value. The default is 1, but you can use the ZPOOL operator command to set any value in the range 1 through 25.

  ALCS Operation and Maintenance describes the ZPOOL command.

  Alterations by the ZPOOL command are kept across a restart. You can use the USRSTR installation-wide monitor exit to alter the *N2* value for individual records.

  ALCS Installation and Customization describes the USRSTR installtion-wide monitor exit.

  Note: If the *N2* value is set to a high value there may be performance implications because ALCS needs to do extra directory searches.

  With type 2 ST-pool support the length of time between release and redispense is typically much greater than for type 1 ST-pool support. This reduces the probability that application programming errors result in database corruption.

Reserved values of the directory byte

Other values of the directory byte are used by ALCS to control special situations such as restart after an unplanned outage.

ST-pool usage errors

ALCS uses the information (in the directory byte) on the status of each ST-pool to detect application errors in ST-pool usage. When ALCS detects such an error, it places an **Attention** message in the diagnostic file.

You should use the information in the error messages to locate and correct application programming errors and so prevent database corruption. Error messages are produced in the following situations:

- Write of a record that is:
  - Already released
  - Never dispensed
  - Timed out
- Read of a record that is:
  - Already released
  - Never dispensed
  - Timed out
- Release a record that is:
  - Already released
  - Never dispensed
  - Timed out
- Dispense of a timed-out record

ALCS Operation and Maintenance describes pool usage errors.

A.3.3 Migration from type 1 ST to type 2 ST-pool support

You control migration from type 1 ST-pool support to type 2 ST-pool support (or back from type 2 to type 1 if required) with the DBHIST macro in the DASD generation.

ALCS Installation and Customization describes the DBHIST macro.

You can do these migrations in NORM state without a restart of ALCS.

A.3.4 Tagging of ST-pool records

Before a record can be dispensed by type 2 ST-pool support it must be identified as an ST-pool record by a process known as **tagging**. The Recoup process performs the tagging of ST-pool records. This means that type 2 ST-pool support is not used until at least one Recoup run has completed.

A.3.5 Adding ST-pool records

With type 2 ST-pool support it is possible to increase the number of records in the ST-pool, and it is possible to add pools of new record sizes, in NORM state without a restart of ALCS. This is done by a new DASD generation and the ZDASD command.

ALCS Operation and Maintenance describes the ZDASD command.

These new records and new pools are not available for dispense if you fall back to type 1 ST-pool support.

When ST-pool records are added, or new ST-pools are added, by means of a new DASD generation, the records are not available for dispense until they are tagged by a Recoup run.

For new ST-pool records to be tagged by Recoup, the DASD generation must be loaded and confirmed.

For new ST-pool records to be dispensed by type 2 ST-pool support, the DASD generation must be loaded and committed.

Ignored requests

When ST-pool records are added, or new ST-pools are added, by means of a new DASD generation, the records are obtained from the LT-pool of the same size. If there are insufficient LT-pool records of that size then ALCS ignores the request to add the ST-pool records.

When ST-pool records are added, ALCS needs a number of L3 records for internal purposes, (whatever the size of the added pool records). Shortage of L3LT-pool is another reason why ALCS may ignore a request to add ST-pool records.

A.3.6 Deleting ST-pool records

When ST-pool records are deleted, or ST-pools are deleted, by means of a new DASD generation, the deleted records cease to be dispensed as soon as the DASD generation has been loaded.

The application to which these records were dispensed can use these records until they are **purged** by a subsequent DASD generation.

When ST-pool records are purged, or ST-pools are purged, by means of a new DASD generation, the purged ST records are added to the LT pool of the same size.

A.3.7 Coexistence

Type 1 ST-pool support can coexist with either type 1 LT-pool support or type 2 LT-pool support.

Type 2 ST-pool support can coexist with either type 1 LT-pool support or type 2 LT-pool support.

This means that a user can move to type 2 ST-pool support without moving to type 2 LT-pool support, and a user can move to type 2 LT-pool support without moving to type 2 ST-pool support.

A.4 Dispense rings

For each LT-pool and each ST-pool file, ALCS keeps a short list of file addresses which are pre-checked as available for dispense to the application. This list is called the **dispense ring**.

When an application requests a pool record, ALCS takes an address from the appropriate dispense ring and passes it to the application.

The use of the dispense ring allows a faster response to the application because:

- ALCS does all the pre-dispense checking before the file address is placed on the dispense ring.
- ALCS takes addresses from the dispense ring in such a way that different dispatcher tasks do not interfere with each other.

ALCS varies dynamically the number of addresses held on a dispense ring to suit the load on the system. If the system is busy it may be as high as several hundred.

A.5 Release rings

For each LT-pool and each ST-pool file, ALCS keeps a short list of file addresses which the application has released but are not yet processed by the post-release checking. This list is known as the **release ring**.

When an application releases a pool record, ALCS places the file address on the appropriate release ring and returns control to the application. Later ALCS removes the address from the release ring and performs the post-release processing.

The use of the release ring allows faster response to the application because:

- ALCS does all the post-release checking without holding up the application.
- ALCS places addresses on the release ring in such a way that different dispatcher tasks do not interfere with each other.

Appendix B. Long-term pool space recovery – Recoup

This section describes the ALCS Recoup utility.

The prime function of Recoup is to recover **long-term pool space** for redispensing. Recoup is not used with short-term pool-file records (these are returned to the system automatically). Recoup identifies long-term pool-file records that are no longer **in use**. To do this, it first identifies all the long-term records that are in use. It then indicates the remaining long-term records as being available.

Released records

When an application program releases a long-term pool-file record, ALCS marks the release in the control information in the pool record itself. The pool file directory record is not updated and the released long-term pool record is not immediately available for redispensing.

Records in use

An application may (by mistake) release a long-term pool record that is used by other applications. Recoup guards against this by verifying that a **released** record is not **in use** by other records before it makes it available.

You should run Recoup regularly to make the long-term pool records available for redispensing.

B.1 Specifying data structures to Recoup

To allow Recoup to identify the long-term pool-file records that are in use, you supply a description of the application database. ALCS provides the `INDEX` macro to describe how one record points to another, and the `GROUP` macro to describe sets of records with the same features.

This section explains the terminology used in ALCS application databases, and also the data structures that the descriptor macros (`INDEX` and `GROUP`) support. *ALCS Installation and Customization* describes how to use these macros.

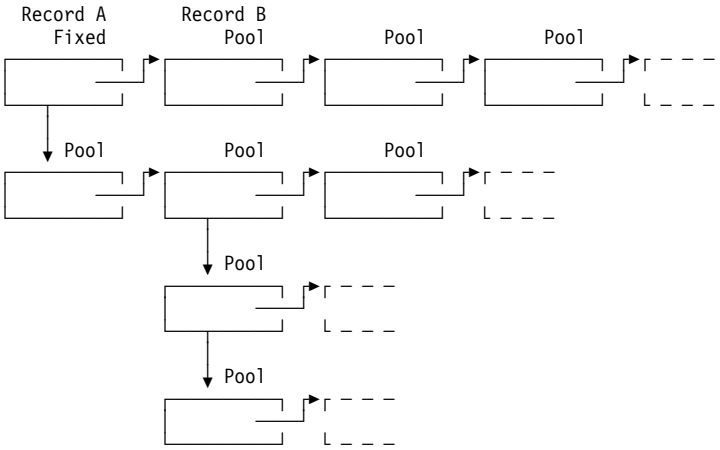
Refer to the following sections for a more general explanation of ALCS data structures:

- 4.1, “The ALCS real-time database” on page 62, gives a overview of the data structures in an application database.
- 4.1.6, “Overflow and chaining” on page 66, describes how records can be linked together.

B.1.1 Chaining pool-file records

When an application uses a pool record to store information, it must record information to identity the pool record that it has used. The simplest way to do this is to store the file address of the pool record in another record. If the address of record B is stored in record A, then record B is **chained** from record A.

A pool record can be chained from a fixed-file record or from another pool-file record, but every chain must start from a fixed file record. Figure 86 shows this data structure.

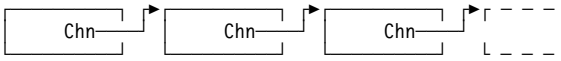


Fixed = Fixed file record Pool = Pool file record

Figure 86. Example of chained records

B.1.2 Standard chain

Two records are connected by a **standard chain** when one record contains the file address of the second record. The reference is contained in a **chain field**. The second record can have chains to subsequent pool-file records, and so on. Figure 87 shows a standard chain.



Chn = chain field

Figure 87. Standard chain

The chain field is normally at the same displacement in every record of the chain. The first record in a chain is called the **first-in-chain**; the last record is called the **last-in-chain**.

There are two types of chain: **forward chain** and an optional **backward chain**. The forward chain address is in the forward chain field. The backward chain address (if there is one) is in the backward chain field.

The forward chain field of the first record contains the address of the second record, the forward chain field of the second record contains the address of the third record, and so on. The forward chain field of the last-in-chain contains zeros.

The backward chain field of each record contains the address of the previous record in the chain. The backward chain field of the first-in-chain points to the last-in-chain. Figure 88 shows forward and backward chains.

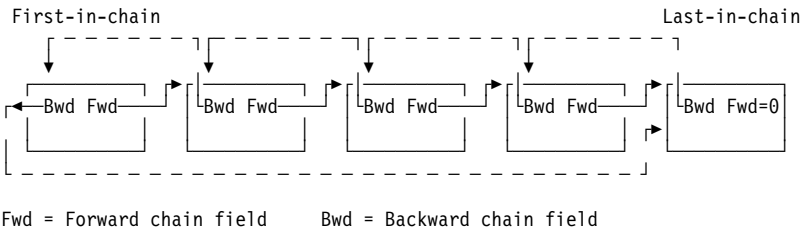


Figure 88. Forward and backward chains

Recoup makes use of these backward chains, if they exist.

B.1.3 Index references

In additon to standard chain fields, records can contain **index references**. Each index reference points to the first record of another group. (Groups are explained in detail in B.2, “Group macro” on page 132.) A reference can be a file address, or some other information. For example, if the reference is to a fixed-file record, it could be an ordinal number. Recoup and the application program can both use this index reference to locate the referenced record.

The record that contains the reference is called the **refer-from** record. The record to which the reference points is called the **refer-to** record. The refer-to record and refer-from record belong to two different groups, the **refer-from** group and the **refer-to** group.

Figure 89 shows three groups of records which summarize the terminology used so far. All the groups contain standard (forward) chains. All the records in Group 1 also contain index references to two other groups.

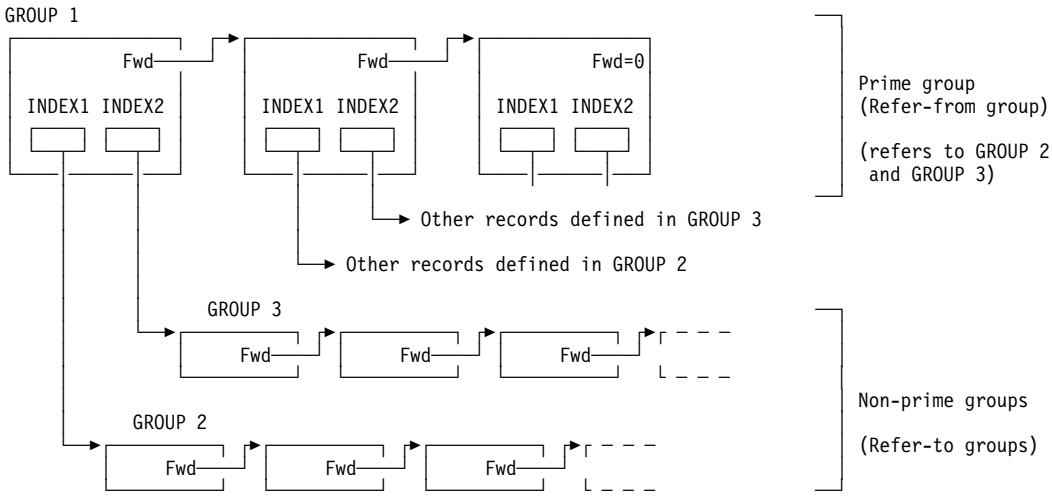


Figure 89. Group of records with index references

The INDEX macro describes the location and characteristics of references to other groups. Group 1 requires one INDEX macro to locate the reference to group 2, and a second INDEX macro to locate the reference to group 3.

B.2 Group macro

Every record that Recoup examines must belong to a **group** of records. A group consists of a number of records that have features in common which you describe to Recoup by using a GROUP macroinstruction.

There are two types of group: **prime group** and **nonprime group**:

B.2.1 Prime group

A prime group normally consists of one or more fixed-file records, all with the same record ID and internal record structure.

Fixed file records in a prime group can be either:

- Records with the same fixed-file record type, and with ordinal numbers in a given range, or
- Records with the record type #GLOBL, all with the same record ID

An additional form of prime group is a logical global record in the application global area.

ALCS also lets you group records into prime groups using conditions that you choose yourself. If you do this, you must provide installation-wide exits to find the records that belong to the group.

ALCS Installation and Customization describes installation-wide exits.

B.2.2 Non-prime group

A nonprime group of records has the following characteristics:

- It consists of one or more pool (and/or fixed) file records, linked together by a standard chain. The first record in the group must be pointed to by an index reference from a record in another group.

  If the chain contains more than one record:

  – All records in the group have the forward chain field at the same displacement from the start of the record

  – All records in the group have the same record ID.

  – The end of the chain must be identified by 4 bytes of hexadecimal zeros or 4 bytes of X'FF' in the forward chain field. (Optionally, you can use an installation-wide exits to implement a different way of identifying the end of chain.

B.2.3 Chain-chasing

Recoup uses a technique called **chain-chase** to work through every defined group of records to find long-term pool file records that are in use.

Recoup starts each chain-chase from a prime group. To read a nonprime group, it first reads a record (the refer-from record) that contains the reference to the first record (the refer-to record) in the nonprime group.

If the refer-from record is itself in a nonprime group, there must be another record that refers to it. So all these references originate directly or indirectly from a prime group.

B.3 Index macro

You use the `INDEX` macro to define where an index reference appears in a record.

The simplest use of the `INDEX` macro is when a reference appears only once in a record. In this case, you specify in the `INDEX` macro the location of the index reference from the start of the record (its **displacement**).

A record can contain more than one index reference to other records. Usually, these index references will be contained in **items** or **subitems** within the record. The `INDEX` macro supports record structures that include items and subitems. Items and subitems can be fixed-length or variable length.

B.3.1 Items

Some physical records contain one or more logical collections of data, each of these logical collections is called an **item**. Figure 90 shows items in a record.

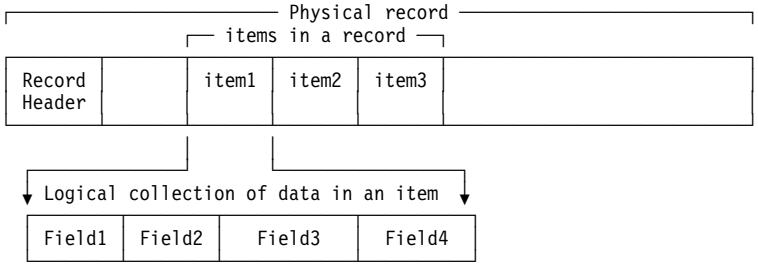


Figure 90. Items in a record, fields in items

In the simplest case, each item in the record has the same format. For example, a record that contains a list of passengers for a flight might contain an item for each passenger. Each item could contain the following fields:

- Passenger name
- File address of a chain of passenger name records (PNRs)
- Boardpoint
- Other fields

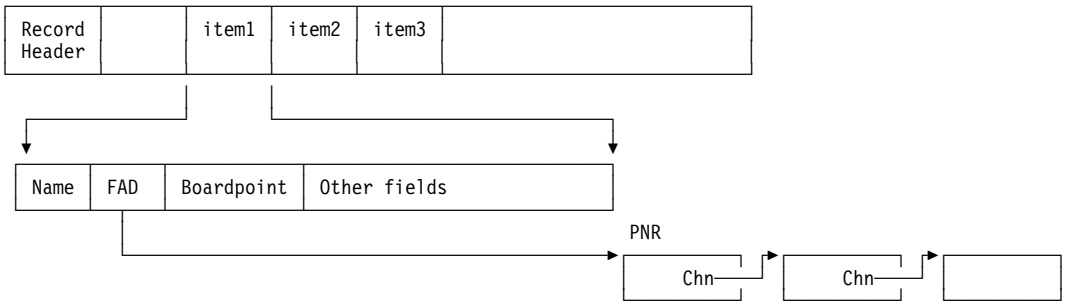


Figure 91. Example fields in an item

The second field in Figure 91 contains a file address field (FAD) which is an index reference to another record.

B.3.2 Variable numbers of items

The INDEX macro supports three different methods for handling items in records. They are as follows:

Item count

You specify either:

Constant The number of items in the record. Use this method if the number of items in the record is fixed.

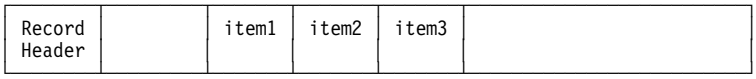


Figure 92. INDEX *macro: Example using a constant for the item count*

Field A field in the record which contains a count of the number of items. Use this method if the number of items in the record is variable.

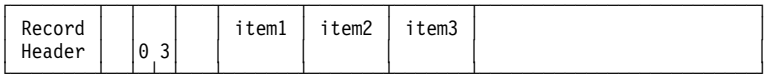


Figure 93. INDEX *macro: Example using a field for the item count*

Next available byte

You specify that the record contains a **next available byte** (NAB) field. The NAB field contains the displacement from the start of the record to the next available byte (for the next item). Figure 94 shows how NAB works when items are in the normal order. Figure 95 on page 135 shows how NAB works when items are in reversed order.

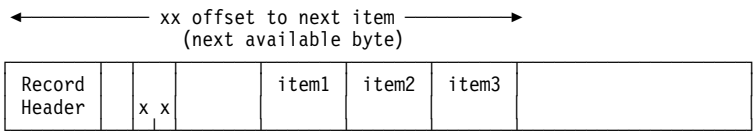


Figure 94. INDEX *macro: Use of NAB (normal order)*

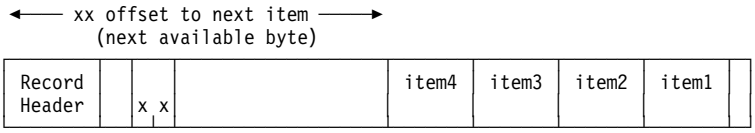


Figure 95. INDEX *macro: Use of NAB (reversed order)*

Add item index (AIX) and delete item index (DIX)

Specify the delete item index (DIX) field in a record to locate the item before the first item in use, and specify the add item index (AIX) to locate the item after the last item in use.

Recoup supports AIX and DIX only for fixed length items. Figure 96 shows the use of AIX and DIX fields.

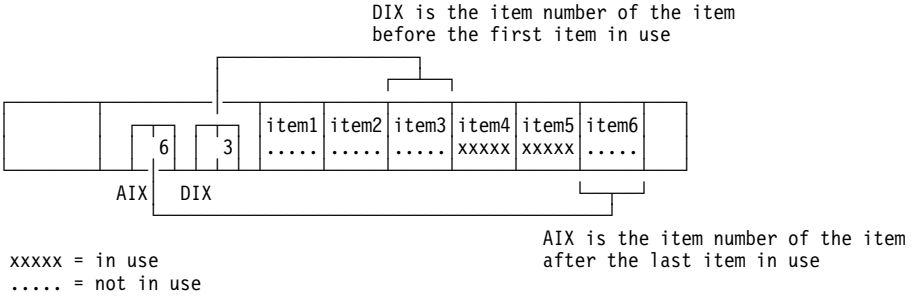


Figure 96. INDEX *macro: Use of AIX and DIX*

B.3.3 Item keys

A record can contain a number of items of different types. In this case, the index reference field is likely to be in different displacements in the record. You need to give Recoup some method of identifying the record type, so that it can find the displacement of the file address.

One way of distinguishing between item types is to have an **item key** at a fixed position in each item to identify its type. This position is usually byte 1 of a fixed-length item and byte 3 of a variable-length item.

You must code a separate INDEX macro for each different item key. Each of these macros specifies the same **list of items** but can, for example, specify a different refer-to GROUP macro. For each of these INDEX macros, Recoup only processes the items with matching item keys.

Notes:

1. If some of the items do not contain references to other groups, you need not code INDEX macros for the corresponding item key or keys.
2. If a record contains different types of items that are not differentiated by standard item keys, then code multiple INDEX macros as described above, but use different user-supplied routines to select the items to process with each INDEX.

B.3.4 Subitems

Items can be broken down into smaller units of data called **subitems**. You can hold an index reference in each subitem and define each reference in an `INDEX` macro.

Subitems can be fixed-length or variable-length.

Appendix C. Communication management for the SLC network

This section discusses SLC concepts and procedures.

C.1 SLC concepts

This section provides a brief overview of SLC concepts. For a full discussion of the SLC protocol, see the *ATA/IATA Interline Communcations Manual*.

ALCS SLC support allows connection between ALCS systems and other systems that support the ATA/IATA SLC processor-to-processor protocol. In particular, it allows connection with processors that are high-level network (HLN) switching centers, and consequently communication between ALCS application programs and communication terminals connected to the HLN.

SLC processor-to-processor communication takes place across an **SLC link**. Each link consists of from one to seven **SLC channels**. The relative number of the channel within the link, from 1 to 7, is called the **link channel number (KCN)**. Each SLC channel consists of two communication lines, known as the **send side** and the **receive side**, which together form a full-duplex connection. Each full-duplex connection is accessed through a pair of EP/VS subchannel addresses (send and receive). The send side of an SLC channel is for outgoing data from the ALCS system. The receive side is for incoming data to the ALCS system. ALCS supports up to a maximum of 255 SLC links.

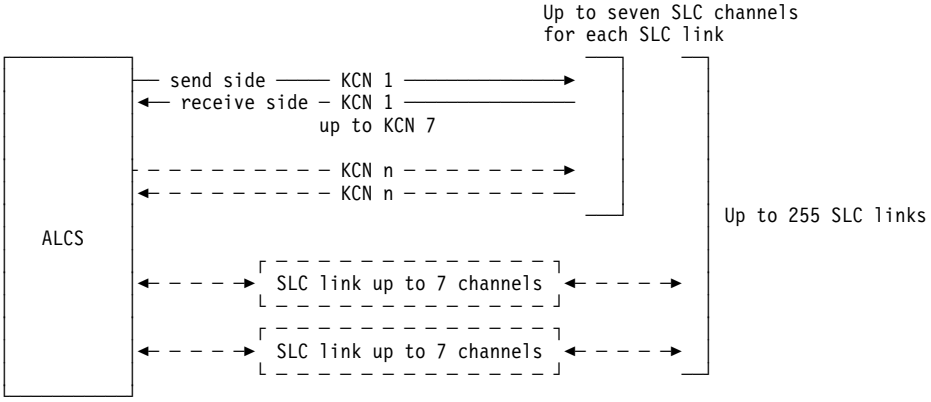


Figure 97. Overview of SLC terminology

Two types of blocks are transmitted across SLC links:

- Link data blocks (LDBs), which transmit data across the link.
- Link control blocks (LCBs), which maintain the integrity of the SLC link; for example, they acknowledge the receipt of LDBs.

C.1.1 LDBs

There are two types of LDB:

- Information blocks, which transmit message data across the link.
- Network control blocks (NCBs), which transmit control data and status information across the link.

In additon to data, LDBs contain control information, such as:

Traffic type
The message data contained in the LDB is:

Type A, low-integrity high-priority
Type B, high-integrity low-priority

Message label
One LDB can contain only a limited amount of data. Long messages occupy several LDBs; message label information in each LDB allows the receiver to reconstruct the message from several LDBs.

When the SLC link communicates with terminals through an HLN, the LDBs contain routing information as well as the message data and control information. This routing information includes:

High-level entry address (HEN)
The 2-byte address that identifies the HLN switching center where the transmitting processor or communication terminal connects.

High-level exit address (HEX)
The 2-byte address that identifies the HLN switching center where the receiving processor or communication terminal connects.

Terminal address
Typically this consists of:

- A 1-byte terminal circuit identity (TCID)
- A 1-byte terminal interchange address (IA)[1]
- A 1-byte terminal address (TA)

In the case of the SITA HLN, only Type A traffic LDBs contain TCID/IA/TA routing information. Type B traffic LDBs contain instead ATA/IATA 7-character message switching destination and origin codes.

C.1.2 SLC terminal addressing

Applications use the terminal's CRI to address a terminal that is connected through an HLN. You provide the information required to convert terminal CRIs to and from SLC addresses (SLC link CRI, terminal HEX, TCID, IA, and TA) in the ALCS communication generation. The SLC address is called the **SLC ID**. An LDTYPE=SLCALC (in the `COMDEF` macro) indicates the definition of an SLC ALC terminal.

[1] This is the address of a terminal control unit.

C.1.3 SLC link characteristics

The owners of processors that connect by an SLC link must define the way the link is to be used. Specify this definition, including the link interval timers (*i* values) and link counters (*c* values), in the ALCS communication generation. An LDTYPE=SLCLINK (in the `COMDEF` macro) indicates the definition of an SLC link.

ALCS generation uses this information to build the communication configuration table.

ALCS supports the following types of SLC links:

- Type 1 SLC protocol
- Type 2 SLC protocol
- Type 3 SLC protocol

C.1.4 Type 1 SLC protocol

The link operates according to the P.1024 specification in the SITA publication *Synchronous Link Control Procedure*.

For input messages, the translation code is in the MCI byte in the envelope of the link data block. For output Type A messages, the translation code is specified using the CODE= parameter for the `COMDEF` macro that defines the terminal. For output messages transmitted by the monitor-request macro `SENDC K` (send direct to link), the translation code is in the message status byte. There are no ACI bytes in the envelope.

The link operates as follows:

- It transmits and receives all the blocks of a multiblock message down the same SLC channel, except under error conditions.
- It transmits and receives clear message label (CML) LCBs as appropriate.
- To stop all the SLC channels, it sends a stop (STP) LCB on each SLC channel.
- If an I/O error occurs during the transmission of a multiblock message, and if another SLC channel is available on the same link, it retransmits only the blocks that have not yet been successfully transmitted.

  It sets and tests the loop test bit in LCBs (except when performing an SLC loop test).
- It acknowledges every link data block.

Note: The response for an enquiry (ENQ) LCB or idle (ILB) LCB is either a resume (RSM) LCB or a stop (STP) LCB.

C.1.5 Type 2 SLC protocol

The link operates according to the Character Oriented Synchronous Link Control specification in *ATA/IATA Interline Communications Manual*.

For input messages, the translation code is in the ACI byte in the envelope of the link data block. For output messages, the translation code is in the message status byte. There is one ACI byte in the envelope.

The link transmits and receives all the blocks of a multiblock message down the same SLC channel, except under error conditions.

C.1.6 Type 3 SLC protocol

The link operates according to the P.1124 specification in the SITA publication *Synchronous Link Control Procedure*. Type 3 also allows the exchange of network control blocks, as described in the same document.

For input messages, the translation code is in the MCI byte in the envelope of the link data block. For output Type A messages, the translation code is specified using the CODE= parameter for the `COMDEF` macro that defines the terminal. For output messages transmitted by the monitor-request macro `SENDC K` (send direct to link), the translation code is in the message status byte. There are no ACI bytes in the envelope.

The link operates as follows:

- It transmits and receives all the blocks of a multiblock message down the same SLC channel, except under error conditions.
- It transmits and receives CMLs as appropriate.
- To stop all the SLC channels, it sends an STP on each SLC channel.
- It sets and tests the loop test bit in LCBs (except when performing an SLC loop test).
- It acknowledges every link data block.

Note: The response for an ENQ or ILB is either an RSM or an STP. If an I/O error occurs during the transmission of a multiblock message, and if another SLC channel is available on the same link, it retransmits only the blocks that have not been successfully transmitted.

C.2 ALCS SLC procedures

This section summarizes the way in which ALCS controls the operation of SLC links and channels. *ALCS Installation and Customization* describes the `COMDEF` macro and the TIMER and COUNTER parameters associated with LDTYPE=SLCLINK. The TIMER parameter defines *i* values, and the COUNTER parameter defines *c* values used in this section.

- An SLC link is **open** when at least one SLC channel of the link is open.
- An SLC link is **started** when at least one SLC channel of the link is started.

C.2.1 Starting a channel

When ALCS starts an SLC channel, it starts a timeout of *i12* seconds running for that channel. ALCS does not send any link control blocks (LCBs) or link data blocks (LDBs) on the channel, and discards any LCBs or LDBs that it receives on the channel, while the timeout is running. During this period the channel is **down**.

When the timeout expires, ALCS sends enquiry LCBs on the channel at intervals of *i4* seconds, but does not send any LDBs. It continues to discard any LCBs or LDBs that it receives on the channel. During this period the channel is **out of service**.

C.2.2 Out-of-service period

For a Type 2 SLC link, the channel remains out of service until ALCS receives an enquiry response LCB on the channel (that is, a resume LCB, a stop LCB, or a positive acknowledgment LCB).

For a Type 1 or Type 3 SLC link, the channel remains out of service until ALCS receives an enquiry LCB followed by an enquiry response LCB on the channel.

When these conditions are met, the channel is **up**; ALCS then sends and receives LCBs and LDBs on the channel according to the SLC protocol for the link.

- An SLC link is up when at least one channel of the link is up.
- An SLC link is down when no channel of the link is up.

C.2.3 When an SLC link changes state

When an SLC link comes up, ALCS:

1. Retransmits any outstanding, unacknowledged, Type B messages that are stored on DASD.
2. Creates an entry to the ALCS printer package for each printer on the link. The printer package then tries to restart each printer. If the printer restarts satisfactorily, the entry enters the application communication exit program ACE1 (if it is loaded) to inform the application that the printer is now available.
3. Creates an entry to application communication exit program ACE2 (if it is loaded) to inform the application that data transmission can start on this SLC link.

If an SLC link goes down, ALCS creates an entry to ACE2 to inform the application that data transmission must stop on this SLC link.

C.2.4 When ALCS changes state

During ALCS state change from idle state to a higher state, ALCS creates an entry to ACE2 (if it is loaded) for every SLC link to inform the application that the SLC link exists.

C.2.5 Positive acknowledgement

If ALCS receives a positive acknowledgment LCB (ACK) for a single block Type B message, or a clear message label (CML) LCB for a multiblock Type B message, ALCS creates an entry to ACE2 to inform the application that a subsequent message can be sent on the SLC link.

C.2.6 Queuing messages on DASD

An application can optionally request ALCS to queue output Type B messages on DASD (`SENDC K` with the `TYPE=QUEUE` parameter). ALCS creates an entry to ECB-controlled program ACE2 as soon as the message is stored on DASD, instead of waiting for an ACK or CML. This can improve throughput, because the application can pass up to eight messages to ALCS without transmission delays.

C.2.7 Idle output line condition

If no LDB is received on an SLC channel during an interval of *i1* seconds, ALCS sends an idle line LCB on the channel (that is, a resume LCB, a stop LCB, or a positive acknowledgment LCB) and repeats this at intervals of *i2* seconds.

C.2.8 Negative acknowledgement (sequence errors)

ALCS sends a negative acknowledgment (sequence error) LCB on the channel if the transmission sequence number (TSN) of an incoming LDB is not the expected next in sequence. ALCS discards the out of sequence LDB, and all subsequent incoming LDBs, until an LDB with the expected next in sequence TSN is received on the channel. The negative acknowledgment LCB is repeated at intervals of *i3* seconds.

C.2.9 Negative acknowledgement (parity errors)

ALCS sends a negative acknowledgment (parity error) LCB on the channel when an incoming LDB has incorrect format, excessive block length, or parity error. ALCS discards the error LDB, and all subsequent incoming LDBs, until an LDB with the expected next in sequence TSN is received on the channel. The negative acknowledgment LCB is repeated at intervals of *i3* seconds.

C.2.10 Error recovery

ALCS uses the SLC channel enquiry procedure to recover from any of the following conditions:

No block received
: No LCB or LDB is received on the channel during an interval of *i9* seconds.

No ACK received
: No acknowledgment is received on the channel within *i5* seconds after the last LDB was sent on the channel.

Too many unacknowledged LDBs
: The number of outstanding, unacknowledged LDBs for the channel reaches *c6* (the transmission sequence number (TSN) exhaustion value).

ATSN error
: An LCB is received on the channel with a zero or illogical acknowledge transmission sequence number (ATSN). If it is an enquiry LCB, ALCS sets all open and started channels of the link down, and follows the procedure for starting SLC channels, as described above.

NAK limit exceeded
: ALCS sends *c2* repeated negative acknowledgment LCBs without receiving the correct LDB.

Stop all channels
: A stop all channels LCB is received; ALCS starts the enquiry procedure on each open and started channel of the link.

C.2.11 SLC channel enquiry procedure

ALCS stops sending LDBs on the channel and sends an enquiry LCB. The enquiry LCB is repeated at intervals of *i4* seconds until an enquiry response LCB is received. Any LDBs received on the channel during this period are discarded. If no enquiry response LCB is received after *c5* repeated enquiry LCBs, ALCS sets the channel down and follows the procedure for starting an SLC channel, as described above.

When ALCS terminates an SLC channel, it sends a stop LCB to stop the exchange of LDBs on the channel. If there are any outstanding, unacknowledged LDBs for the channel, ALCS also sends an enquiry LCB.

C.3 Testing the SLC network

ALCS includes functions designed to test the SLC network. These functions are the link test facility (`ZLTST` command), used for loop tests and functional tests, and the link trace facility (`ZLKTR` command), used for tracing activity on SLC links to a printer, to the ALCS diagnostic file, or to the SLC link trace block.

C.3.1 Performing an SLC loop test

Before a functional acceptance test (see C.3.2, “Performing a SITA functional acceptance test” on page 144), you should consider performing an SLC loop test to check the EP/VS, the communication equipment, and the MVS I/O configuration.

Loop the send and receive sides of the full-duplex communication line through a modem, or through a modem simulator, so that any data transmitted on the send side of the line is received on the receive side of the same line. ALCS then functions as the **send station** and **receive station** on the SLC link simultaneously.

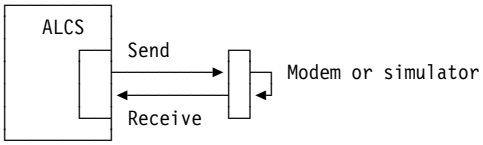


Figure 98. SLC loop test using one channel – with and without modems

Alternatively, you can connect the send and receive sides of one full-duplex communication line to the receive and send sides of another, so that any data transmitted on one line is received on the other. Make sure that each line has the same SLC link channel number (KCN); for example, define them as channel 1 on link 1 and channel 1 on link 2. ALCS then functions simultaneously as the send station on one SLC link and the receive station on the other.

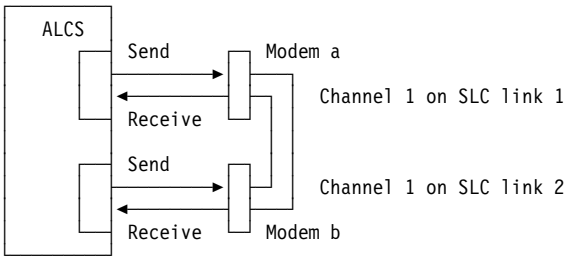


Figure 99. SLC loop test using two channels – and two modems

ALCS Operation and Maintenance provides the information you need to use the ZLTST command for a loop test. Use this command to set test parameters to suit your ALCS configuration.

C.3.2 Performing a SITA functional acceptance test

ALCS is certificated by SITA for SLC connection to the SITA HLN, and a loop test should assure you that all your components of the SLC network are working satisfactorily.

However, before making the connection to the network, you must obtain SITA's approval, and they may require a formal functional acceptance test before you use the links in a production environment.

- If SITA require a test based on the *P.1024 Test Guide*, you can again use the ZLTST command for this purpose, though not necessarily with exactly the same parameters as you used for loop testing.
- If SITA require a UCTF test[2], follow the SITA procedures.

IBM does not supply SITA publications; see “Bibliography” on page 193 for more information on how to obtain them.

C.3.3 SLC link trace facility

Use the SLC link trace facility to obtain a printed record of LCBs and LDBs transmitted or received, or both, on SLC channels. The ZLKTR command controls the link trace facility, and is described in *ALCS Operation and Maintenance*.

[2] See the two SITA publications *P1X24 automatic testing (with UCTF on DIS)* and *P1X24 protocol acceptance tool*.

Appendix D. ALCS services

This section describes the services that ALCS provides for application programmers.

D.1 ALCS services for communication

Application programs use monitor-request macros to request communication operations. *ALCS Application Programming Reference – Assembler* describes the communication monitor-request macros. They are:

`AUTHC` Check whether an entry has authority to access data, ALCS, or application facilities. Allow an entry to retrieve installation-defined data from the external security manager.

`COMCC` Updates an entry in the communication table. An application does not have access to the communication tables and this is the only way to alter an entry in the communication table. Application programs can alter only selected fields.

`COMIC` Obtains information about a communication resource. An application does not have access to the communication tables and this is the only way to obtain information about a resource. The CRI does not contain information about the resource.

`CRASC` Sends a message to the CRAS printer terminal, or to the CRAS printer terminal associated with a CRAS display terminal.

`DISPC` Builds a multiline output message and optionally sends it to a terminal or printer. `DISPC` can invoke the ALCS scrolling package.

`ROUTC` Sends a message to a terminal, to another application, to an MQ queue, or across a communication link (LU6.1, APPC or TCP/IP). The destination terminal or application can be owned (hosted) by the same ALCS system as the originating application, or by another system in the same or in a different processor.

`SENDC` Sends a message. A macro parameter, the type code, allows a variety of message types to be transmitted.

`SLMTC` Sends a special message to a printer. This monitor-request macro enables the application to control the printer. The application is notified when the message has been printed successfully. It is the application's responsibility to handle error conditions that occur on the printer.

`WTOPC` Builds and optionally sends an output message. The message can include a header and variable text. `WTOPC` can suppress blanks (space characters) so that two or more blanks are replaced with a single blank.

The following monitor-request macros are for WTTY communication only. Other application programs should not use them.

`REQSC` Request to send. This monitor-request macro waits for any incoming data on a half-duplex WTTY line to complete, so that output data can be transmitted.

`SCDCC` Checks the status of a WTTY line (before transmitting output data).

STXTC Sends a block (segment) of a multi-block WTTY message. Use STXTC to send all blocks except the last. Use SEOMC to send the last block.

SEOMC Sends the last or only block of a WTTY message.

POLLC Start input on a WTTY line. Signals that transmission of output data on a half-duplex WTTY line is complete, and that input data can be accepted.

The equivalent C language functions which are implemented in this release of ALCS are:

```
comic
routc
```

These are described in *ALCS Application Programming Reference – C Language*.

D.2 ALCS services for DASD processing

ALCS application programs use ALCS monitor services to request DASD I/O. For a more detailed description of the ALCS monitor services that initiate DASD I/O to or from:

- Fixed-file records
- Pool-file records
- General file records

refer to:

- *ALCS Application Programming Reference – Assembler*
- *ALCS Application Programming Reference – C Language*

This following list shows the assembler macros; see the *ALCS Application Programming Reference – C Language* for descriptions of the equivalent C language functions.

FILEC Files (writes) a DASD record.
FILNC Files (writes) a DASD record without detaching the storage block.
FILUC Files (writes) a held DASD record and unholds the record.
FINDC Finds (reads) a DASD record.
FINHC Finds (reads) a DASD record and hold for update.
FINWC Finds (reads) a DASD record. The requesting entry loses control until the read completes.
FIWHC Finds (reads) a DASD record and hold for update. The requesting entry loses control until the read completes.

The following monitor-request macros allocate and release pool file addresses:

GETFC Gets (dispenses) a pool-file record address, and optionally attaches a storage block.
RELFC Releases a pool file address, and optionally detaches the storage block.

The following monitor-request macros request functions related to file addresses:

FACEC (application program codes ENTRC FACE)
Calculates a fixed-file address from the fixed-record type number and record ordinal.
FAC8C Calculates an 8-byte fixed-file address in 4x4 format from the fixed-file record type name or number and record ordinal.

FA4X4C Converts a 4-byte file address to an 8-byte file address in 4x4 format, or an 8-byte file address in 4x4 format to a 4-byte file address.
GDSNC Opens or closes general data set.
GDSRC Gets a general data set file address.
HLDTC Check if a file address at a data level is held by a specified ECB.
RAISA Calculates a general file address for the first record of a general file; or increments the general file address by record ordinal increment.
RIDIC Extracts information about a record ID.

Note: Although ALCS provides the RIDIC macro, there is no equivalent C language function.

RONIC Extracts information about a fixed, pool or general file address; or calculates the file address from the record type symbol and record ordinal.
UNFRC Unhold a held DASD record.

D.3 ALCS services for sequential file processing

Application programs use monitor-request macros to request sequential file operations.

ALCS Application Programming Reference – Assembler describes the sequential file monitor-request macros.

There are different monitor-request macros for general sequential files and for real-time sequential files.

The following monitor-request macros control ownership of a general sequential file:

TASNC Assigns a general sequential file to the entry. The general sequential file data set must be allocated and open. If the general sequential file is assigned to another entry, TASNC waits until the other entry unassigns it (TRSVC); TASNC then assigns it to the entry. If the general sequential file is unassigned (reserved), then TASNC immediately assigns it to the entry.
TCLSC Closes and deallocates a general sequential file data set and unassigns the general sequential file from the entry.
TOPNC Allocates and opens a general sequential file data set, then assigns the general sequential file to the entry.
TRSVC Unassigns (reserves) a general sequential file from the entry, but does not close or deallocate the general sequential file data set. This allows another entry to assign the general sequential file using TASNC.

The following monitor-request macros read and write general sequential file records:

TPRDC Reads a standard size (L1, L2, ...) record from a general sequential file.
TWRTC Writes a standard size (L1, L2, ...) record to a general sequential file.
TDTAC Reads or writes any size record from or to a general sequential file.

The following monitor-request macros write real-time sequential file records:

TOURC Writes a standard size (L1, L2, ...) record to a real-time sequential file.
TOUTC Writes any size record to a real-time sequential file.

Another sequential file monitor-request macro is:

TDSPC Extracts information about real-time, general, and system sequential files.

D.3.1 ALCS C language functions for sequential file processing

The assembler macros in D.3, “ALCS services for sequential file processing” on page 147 are all available in ALCS as equivalent C language functions which can be used for sequential file processing.

ALCS also provides these C language functions, for compatibility with TPF:

tape_open Open a general sequential file.
tape_close Close a general sequential file.
tape_read Read a record.
tape_write Write a record.

All these C language functions are fully described in *ALCS Application Programming Reference – C Language*.

D.4 ALCS entry management services

The following entry-related monitor services are available for application programmers and are described in *ALCS Application Programming Reference – Assembler*.

CORHC Define and hold a resource.
CORUC Unhold a resource.
CREDC Create a new entry and put it into the defer list.
CREEC Create a new entry with attached storage block.
CREMC Create a new entry and put it into the ready list.
CRETC Create a new entry for scheduling after a time delay.
CREXC Has identical function to CREDC.
DEFRC Defer processing of this entry to allow other entries to proceed. Entries request DEFRC monitor service to avoid monopolizing resources.
DEQC Dequeue (unhold) a resource.
DLAYC Has identical function to DEFRC.
ENQC Define and enqueue (hold) a resource.
EVNTC Define an event for EVNWC and POSTC.
EVNWC Wait until an event completes.
LODIC Calculates the number of additional entries you can create without performance degradation.
POSTC Signal that an event is complete. Some events can require more than one POSTC to complete them.
SAVEC Save and restore the contents of requested information (for example registers, ECB work areas) in a local save stack.
SYNCC Synchronize access to application global area.
TASTC Start TAS (see “Time available supervisor” on page 100). TASBC stops TAS.
WAITC Wait for I/O to complete (see 6.4, “Input/output counter and wait service” on page 104). Some other monitor services include a request for the wait service; for example, FINWC requests a DASD record read, followed by a wait.

TPF compatibility

Do not use the SAVEC service in programs that must be compatible with TPF.

D.4.1 C language functions for entry management

Application programs written in the C language can use these entry-related functions provided with ALCS. They are described in *ALCS Application Programming Reference – C Language*.

corhc Define and hold a resource.
coruc Unhold a resource.
credc Create a new entry and put it into the defer list.
creec Create a new entry and pass storage block contents to the new entry.
cremc Create a new entry and put it into the ready list.
cretc Create a new entry after a time delay.
crexc Has identical function to credc.
defrc Suspend processing of this entry to allow other entries to proceed. Entries request defrc monitor service to avoid monopolizing resources.
dlayc Has function identical to defrc.
lodic Calculates the number of additional entries you can create without performance degradation
waitc Wait for I/O to complete. Some other functions include a request for the wait service; for example, finwc requests a DASD record read, followed by a wait (see 6.4, “Input/output counter and wait service” on page 104).

D.5 ALCS storage management services

Application programs use monitor services to request services related to storage management.

ALCS Application Programming Reference – Assembler describes the storage related monitor services. They are:

ALASC Get (obtain and attach) a storage block of specified size as the automatic storage block.
ATTAC Re-attach a storage block after a DETAC.
DECBC Create, locate, validate or release a DECB, or swap the storage level in a DECB with a storage level in the ECB.
DETAC Detach the storage block at a specified level and save the block address.
FLIPC Flip (exchange) the contents of two ECB storage levels. FLIPC also exchanges the contents of ECB data levels and detail error indicators.
GETCC Get (obtain and attach) a storage block of a specified size at a specified level.
RELCC Release (detach and free) a storage block attached at a specified level.
CRUSA Conditionally release (detach and free) storage blocks attached at specified levels.
SAVEC Save and restore the contents of requested information (for example registers, ECB work areas) in a local save stack.
CALOC Reserve and initialize a storage area.
FREEC Release a storage area.
MALOC Reserve a storage area.
RALOC Change size of reserved storage area.

and the following macro that generates inline code:

LEVTA Test if a storage block is attached at a specified level.

TPF compatibility

Do not use the SAVEC service in programs that must be compatible with TPF

Note: In addition to the above, many ALCS monitor services either obtain and attach or detach and free a storage block as part of the requested function. For example, the FINDC (read a DASD record) monitor service automatically obtains and attaches a storage block.

D.5.1 C language functions for storage management

Application programs written in the C language can use functions provided with ALCS for storage management. *ALCS Application Programming Reference – C Language* describes these functions. They are:

attac	Re-attach a storage block after a detac.
detac	Detach the storage block at specified level and save the block address.
flipc	Flip (exchange) the contents of two ECB storage levels. flipc also exchanges the contents of ECB data levels and detail error indicators.
getcc	Get (obtain and attach) a storage block of specified size at a specified level.
relcc	Release (detach and free) a storage block attached at a specified level.
crusa	Conditionally release (detach and free) storage blocks attached at specified levels.
tpf_decb_create	Create a DECB.
tpf_decb_locate	Locate a DECB.
tpf_decb_release	Release a DECB.
tpf_decb_swapblk	Swap the contents of an ECB storage level and a DECB storage level.
tpf_decb_validate	Validate a DECB.

There are also functions for generating inline code:

levtest	Tests if a storage block is attached to a specified storage level
malloc	Standard function to get heap storage
calloc	Standard function to get heap storage
free	Standard function to release heap storage.

Note: Any C variables may use stack space.

D.6 ALCS services for global area processing

ALCS supports macros that provide services related to the application global area.

ALCS Application Programming Reference – Assembler describes the following global-related monitor-request macros.

FILKW Reverses the effect of GLMOD and keypoints a global record.

GLOUC Writes keypointable records from the application global area to the database.
KEYUC Writes keypointable records from the application global area to the database.
SYNCC Synchronizes access to the application global area.

and the following macro that is not a monitor-request macro:

GLOBZ Get the address of a directory.

TPF compatibility

When ALCS global area protection is specified, ALCS also supports:

KEYCC **and** GLMOD To change the PSW protect key to allow storing into global areas 1 or 3
KEYRC **and** FILKW **(option R)** To restore the PSW protect key after storing into global areas 1 or 3

If global area protection is not specified, KEYCC, GLMOD, KEYRC, FILKW (option R) have no effect, but show up in a macro trace.

D.6.1 C language functions for global area processing

The following C language functions are provided with ALCS for global area processing:

glob Addresses an application global field or record.
global Addresses, updates, synchronizes or keypoints a global field.

These functions are fully described in *ALCS Application Programming Reference – C Language*.

D.7 ALCS services for program linkage

Application programs use monitor-request macros to transfer control to other application programs. They are called **program linkage** or **ENTER/BACK** macros.

See *ALCS Application Programming Reference – Assembler* for descriptions of the program linkage monitor-request macros listed below:

BACKC Returns control to the calling application program; that is, to the instruction following a previous ENTRC macro.
ENTDC Transfers control to an application program or transfer vector, and clears (drops) any return addresses that previous ENTRC macros (if any) saved. Use this macro when control does not return to this program.
ENTNC Transfers control to an application program or transfer vector but does not save a return address. Use this macro when control does not return to this program. If this program was entered using ENTRC, then a return address was saved; if a BACKC macro is subsequently issued, then control returns to that address (the instruction following the ENTRC).
ENTRC Transfers control to an application program or transfer vector, and saves the return address. ENTRC saves this return address information in the ECB prefix; Use this macro when control will return to this program (see BACKC).
FINPC Gets the address of an application program.

`FIPWC` Finds an application program, moves it into a storage block, and attaches the storage block to the ECB.

D.7.1 C language functions for program linkage

See *ALCS Application Programming Reference – C Language* for full descriptions of the corresponding C language functions:

`entdc` Transfers control to an application program without return.
`entrc` Transfers control to an application program with an expected return.

Appendix E. Direct-access files

This section describes additional aspects of the ALCS files.

E.1 How ALCS uses the duplicated database

Database duplication is transparent to ECB-controlled application programs. When an application program requests a write, ALCS automatically updates both copies of the database record.

When an application program requests a read, ALCS reads the copy of the record from the data set with the lowest number of outstanding ALCS DASD I/O operations. ALCS reads the other copy of the record only if the read of the first copy results in a permanent I/O error.

When ALCS is executing with a duplicated database, the operator can deallocate one copy of a database data set, using the `VARY OFFLINE` function of the `ZDASD` command. The operator can do this, for example, before powering off a DASD to allow repairs.

E.1.1 I/O errors

If there are two copies of a real-time database data set, then ALCS automatically deallocates one copy if it has one I/O error on a write, or five consecutive I/O errors on reads.

If there is only one copy, ALCS deallocates it after 10 consecutive I/O errors. (This is a catastrophic error, since ALCS cannot continue without at least one copy of every data set.)

For general files and general data sets, ALCS deallocates after 10 consecutive I/O errors. This is not a catastrophic error, since ALCS itself does not need general files and general data sets, but applications that are attempting to use the data will probably fail.

Recoup is an exception in that Recoup will continue to run even if the Recoup general file becomes unusable.

E.1.2 ALCS action when one copy is offline

When the operator or ALCS deallocates one copy of a data set, the copy becomes offline to ALCS. ALCS stops accessing the offline copy and continues to run using only the other (online) copy.

Because ALCS continues to update the online copy, the offline copy becomes out of date. Before ALCS can use the offline copy again, it must update the offline copy so that it becomes identical to the online copy. The `VARY ONLINE` function of the `ZDASD` command reallocates the data set to ALCS, updates it, and makes it online to ALCS.

When the operator or ALCS makes a copy of a data set offline, ALCS marks that copy as unusable. Even when the ALCS job terminates, a subsequent ALCS job does not use the offline copy until the operator requests the `VARY ONLINE` function of

the ZDASD command. ALCS uses the first record of the first size L3 data set to record which data sets are online and which are offline. (Application programs cannot access this record.)

Because there are two copies of this record, each copy contains the time of the last update. At restart, ALCS uses the more recent copy of the record.

For a description of the ZDASD command, see the *ALCS Operation and Maintenance*.

E.2 Update logging

When an ALCS application program issues a FILE monitor-request macro, ALCS can log the update; that is, write a copy of the contents of the new and old records to a system sequential file, (the ALCS update log file). ALCS can also log updates from the online monitor and updates that application programs request indirectly; for example, when the online monitor writes out a keypointable global record.

This update logging, together with suitable backups, makes it possible to restore the real-time database to its status at any specific time. It also allows backup jobs to run while ALCS is updating the database. This is because updates that occur while a backup is running, and after the backup completes, are on the ALCS update log file. Each record on the ALCS update log file contains the update time (TOD clock).

The real-time database should be restored from a suitable backup. To do this with maximum efficiency, it is important to establish schedules and procedures for regular backups.

After the restore, load the updates from the ALCS update log file. To recover from damage that a program error causes, load updates up to just before the time of the damage.

To disable the logging function, define the ALCS update log file as a dummy data set.

E.2.1 Logging criteria

ALCS logging runs all the time that the online monitor is executing. There is no command to stop or start logging. However ALCS does not log all updates.

Some ALCS commands perform updates that need not be logged. For example, the ZDATA command can load records from a sequential file to the real-time database. If these records are lost or damaged, the ZDATA command can load them again. The ZRSTR command also restores records from the ALCS update log. If these records are lost or damaged, the ZRSTR command can load them again.

These ALCS commands use the FLNPC monitor-request macro. ALCS does not log updates by FLNPC. ALCS application programs do not normally need to use FLNPC. If they do, and if the update requires logging, the application program must issue another FILE monitor-request macro (for example, FILEC) following the FLNPC.

For other FILE monitor-request macros and C language functions, ALCS by default logs all updates, except general file and general data set records, and records with

the delayed file VFA option. For example, a short-term pool file has the delayed file option.

Provided that a forward log is defined the new record contents are logged as described above. When a backward log is defined, the old record contents are logged *if* the VFA process options for the record specify update mode.

Overriding default logging criteria

If the default logging is not regarded as satisfactory for your installation, you can write an installation-wide exit routine to override the default. This installation-wide exit can check, for example:

- Record class (general file, fixed, or pool-file record)
- Record type
- Record ordinal
- Record ID
- Record contents

The installation-wide exit can then request one of the following:

- Log the update.
- Do not log the update.
- Log or do not log according to the default

ALCS Installation and Customization describes the USRLOG installation-wide exit.

After some types of hardware failure, or program errors that cause extensive data damage, the ALCS update log file is the only way to recover the ALCS database. Therefore, it is important to use correct criteria to decide which updates to log. The criteria that ALCS uses by default are designed to be *safe* for many applications. Before changing them, consider the following:

- The data that some records contain can be valuable.
- Application programs can fail, if the data they use is invalid. This can happen even with data that is not intrinsically valuable. It can be difficult to recover from loss or damage of this type of data unless ALCS logs the updates.
- Avoid unnecessary logging. Increasing the number of updates that ALCS logs, increases the overhead on ALCS and increases the time it takes to restore the updates.

Note: ALCS logs a keypointable global record if you specify `LOGGING=YES` in the corresponding `GO1GO` macro in the global load control program. You cannot override this in the USRLOG installation-wide exit.

E.3 Record hold facility

If two entries try to update the same DASD record at the same time, the result can be unpredictable. Application programs can avoid these errors by using the **record hold** facility. It is the responsibility of the application programmer to use record hold correctly. If the application does not use it, or uses it incorrectly, ALCS does not necessarily detect or prevent the resulting data loss.

To use record hold, the application program reads the record with a `FINHC` or `FIWHC` monitor-request macro. When either of these macros completes, the entry is holding the record. The entry continues to hold the record until it unholds it with a

FILUC or UNFRC monitor-request macro. When an entry that is holding one or more records exits, ALCS takes a system error dump and unholds the record or records.

Application programs can also use the equivalent C language functions. These are described in:

ALCS Application Programming Reference – Assembler
ALCS Application Programming Reference – C Language

If a second entry tries to hold a record (FINHC or FIWHC) while the first entry is holding the record, the second entry loses control until the first entry unholds the record. In this way, application programs can prevent simultaneous updates to the same record, provided that all programs that update the record use the record hold facility.

Record hold does not prevent another entry from reading the record, unless the other entry also uses record hold.

E.3.1 When record hold is unnecessary

It is not necessary for an application program to hold every record that it updates. For example, a fixed file record can contain the file address of a chain of pool-file records. That is, the fixed record contains the file address of a pool file record (the first in chain). The pool record contains the file address of a second pool record (the second in chain), and so on.

Application programs can safely update any (or all) of the pool records provided that they always hold the fixed record, while they update the pool records. In this way, application programs can reduce the number of records that the entry holds at one time. This improves the performance of the application.

E.3.2 Performance considerations

Incautious use of record hold can cause severe performance degradation, deadlocks, and other problems. Application programs should be designed so that they:

Avoid holding any frequently-used record
If this is impossible, design the application so that entries hold the record for as short a time as possible.

Avoid holding any record for a long time
In general, avoid holding a record for longer than a small proportion of the required transaction response time. If a transaction loses control while another entry is holding a record, its response time increases by the length of time that the other entry holds the record. If a third transaction must wait, its response time increases by twice as much. In this way, record hold can severely degrade transaction response times.

Avoid holding more than one record at one time
Avoid designs that require entries to hold other resources at the same time as records. If you cannot avoid this, take care to ensure that deadlocks cannot occur.

E.3.3 Data sets

The ALCS real-time database and general files are either:

- Single extent VSAM entry-sequenced data sets (ESDSs)
- Relative-record data sets (RRDSs)

Use access method services (AMS) to create these as VSAM clusters in the normal way. You can create ALCS realtime and general files on any DASD device that the MVS Data Facility Product (DFP) supports.

Note: The ALCS database generation listing gives the appropriate AMS commands.

The VSAM cluster is accessed through an integrated catalog facility (ICF) catalog. The VSAM cluster consists of:

- A cluster component (which is simply a catalog entry)
- A data component (the data set that ALCS uses, together with the corresponding catalog entry)

As the ALCS VSAM clusters are not indexed (though you can create indexes if you wish), there is normally no index component. In this book, **data set** refers to the data component of the VSAM cluster.

Each data set contains records of one size only, one record to every VSAM control interval. The VSAM control interval size is the same as the ALCS storage block size. The VSAM user record length is 8 bytes less than the control interval size.

Note: The VSAM user record length is not the same as the ALCS generation macro user record length. In the ALCS macro you define:

- The CISIZE, the physical record size for a record (its VSAM control interval)
- The RECSIZE, the corresponding logical record size

There must be a difference of at least 56 bytes between the two. Figure 100 summarizes the relationship between CISIZE and RECSIZE.

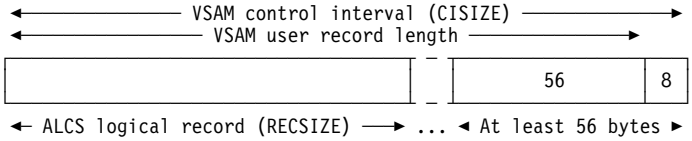


Figure 100. ALCS record sizes and VSAM control intervals

For example an L1 record size defined with a RECSIZE of 381 bytes has a CISIZE of 512 (the nearest VSAM control interval to that size). The *ALCS Installation and Customization* describes the RECSIZE and CISIZE values used in the ALCS macro.

Real-time database data set names

ALCS real-time data base dataset names (for the data component) have the following format:

prefix.**L***n*.**C***n*.*issssss*

Where:

prefix
Prefix specified by the DSNAME= parameter of the DBGEN macro.

L*n* Record size (L1, L2, and so on).

C*n* Copy (C1 for copy 1 or C2 for copy 2).

issssss
ALCS system ID and data set sequence number, where:

i ALCS system ID specified by the ID= parameter of the ALCS macro

ssssss Data set sequence number, 0000001 for the first data set, 0000002 for the second data set, and so on.

E.3.4 Allocating data sets

ALCS allocates the cluster component of the VSAM cluster in the same way as other MVS utilities.

Application programs that do not run under the control of the ALCS monitor can use standard VSAM facilities with relative byte addressing to access the real-time database and general file and general data sets. The relative byte address (RBA) is the CISIZE multiplied by the relative record number. Figure 101 demonstrates the relationship between CISIZE and the RBA.

Figure 101. Relationship between CISIZE and RBA

Record	Relative record number	RBA
1	0	0
2	1	CISIZE
3	2	2 X CISIZE

In ALCS DASD generation, for each record type, specify either the record ID or a symbol for the record ID (IDSYM). The ID= and IDSYM= parameters of the `USRDTA` macro are described in the *ALCS Installation and Customization*.

E.4 Offline access to file address information

Offline utility application programs (that is, programs that run directly under the control of MVS and not under ALCS) may need information about ALCS file addresses. ALCS provides three utility routines in CSECT DXCFARO, which act as entry points enabling application programs to obtain this information. These routines are as follows:

DXCCDDLD Load DASD configuration table load module.

DXCMRT Return the highest used general file number, pool interval number, or fixed-record type number.

DXCFARIC Return information about a given file address.

To use these routines, use one of the following procedures:

- Invoke them dynamically, using MVS `LINK`. If you use `LINK`, ensure that the PDS containing the load module DXCFAROL is included in the JOBLIB or STEPLIB concatenation at execution time. See the ALCS *Program Directory* for information on the location of this module.
- Use `CALL`, and link-edit the load module DXCFAROL with your application program.

Note: You are strongly recommended to use `LINK` rather than `CALL`. Using `LINK` ensures access to the latest versions of these routines. If you use `CALL`, you will have to relink your application programs manually to incorporate IBM-supplied maintenance.

The ALCS DASD configuration table load module is the source of all file address information, so you must either load this module (using routine DXCCDDLD), or include the DASD configuration load module in the link-edit of the application program.

See *ALCS Application Programming Reference – Assembler* for further information.

Appendix F. Application global area

There is an area of storage in the ALCS MVS address space that all entries can access. It is called the **application global area** or the **global area**. The application global area contains:

- Global area records
- Global area directories.

F.1 Global area records

These are selected records from the real-time database. The records can be keypointable or non-keypointable, as follows:

- **Keypointable** Records that application programs update. After an update, ALCS writes the record to the database. The process of writing a record to the database is called **keypointing**.
- **Non-keypointable** Records that application programs do not update, or that are reinitialized when ALCS restarts. ALCS does not write the record to the database.

F.2 Global area directories

The application global area contains up to 16 global area directories. Each directory can contain up to 255 directory slots. Application programs use these directories to find global area records.

Each global area directory can be followed by **directly addressable records** (within a 4KB block). These records can be addressed using an offset. Figure 102 shows a logical view of a global area directory and the two types of global record.

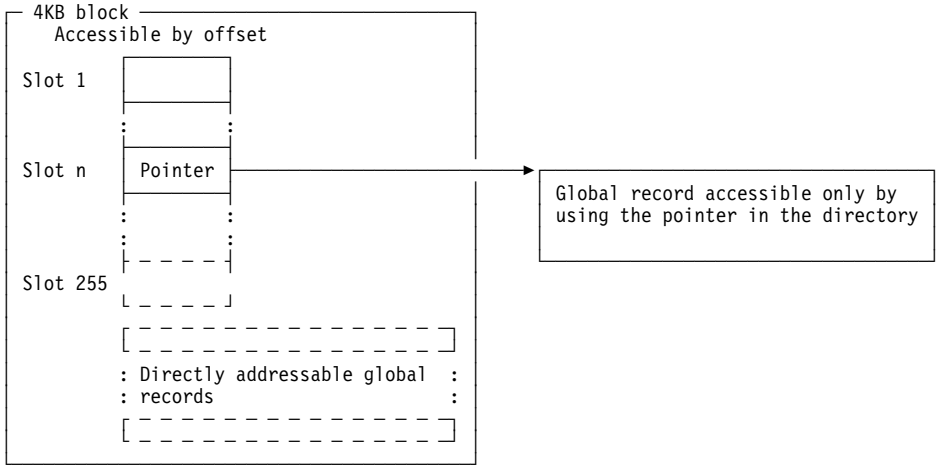


Figure 102. ALCS global areas and global area directory – logical view

Application programs use the global area directory to find global records. To do this, application programs use the ALCS `GLOBZ` macro which generates DSECTs for the 16 global area directories and for the directly addressable global areas. `GLOBZ` also generates instructions that load a base register. This base register allows the application program to address one directory and the directly addressable areas.

For a description of how to use the GL0BZ macro, see *ALCS Application Programming Reference – Assembler*.

The GL0BZ macrodefinition does not contain the DSECTs for the directories and the directly addressable areas. Instead, GL0BZ calls DSECT macros that include the DSECTs. Figure 103 shows how all the global area directories and associated records are physically organized in storage.

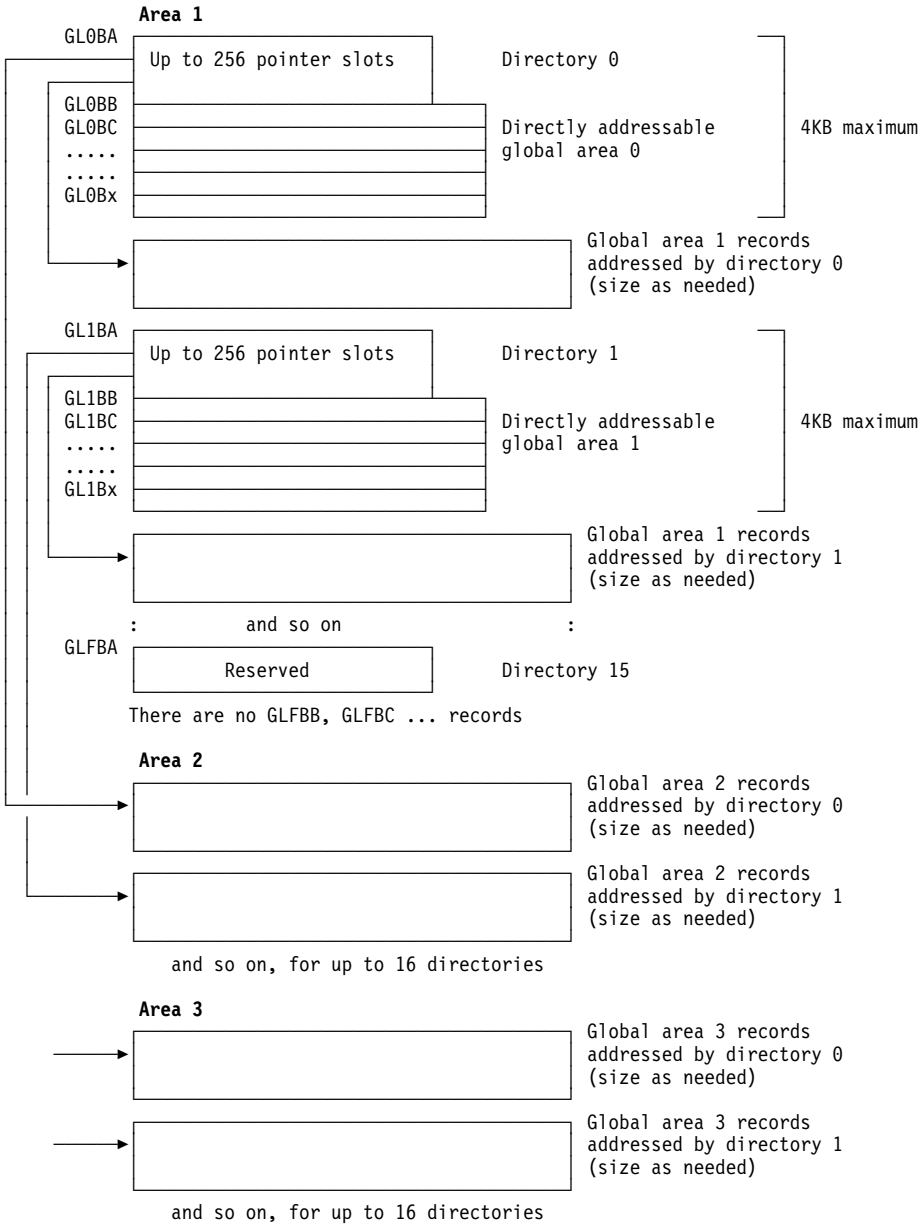


Figure 103. ALCS global area – physical view

Notes:

1. There are three global areas, the global area directories must reside in area 1 and can point to Area 1, Area 2, and Area 3.
2. You can move Areas 1 and 2 above the 16MB boundary by means of the AMODE31= parameter of the SCTGEN macro.
3. There is usually a directory slot for each record in the global area. The slot contains the storage address of the record and (optionally) the file address of the record.
4. Directory F and the first slot in directory 0 is reserved; all other slots are available for application use and can point to keypointable records.

TPF compatibility

In TPF and ALCS, when global area protection is specified, global area 1 and 3 – the "protected global areas" – have a different key from entry storage.

DSECTs for the 16 directories are in macros GL0BA through GLFBA. The global load control programs are GOA0 through GOAE, and CGAF.

F.3 Header stripping and logical globals

ALCS DASD records include a standard header. This header contains information such as the record ID, record code check (RCC), and so on. Multiple physical records can be loaded into the global area as one logical record. To do this, the ALCS global load program loads multiple records from the real-time database into contiguous areas of storage, and strips the header from all but the first record. This can simplify an application program's handling of the information within the records, but it complicates keypointing. ALCS can keypoint header-stripped records; this is called **logical global support**.

You should create the records that comprise a logical global in the usual way. For example, use STC to create the records on a data file, and the ZDATA command to load the records from the data file on to the database.

F.4 Including records in the application global area

The ALCS application global area can contain records from the real-time database. Because the records are permanently in storage, application programs can access the data that they contain with the minimum overhead. In particular, application programs can read data in the application global area without the need for any I/O.

However, it is better to use VFA attributes to reduce the number of I/Os. In particular, the permanently resident attribute and the time-initiated file attribute can allow application programs to use FIND and FILE monitor-request macros with relatively few I/Os.

Using the application global area for records for the real-time database has the following disadvantages:

- Applications programs that share access to records in the global area must serialize access to the records. This process can be complex.

- It is difficult to add existing records to the global area. Existing application programs that use FIND and FILE macros to access a record must be changed to access the record in the global area. Similarly, it is difficult to remove records from the global area. Existing application programs that access a record in the global area must be changed to use FIND and FILE macros.

  By contrast, it is easy to change the VFA options for a record. There is no need to change programs that access the record.
- It may become necessary at some stage to run the application programs on a loosely-coupled system (such as TPF). Application programs that use the global area can require major changes to work in a loosely coupled environment.
- The use of globals greatly reduces the portability of a program.

When there is no alternative to using the global area, add a record to the global area, as follows:

1. Choose which global area you will use for the record. There are three global areas, 1, 2 and 3. If your installation is using global area protection (refer to the GLBLPROT parameter on the ALCS generation SCTGEN macro) global areas 1 and 3 will be in protected storage, therefore the global area that you choose for your record must be determined by your requirement to place the record in either protected or unprotected storage.
2. Update the corresponding global area directory DSECT macro to include a directory slot for the record.
3. If your installation is using C language programs that access the global areas, you must recreate `<c$globz.h>`.
4. Step 2 can change the value of symbols (that is, the displacement to labels in the DSECT) that existing application programs use. If it does, reassemble/recompile and link-edit the affected programs.
5. If the new record is directly addressable, update the corresponding directory macro (GL*n*BA) to include the new record in the GL*n*BA DSECT.
6. To load the new record into the global area, update the global load control program for the directory.
7. Update the GLBLSZE= parameter in the ALCS generation SCTGEN macro.

   Refer to the SCTGEN macro in *ALCS Installation and Customization* for a description of the GLBLSZE= parameter and how to determine the size of the global area.
8. Reassemble and relink program AGT1.

Appendix G. Application program management

This chapter documents General-Use Programming Interface information provided by ALCS. General-Use Programming Interfaces allow the user to write programs that obtain the services of ALCS.

ALCS supports application programs written in IBM System/390 assembler language[3] using macrodefinitions supplied with ALCS. You can also write application programs in the C language. Application programs can use SQL to access relational databases. They can also use CPI-C, MQI, TCP/IP, and APPC/MVS calls to communicate with other application programs (which need not be running under ALCS).

Assembler application programs must adhere to the restrictions and requirements described in *ALCS Application Programming Reference – Assembler*. For example:

- Each program and transfer vector (entry point) must have a unique 4-character name.
- The maximum size for a program is 32KB.
- All programs must be reentrant.
- The first instruction of each program must be the `BEGIN` macro. `BEGIN` generates a 32-byte program header. This header includes information such as the program name, the program version number, the program length in bytes, and the count of transfer vectors within the program.
- `TRANV` macros to define multiple entry points in a program immediately follow the `BEGIN` macro. A `TRANV` macro generates 8 bytes of transfer vector definition.

The rules that C application programs must adhere to are described in *ALCS Application Programming Reference – C Language*.

At restart, ALCS loads each load module in the application program load list (see G.1, “The program configuration table” on page 166). ALCS searches each load module for ALCS application programs; the load modules must contain only ALCS application programs. ALCS recognizes ALCS application programs by checking the 32-byte program header. If the header is valid, ALCS uses it to build program table entries for the program and for the transfer vectors (if any).

When either an application program or the ALCS online monitor transfers control to an application program, the calling program refers to the called program using the 4-character program name. ALCS program management uses the program table to determine the storage address of the called program.

You must ensure that application programs which run in 24-bit addressing mode are loaded below 16MB. Do this by link-editing 24-bit mode programs into one or more load modules with RMODE (residence mode) 24. Load modules containing only programs which can run in 31-bit addressing mode should be link-edited with AMODE (addressing mode) 31 and RMODE ANY.

---

3 This includes programs written in subsets of IBM System/390 assembler language – for example, System/360, System/370, and so on.

If the called program is a transfer vector, ALCS passes control at the relevant entry point.

G.1 The program configuration table

The program configuration table contains all the information that ALCS needs to load and manage application programs. *ALCS Installation and Customization* describes how to update the program configuration table. This contains general information, such as the maximum number of load modules and programs expected. It also contains the **application program load list**.

G.2 Application program load list

This is a list of load modules for ALCS to load at restart. During restart, ALCS loads modules in the sequence that they appear in this list. If a program appears in more than one module, ALCS creates multiple entries in the program table. The copy that ALCS loads last becomes effective. ALCS automatically loads first the modules that contain ECB-controlled monitor programs.

G.3 Naming application programs

Give each new program and transfer vector a unique 4 character name. Each name must start with an alphabetic character, but names beginning with A, B, and C are reserved for IBM programs as follows:

A Installation-wide ECB-controlled exit programs
B Data programs
C ALCS ECB-controlled monitor programs

TPFDF compatibility

To avoid conflict with TPFDF, avoid using program names beginning with UF.

The following names are also reserved:

FACE
FACS
GOAn
RLCH
TIA1
UGU1
XHP1

Optionally, a program can also have a version number of 2-alphanumeric characters. A version number can be useful for identifying the program version in a dump. ALCS program management does not use the version number.

Appendix H. Acronyms and abbreviations

The following acronyms and abbreviations are used in books of the ALCS Version 2 library. Not all are necessarily present in this book.

AAA	agent assembly area
ACB	VTAM access method control block
ACF	Advanced Communications Function
ACF/NCP	Advanced Communications Function for the Network Control Program, usually referred to simply as “NCP”
ACF/VTAM*	Advanced Communications Function for the Virtual Telecommunication Access Method, usually referred to simply as “VTAM”
ACK	positive acknowledgment (SLC LCB)
ACP	Airline Control Program
AID	IBM 3270 attention identifier
AIX	add item index
ALC	airlines line control
ALCI	Airlines Line Control Interconnection
ALCS/MVS/XA	Airline Control System/MVS/XA
ALCS/VSE	Airline Control System/Virtual Storage Extended
ALCS V2	Airline Control System Version 2
AML	acknowledge message label (SLC LCB)
AMS	access method services
AMSG	AMSG application message format
APAR	authorized program analysis report
APF	authorized program facility
API	application program interface
APPC	advanced program-to-program communications
ARINC**	Aeronautical Radio Incorporated
ASCU	agent set control unit (SITA), a synonym for “terminal control unit”
AT&T**	American Telephone and Telegraph Co.
ATA	Air Transport Association of America
ATSN	acknowledge transmission sequence number (SLC)
BATAP	Type B application-to-application program
BSC	binary synchronous communication
C	C programming language
CAF	DB2 Call Attach Facility
CCW	channel command word
CDPI	clearly differentiated programming interface
CEC	central electronic complex
CEUS	communication end-user system
CI	VSAM control interval
CICS*	Customer Information Control System
CLIST	command list
CMC	communication management configuration
CML	clear message label (synonym for AML)
COBOL	COmmon Business Oriented Language
CPI-C	Common Programming Interface – Communications
CPU	central processing unit
CRAS	computer room agent set
CRI	communication resource identifier

CRN	communication resource name
CSA	common service area
CSECT	control section
CSID	cross system identifier
CSW	channel status word
CTKB	Keypoint record B
CTL	control system error
CUA*	Common User Access
DASD	direct access storage device
DBCS	double-byte character set
DBRM	DB2 database request module
DB2*	IBM DB2 for z/OS
DCB	data set control block
DECB	ALCS data event control block
DF	delayed file record
DFDSS	Data Facility Data Set Services
DFHSM	Data Facility Hierarchical Storage Manager
DFP	Data Facility Product
DFSMS*	Data Facility Storage Management Subsystem
DFT	distributed function terminal
DIX	delete item index
DRIL	data record information library
DSI	direct subsystem interface
DSECT	dummy control section
DTP	ALCS diagnostic file processor
EBCDIC	extended binary-coded decimal interchange code
ECB	ALCS entry control block
EIB	error index byte
EID	event identifier
ENQ	enquiry (SLC LCB)
EOF	end of file
EOM	end of message
EOI	end of message incomplete
EOP	end of message pushbutton
EOU	end of message unsolicited
EP	Emulation Program
EP/VS	Emulation Program/VS
EVCB	MVS event control block
EXCP	Execute Channel Program
FACE	file address compute
FIFO	first-in-first-out
FI	file immediate record
FM	function management
FMH	function management header
GB	gigabyte (1 073 741 824 bytes)
GDS	general data set
GFS	get file storage (called pool file storage in ALCS)
GMT	Greenwich Mean Time
GTF	generalized trace facility (MVS)
GUPI	general-use programming interface
HEN	high-level network entry address
HEX	high-level network exit address
HFS	Hierarchical File System
HLASM	High Level Assembler

HLL	high-level language
HLN	high-level network
HLS	high-level system (for example, SITA)
IA	interchange address
IASC	International Air Transport Solution Centre
IATA	International Air Transport Association
IATA5	ATA/IATA transmission code 5
IATA7	ATA/IATA transmission code 7
ICF	integrated catalog facility
ID	identifier
ILB	idle (SLC LCB)
IMA	BATAP acknowledgement
IMS*	Information Management System
IMSG	IMSG input message format
I/O	input/output
IOCB	I/O control block
IP	Internet Protocol
IPARS	International Programmed Airlines Reservation System
IPCS	Interactive Problem Control System
IPL	initial program load
ISA	initial storage allocation
ISC	intersystem communication
ISO/ANSI	International Standards Organization/American National Standards Institute
ISPF	Interactive System Productivity Facility
ISPF/PDF	Interactive System Productivity Facility/Program Development Facility
ITA2	International Telegraph Alphabet number 2
JCL	job control language
JES	job entry subsystem
KB	kilobyte (1024 bytes)
KCN	link channel number (SLC)
KSDS	VSAM key-sequenced data set
LAN	local area network
LCB	link control block (SLC)
LDB	link data block (SLC)
LDI	local DXCREI index
LEID	logical end-point identifier
LE	Language Environment*
LICRA	Link Control – Airline
LMT	long message transmitter
LN	line number (ALCS/VSE and TPF terminology)
LN/ARID	line number and adjusted resource identifier (ALCS/VSE terminology)
LSI	link status identifier (SLC)
LU	logical unit
LU 6.2	Logical Unit 6.2
MATIP	Mapping of airline traffic over IP
MB	megabyte (1 048 576 bytes)
MBI	message block indicator (SLC)
MCHR	module/cylinder/head/record
MESW	message switching
MNOTE	message note
MQI	Message Queueing Interface

MQM	Message Queue Manager
MSNF	Multisystem Networking Facility
MVS*	Multiple Virtual Storage (refers to both MVS/XA and MVS/ESA, and also to OS/390* and z/OS*)
MVS/DFP*	Multiple Virtual Storage/Data Facility Product
MVS/ESA*	Multiple Virtual Storage/Enterprise System Architecture
MVS/XA*	Multiple Virtual Storage/Extended Architecture
NAB	next available byte
NAK	negative acknowledgment (SLC LCB)
NCB	network control block (SLC)
NCP	Network Control Program (refers to ACF/NCP)
NCP/VS	Network Control Program/Virtual Storage.
NEF	Network Extension Facility
NEF2	Network Extension Facility 2
NPDA	Network Problem Determination Application
NPSI	Network Control Program packet switching interface
NTO	Network Terminal Option
OCR	one component report
OCTM	online communication table maintenance
OMSG	OMSG output message format
OPR	operational system error
OSID	other-system identification
OS/2*	IBM Operating System/2
PARS	Programmed Airlines Reservation System
PDF	parallel data field (refers to NCP)
PDM	possible duplicate message
PDS	partitioned data set
PDSE	partitioned data set extended
PDU	pool directory update
PER	program event recording
PFDR	pool file directory record
PL/I	programming language one
PLM	purge long message (name of ALCS/VSE and TPF general tape)
PLU	primary logical unit
PNL	passenger name list
PNR	passenger name record
PP	IBM program product
PPI	program-to-program interface
PPMSG	program-to-program message format
PPT	program properties table
PR	permanently resident record
PRC	prime computer room agent set
PRDT	physical record (block) descriptor table
PRPQ	programming request for price quotation
PR/SM*	Processor Resource/Systems Manager*
PS	VTAM presentation services
PSPI	product sensitive programming interface
PSW	program status word
PTF	program temporary fix
PTT	Post Telephone and Telegraph Administration
PU	physical unit
PVC	permanent virtual circuit
QSAM	queued sequential access method
RACF*	resource access control facility

RB	request block
RBA	relative byte address
RCC	record code check
RCPL	routing control parameter list
RCR	resource control record
RCS	regional control center
RDB	Relational Database
RDBM	Relational Database Manager
REI	resource entry index
RLT	record locator table
RMF*	Resource Measurement Facility*
RO CRAS	receive-only computer room agent set
RON	record ordinal number
RPL	VTAM request parameter list
RPQ	request for price quotation
RSM	resume (SLC LCB)
RTM	recovery and termination management
RU	request unit
SAA*	Systems Application Architecture*
SAF	System Authorization Facility
SAL	system allocator list (TPF terminology)
SAM	sequential access method
SDLC	Synchronous Data Link Control
SDMF	standard data and message file
SDSF	System Display and Search Facility
SDWA	system diagnostic work area
SI	DBCS shift in
SITA**	Société Internationale de Télécommunications Aéronautiques
SLC	ATA/IATA synchronous link control
SLIP	serviceability level indication processing
SLN	symbolic line number
SLR	Service Level Reporter
SLU	secondary logical unit
SMP/E	System Modification Program Extended
SNA	Systems Network Architecture
SO	DBCS shift out
SON	system ordinal number
SQA	system queue area
SQL	Structured Query Language
SQLCA	SQL Communication Area
SQLDA	SQL Descriptor Area
SRB	service request block
SRG	statistical report generator
SRM	System Resource Manager
STC	system test compiler
STP	stop (SLC LCB)
STV	system test vehicle
SWB	service work block
SYN	character synchronization character
TA	terminal address
TAS	time available supervisor
TCB	task control block
TCID	terminal circuit identity
TCP/IP	Transmission Control Protocol / Internet Protocol

TI	time-initiated record
TOD	time of day
TPF	Transaction Processing Facility
TPF/APPC	Transaction Processing Facility/Advanced Program to Program Communications
TPF/DBR	Transaction Processing Facility/Data Base Reorganization
TPFDF	TPF Database Facility
TPF/MVS	Transaction Processing Facility/MVS (alternative name for ALCS V2)
TP_ID	transaction program identifier
TSI	transmission status indicator
TSN	transmission sequence number
TSO	time-sharing option
TSO/E	Time Sharing Option Extensions
TUT	test unit tape (sequential file)
UCB	unit control block
UCTF	Universal Communications Test Facility
VFA	virtual file access
VIPA	virtual IP address
VM	virtual machine
VM/CMS	virtual machine/conversational monitor system
VS	virtual storage
VSAM	virtual storage access method
VSE	Virtual Storage Extended
VSE/AF	Virtual Storage Extended/Advanced Function
VSE/VSAM	Virtual Storage Extended/Virtual Storage Access Method
VTAM*	Virtual Telecommunications Access Method (refers to VTAM)
VTOC	volume table of contents
WSF	Write Structured Field
WTTY	World Trade Teletypewriter
XMSG	XMSG message switching message format
XREF	ALCS cross referencing facility

Glossary

Notes:

1. Acronyms and abbreviations are listed separately from this Glossary. See Appendix H, “Acronyms and abbreviations” on page 167.
2. For an explanation of any term not defined here, see the IBM *Dictionary of Computing*.

A

AAA hold. See terminal hold.

abnormal end of task (abend). Termination of a task before its completion because of an error condition that cannot be resolved by recovery facilities while the task is executing.

access method services (AMS). A utility program that defines VSAM data sets (or files) and allocates space for them, converts indexed sequential data sets to key-sequenced data sets with indexes, modifies data set attributes in the catalog, facilitates data set portability between operating systems, creates backup copies of data sets and indexes, helps make inaccessible data sets accessible, and lists data set records and catalog entries.

activity control variable. A parameter that ALCS uses to control its workload. The system programmer defines activity control variables in the ALCS system configuration table generation.

Advanced Communications Function for the Network Control Program (ACF/NCP). An IBM licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability.

Advanced Program-to-Program Communications (APPC). A set of inter-program communication services that support cooperative transaction processing in an SNA network. APPC is the implementation, on a given system, of SNA’s logical unit type 6.2 (LU 6.2). See APPC component and APPC transaction scheduler.

Aeronautical Radio Incorporated (ARINC). An organization which provides communication facilities for use within the airline industry.

agent assembly area (AAA). A fixed-file record used by IPARS applications. One AAA record is associated with each terminal and holds data that needs to be kept beyond the life of an entry. For example, to collect information from more than one message.

agent set. Synonym for communication terminal.

agent set control unit (ASCU). Synonym for terminal interchange.

Airline Control Program (ACP). An earlier version of the IBM licensed program Transaction Processing Facility (TPF).

Airline Control System (ALCS). A transaction processing platform providing high performance, capacity, and availability, that runs specialized (typically airline) transaction processing applications.

Airline Control System/Multiple Virtual Storage/Extended Architecture (ALCS/MVS/XA). An ALCS release designed to run under an MVS/XA operating system.

Airline Control System Version 2 (ALCS V2). An ALCS release designed to run under a z/OS operating system.

Airline Control System/Virtual Storage Extended (ALCS/VSE). An ALCS release designed to run under a VSE/AF operating system.

airlines line control (ALC). A communication protocol particularly used by airlines.

Airlines Line Control Interconnection (ALCI). A feature of Network Control Program (NCP) that allows it to manage ALC networks in conjunction with a request for price quotation (RPQ) scanner for the IBM 3745 communication controller.

Airline X.25 (AX.25). A discipline conforming to the ATA/IATA AX.25 specification in the ATA/IATA publication *ATA/IATA Interline Communications Manual*, ATA/IATA document DOC.GEN 1840. AX.25 is based on X.25 and is intended for connecting airline computer systems to SITA or ARINC networks.

ALCS command. A command addressed to the ALCS system. All ALCS commands start with the letter Z (they are also called “Z messages”) and are 5 characters long.

These commands allow the operator to monitor and control ALCS. Many of them can only be entered from CRAS terminals. ALCS commands are called “functional messages” in TPF.

ALCS data collection file. A series of sequential data sets to which ALCS writes performance-related data for subsequent processing by the statistical report

generator or other utility program. See also data collection and statistical report generator.

ALCS diagnostic file. A series of sequential data sets to which the ALCS monitor writes all types of diagnostic data for subsequent processing by the diagnostic file processor.

ALCS diagnostic file processor. An offline utility, often called the “post processor”, that reads the ALCS diagnostic file and formats and prints the dump, trace, and system test vehicle (STV) data that it contains.

ALCS entry dispatcher. The ALCS online monitor's main work scheduler. Often called the “CPU loop”.

ALCS offline program. An ALCS program that runs as a separate MVS job (not under the control of the ALCS online monitor).

ALCS online monitor. The part of ALCS that performs the services for the ECB-controlled programs and controls their actions.

ALCS trace facility. An online facility that monitors the execution of application programs. When it meets a selected monitor-request macro, it interrupts processing and sends selected data to an ALCS display terminal, to the ALCS diagnostic file, or to the system macro trace block. See also instruction step.

The ALCS trace facility also controls tracing to the MVS generalized trace facility (GTF), for selected VTAM communication activity.

ALCS update log file. A series of sequential data sets in which the ALCS monitor records changes to the real-time database.

ALCS user file. A series of sequential data sets to which you may write all types of diagnostic data for subsequent processing by an offline processor. You write the data from an installation-wide monitor exit using the callable service UWSEQ.

allocatable pool. The ALCS record class that includes all records on the real-time database. Within this class, there is one record type for each DASD record size.

The allocatable pool class is special in that ALCS itself can dispense allocatable pool records and use them for other real-time database record classes. For example, all fixed-file records are also allocatable pool records (they have a special status of “in use for fixed file”).

When ALCS is using type 2 long-term pool dispense, ALCS satisfies requests for long-term pool by dispensing available allocatable pool records.

See DASD record, real-time database, record class, and record type.

alternate CRAS. A computer room agent set (CRAS) that is not Prime CRAS or receive only CRAS. See computer room agent set, Prime CRAS, and receive only CRAS.

alternate CRAS printer. A CRAS printer that is not receive only CRAS. See CRAS printer and receive only CRAS.

answerback. A positive acknowledgement (ACK) from an ALC printer.

APPC component. The component of MVS that is responsible for extending LU 6.2 and SAA CPI Communications services to applications running in any MVS address space. Includes APPC conversations and scheduling services.

APPC transaction scheduler. A program such as ALCS that is responsible for scheduling incoming work requests from cooperative transaction programs.

application plan. See DB2 application plan.

application. A group of associated application programs that carry out a specific function.

application global area. An area of storage in the ALCS address space containing application data that any entry can access.

The application global area is subdivided into keypointable and nonkeypointable records. Keypointable records are written to the database after an update; nonkeypointable records either never change, or are reinitialized when ALCS restarts.

C programs refer to global records and global fields within the application global area.

application program. A program that runs under the control of ALCS. See also ECB-controlled program.

application program load module. In ALCS, a load module that contains one or more application programs.

application queue. In message queuing with ALCS, any queue on which application programs put and get messages using MQI calls.

assign. Allocate a general sequential file to an entry. The `TOPNC` monitor-request macro (or equivalent C function) opens and allocates a general sequential file. The `TASNC` monitor-request macro (or equivalent C function) allocates a general sequential file that is already open but not assigned to an entry (it is reserved).

associated resource. Some ALCS commands generate output to a printer (for example, `ZDCOM` prints information about a communication resource). For this type of command the printed output goes to the

associated resource; that is, to a printer associated with the originating display. There is also a response to the originating display that includes information identifying the associated resource.

asynchronous trace. One mode of operation of the ALCS trace facility. Asynchronous trace is a conversational trace facility to interactively trace entries that do not originate from a specific terminal.

automatic storage block. A storage block that is attached to an entry, but is not attached at a storage level. An assembler program can use the ALASC monitor-request macro to obtain an automatic storage block and BACKC monitor-request macro to release it. C programs cannot obtain automatic storage blocks.

B

backward chain. The fourth fullword of a record stored on the ALCS database, part of the record header. See chaining of records.

When standard backward chaining is used, this field contains the file address of the previous record in the chain, except that the first record contains the file address of the last record in the chain. (If there is only one record, the backward chain field contains zeros.)

balanced path. A path where no single component (channel, DASD director or control unit, head of string, and internal path to the DASD device) is utilized beyond the limits appropriate to the required performance.

bar. In the MVS 64-bit address space, a virtual line called the bar marks the 2-gigabyte address. The bar separates storage below the 2-gigabyte address, called **below the bar**, from storage above the 2-gigabyte address, called **above the bar**.

BATAP. Type B application-to-application program

Binary Synchronous Communication (BSC). A form of telecommunication line control that uses a standard set of transmission control characters and control character sequences, for binary synchronous transmission of binary-coded data between stations.

bind. See DB2 bind

BIND. In SNA, a request to activate a session between two logical units (LUs). The BIND request is sent from a primary LU to a secondary LU. The secondary LU uses the BIND parameters to help determine whether it will respond positively or negatively to the BIND request.

binder. The program that replaces the linkage editor and batch loader programs that were provided with earlier versions of MVS.

BIND image. In SNA, the set of fields in a BIND request that contain the session parameters.

block. See storage block.

C

catastrophic. A type of system error that results in the termination of ALCS.

chain-chase. See Recoup.

chaining of records. One record can contain the file address of another (usually a pool-file record). The addressed record is said to be chained from the previous record. Chains of records can contain many pool-file records. See forward chain and backward chain.

class. See record class.

clearly differentiated programming interfaces (CDPI). A set of guidelines for developing and documenting product interfaces so that there is clear differentiation between interfaces intended for general programming use (GUPIs) and those intended for other specialized tasks.

close. Close a sequential file data set (MVS CLOSE macro) and deallocate it from ALCS. For general sequential files this is a function of the TCLSC monitor-request macro (or equivalent C function). ALCS automatically closes other sequential files at end-of-job.

command. See ALCS command.

command list (CLIST). A sequential list of commands, control statements, or both, that is assigned a name. When the name is invoked the commands in the list are executed.

commit. An operation that terminates a unit of recovery. Data that was changed is now consistent.

common entry point (CEP). A function in the Transaction Processing Facility Database Facility (TPFDF) product that provides common processing for all TPFDF macro calls issued by ALCS application programs. It also provides trace facilities for TPFDF macro calls.

Common Programming Interface – Communications (CPI-C). The communication element of IBM Systems Application Architecture (SAA). CPI-C provides a programming interface that allows program-to-program communication using the IBM SNA logical unit 6.2.

Common User Access. Guidelines for the dialog between a user and a workstation or terminal.

communication management configuration (CMC). A technique for configuring a network that allows for the consolidation of many network management functions for the entire network in a single host processor.

communication resource. A communication network component that has been defined to ALCS. These include each terminal on the network and other network components that ALCS controls directly (for example, SLC links). Resources can include, for example:

SNA LUs (including LU 6.1 links)
ALC terminals
SLC and WTTY links
Applications.

communication resource identifier (CRI). A 3-byte field that uniquely identifies an ALCS communication resource. It is equivalent to the LN/IA/TA in TPF and the LN/ARID in ALCS/VSE. ALCS generates a CRI for each resource.

communication resource name (CRN). A 1- to 8-character name that uniquely identifies an ALCS communication resource. For SNA LUs, it is the LU name. The system programmer defines the CRN for each resource in the ALCS communication generation.

communication resource ordinal. A unique number that ALCS associates with each communication resource. An installation can use the communication resource ordinal as a record ordinal for a particular fixed-file record type. This uniquely associates each communication resource with a single record.

For example, IPARS defines a fixed-file record type (#WAARI) for AAA records. Each communication resource has its own AAA record – the #WAARI record ordinal is the communication resource ordinal. See also record ordinal and agent assembly area.

compiler. A program that translates instructions written in a high level programming language into machine language.

computer room agent set (CRAS). An ALCS terminal that is authorized for the entry of restricted ALCS commands.

Prime CRAS is the primary terminal that controls the ALCS system. Receive Only CRAS (RO CRAS) is a designated printer or NetView operator identifier to which certain messages about system function and progress are sent.

configuration data set. (1) A data set that contains configuration data for ALCS. See also configuration-dependent table. (2) The ALCS record class that includes all records on the configuration data set. There is only one record type for this class. See record class and record type.

configuration-dependent table. A table, constructed by the ALCS generation process, which contains configuration-dependent data. Configuration-dependent tables are constructed as conventional MVS load modules. In ALCS V2, there are separate configuration-dependent tables for:

System data
DASD data
Sequential file data
Communication data
Application program data.

See also configuration data set.

control byte. The fourth byte of a record stored on the ALCS database, part of the record header. ALCS ignores this byte; some applications, however, make use of it.

control interval (CI). A fixed-length area of direct access storage in which VSAM stores records. The control interval is the unit of information that VSAM transmits to or from direct access storage.

control transfer. The process that the ALCS online monitor uses to create a new entry and to transfer control to an ECB-controlled program.

conversation_ID:. An 8-byte identifier, used in Get_Conversation calls, that uniquely identifies a conversation. APPC/MVS returns a conversation_ID on the CMINIT, ATBALLOC, and ATBGETC calls; a conversation_ID is required as input on subsequent APPC/MVS calls.

CPU loop. See ALCS entry dispatcher.

CRAS printer. A computer room agent set (CRAS) that is a printer terminal. See computer room agent set.

CRAS display. A computer room agent set (CRAS) that is a display terminal. See computer room agent set.

CRAS fallback. The automatic process that occurs when the Prime CRAS or receive only CRAS becomes unusable by which an alternate CRAS becomes Prime CRAS or receive only CRAS. See also Prime CRAS, receive only CRAS, and alternate CRAS.

create service. An ALCS service that enables an ALCS application program to create new entries for asynchronous processing. The new ECBs compete for system resources and, once created, are not dependent or connected in any way with the creating ECB.

cycling the system. The ALCS system can be run in one of four different system states. Altering the system state is called cycling the system. See SLC link for another use of the term “cycling”.

D

DASD record. A record stored on a direct access storage device (DASD). ALCS allows the same range of sizes for DASD records as it allows for storage blocks, except no size L0 DASD records exist.

data collection. An online function that collects data about selected activity in the system and sends it to the ALCS data collection file, if there is one, or to the ALCS diagnostic file. See also statistical report generator.

database request module (DBRM). A data set member created by the DB2 precompiler that contains information about SQL statements. DBRMs are used in the DB2 bind process. See DB2 bind.

data-collection area. An ECB area used by the ALCS online monitor for accumulating statistics about an entry.

data event control block (DECB). An ALCS control block, that may be acquired dynamically by an entry to provide a storage level and data level in addition to the 16 ECB levels. It is part of entry storage.

The ALCS DECB is independent of the MVS control block with the same name.

Data Facility Storage Management Subsystem (DFSMS*). An MVS operating environment that helps automate and centralize the management of storage. It provides the storage administrator with control over data class, management class, storage group, and automatic class selection routine definitions.

Data Facility Sort (DFSORT*). An MVS utility that manages sorting and merging of data.

data file. A sequential data set, created by the system test compiler (STC) or by the `ZDATA DUMP` command, that contains data to be loaded on to the real-time database. (An ALCS command `ZDATA LOAD` can be used to load data from a data file to the real-time database.) A data file created by STC is also called a “pilot” or “pilot tape”.

data level. An area in the ECB or a DECB used to hold the file address, and other information about a record. See ECB level and DECB level.

data record information library (DRIL). A data set used by the system test compiler (STC) to record the formats of data records on the real-time system. DRIL is used when creating data files.

DB2 application plan. The control structure produced during the bind process and used by DB2 to process SQL statements encountered during program execution. See DB2 bind.

DB2 bind. The process by which the output from the DB2 precompiler is converted to a usable control structure called a package or an application plan. During the process, access paths to the data are selected and some authorization checking is performed.

DB2 Call Attach Facility (CAF). An interface between DB2 and batch address spaces. CAF allows ALCS to access DB2.

DB2 for z/OS. An IBM licensed program that provides relational database services.

DB2 host variable. In an application program, an application variable referenced by embedded SQL statements.

DB2 package. Also called application package. An object containing a set of SQL statements that have been bound statically and that are available for processing. See DB2 bind.

DB2 package list. An ordered list of package names that may be used to extend an application plan.

DECB level. When an application program, running under ALCS, reads a record from a file, it must “own” a storage block in which to put the record. The address of the storage block may be held in an area of a DECB called a storage level.

Similarly, there is an area in a DECB used for holding the 8-byte file address, record ID, and record code check (RCC) of a record being used by an entry. This is a data level.

The storage level and data level in a DECB, used together, are called a DECB level.

See also ECB level.

diagnostic file. See ALCS diagnostic file.

dispatching priority. A number assigned to tasks, used to determine the order in which they use the processing unit in a multitasking situation.

dispense (a pool-file record). To allocate a long-term or short-term pool-file record to a particular entry. ALCS performs this action when requested by an application program. See release a pool-file record.

double-byte character set. A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets.

Because each character requires 2 bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software that are DBCS-capable.

duplex. A communication link on which data can be sent and received at the same time. Synonymous with full duplex. Communication in only one direction at a time is called “half-duplex”. Contrast with simplex transmission.

duplex database. Synonym for duplicated database.

duplicated database. A database where each data set is a mirrored pair. In ALCS, you can achieve this using either ALCS facilities or DASD controller facilities (such as the IBM 3990 dual copy facility). See mirrored pair.

dynamic program linkage. Program linkage where the connection between the calling and called program is established during the execution of the calling program. In ALCS dynamic program linkage, the connection is established by the ALCS ENTER/BACK services. Contrast with static program linkage.

dynamic SQL. SQL statements that are prepared and executed within an application program while the program is executing. In dynamic SQL, the SQL source is contained in host language variables rather than being coded into the application program. The SQL statement can change several times during the application program's execution. Contrast with embedded SQL.

E

ECB-controlled program. A program that runs under the control of an entry control block (ECB). These programs can be application programs or programs that are part of ALCS, for example the ALCS programs that process operator commands (Z messages). ECB-controlled programs are known as E-type programs in TPF.

ECB level. When an application program, running under ALCS, reads a record from file, it must “own” a storage block in which to put the record. The address of the storage block may be held in an area of the ECB called a storage level.

There are 16 storage levels in the ECB. A storage block with its address in slot zero in the ECB is said to be attached on level zero.

Similarly, there are 16 areas in the ECB that may be used for holding the 4-byte file addresses, record ID, and record code check (RCC) of records being used by an entry. These are the 16 data levels.

Storage levels and data levels, used together, are called ECB levels.

See also DECB level.

embedded SQL. Also called static SQL. SQL statements that are embedded within an application program and are prepared during the program preparation process before the program is executed. After it is prepared, the statement itself does not change (although values of host variables specified within the statement can change). Contrast with dynamic SQL.

Emulation Program/Virtual Storage (EP/VS). A component of NCP/VS that ALCS V2 uses to access SLC networks.

ENTER/BACK. The general term for the application program linkage mechanism provided by ALCS.

entry. The basic work scheduling unit of ALCS. An entry is represented by its associated entry control block (ECB). It exists either until a program that is processing that entry issues an `EXITC` monitor-request macro (or equivalent C function), or until it is purged from the system. An entry is created for each input message, as well as for certain purposes unrelated to transactions. One transaction can therefore generate several entries.

entry control block (ECB). A control block that represents a single entry during its life in the system.

entry dispatcher. See ALCS entry dispatcher.

entry macro trace block. There is a macro trace block for each entry. Each time an entry executes a monitor-request macro (or a corresponding C function), ALCS records information in the macro trace block for the entry.

This information includes the macro request code, the name of the program that issued the macro, and the displacement in the program. The ALCS diagnostic file processor formats and prints these macro trace blocks in ALCS system error dumps.

See also system macro trace block.

entry storage. The storage associated with an entry. It includes the ECB for the entry, storage blocks that are attached to the ECB or DECBs, storage blocks that are detached from the ECB or DECBs, automatic storage blocks, and DECBs. It also includes heap storage (for high-level language or assembler language programs) and stack storage (for high-level language programs).

equate. Informal term for an assignment instruction in assembler languages.

error index byte (EIB). See SLC error index byte.

extended buffer. A storage area above 2 GB used for large messages.

extended message format. For input and output messages, a message format which includes a 4-byte field for the message length.

Execute Channel Program (EXCP). An MVS macro used by ALCS V2 to interface to I/O subsystems for SLC support.

F

fetch access. Access which only involves reading (not writing). Compare with store access.

file address. 4-byte (8 hexadecimal digits) value or 8-byte value in 4x4 format (low order 4-bytes contain a 4-byte file address, high order 4 bytes contain hexadecimal zeros) that uniquely identifies an ALCS record on DASD. `FIND/FILE` services use the file address when reading or writing DASD records. See fixed file and pool file.

file address compute routine (FACE). An ALCS routine, called by a monitor-request macro (or equivalent C function) that calculates the file address of a fixed-file record. The application program provides the FACE routine with the fixed-file record type and the record ordinal number. FACE returns the 4-byte file address.

There is also an `FAC8C` monitor-request macro (or equivalent C function), that will return an 8-byte file address in 4x4 format.

FIND/FILE. The general term for the DASD I/O services that ALCS provides.

fixed file. An ALCS record class – one of the classes that reside on the real-time database. All fixed-file records are also allocatable pool records (they have a special status of “in use for fixed file”).

Within this class there are two record types reserved for use by ALCS itself (#KPTRI and #CPRCR). There can also be installation-defined fixed-file record types.

Each fixed-file record type is analogous to a relative file. Applications access fixed-file records by specifying the fixed-file record type and the record ordinal number. Note however that fixed-file records are not physically organized as relative files (logically adjacent records are not necessarily physically adjacent).

See real-time database, record class, and record type. See also system fixed file. Contrast with pool file.

fixed-file record. One of the two major types of record in the real-time database (the other is a pool-file record). When the number of records of a particular kind will not vary, the system programmer can define a fixed file record type for these records. ALCS application programs accessing fixed-file records use the `ENTRC` monitor-request macro to invoke the 4-byte file address compute routine (FACE or FACS) or use the `FAC8C` monitor-request macro to compute an 8-byte file address. The equivalent C functions are `face` or `facs` or `tpf_fac8c`.

fixed-file record type. (Known in TPF as FACE ID.) The symbol, by convention starting with a hash sign (#)[4] which identifies a particular group of fixed-file records. It is called the fixed-file record type symbol. The equated value of this symbol (called the fixed-file record type value) also identifies the fixed-file record type.

forward chain. The third fullword of a record stored on the ALCS database (part of the record header). When standard forward chaining is used, this field contains the file address of the next record in the chain, except that the last (or only) record contains binary zeros.

full-duplex. Deprecated term for duplex.

functional message. See ALCS command.

G

general data set (GDS). The same as a general file, but accessed by different macros or C functions in ALCS programs.

general file. (1) A DASD data set (VSAM cluster) that is used to communicate data between offline utility programs and the online system. General files are not part of the real-time database. (2) The ALCS record class that includes all records on the general files and general data sets. Each general file and general data set is a separate record type within this class. See record class and record type.

general file record. A record on a general file.

generalized trace facility (GTF). An MVS trace facility. See also ALCS trace facility.

general sequential file. A class of sequential data set that is for input or output. ALCS application programs must have exclusive access to a general sequential file before they can read or write to it. See also real-time sequential file.

general tape. TPF term for a general sequential file.

general-use programming interface (GUPI). An interface intended for general use in customer-written applications.

get file storage (GFS). The general term for the pool file dispense mechanisms that ALCS provides.

---

[4] This character might appear differently on your equipment. It is the character represented by hexadecimal 7B.

global area. See application global area.

global resource serialization. The process of controlling access of entries to a global resource so as to protect the integrity of the resource.

H

half-duplex. A communication link that allows transmission in one direction at a time. Contrast with duplex.

halt. (1) The ALCS state when it is terminated. (2) The action of terminating ALCS.

heap. An area of storage that a compiler uses to satisfy requests for storage from a high-level language (for example, `calloc` or `malloc` C functions). ALCS provides separate heaps for each entry (if needed). The heap is part of entry storage. Assembler language programs may also obtain or release heap storage using the `CALOC`, `MALOC`, `RALOC`, and `FREEC` monitor-request macros.

High Level Assembler (HLASM). A functional replacement for Assembler H Version 2. HLASM contains new facilities for improving programmer productivity and simplifying assembler language program development and maintenance.

high-level language (HLL). A programming language such as C or COBOL.

high-level language (HLL) storage unit. Alternative name for a type 2 storage unit. See storage unit.

high-level network (HLN). A network that provides transmission services between transaction processing systems (for example, ALCS) and terminals. Strictly, the term “high-level network” applies to a network that connects to transaction processing systems using SLC. But in ALCS publications, this term is also used for a network that connects by using AX.25 or MATIP.

high-level network designator (HLD). The entry or exit point of a block in a high-level network. For SLC networks, it is the SLC address of a switching center that is part of a high-level network. It comprises two bytes in the 7-bit transmission code used by SLC.

HLN entry address (HEN). The high-level designator of the switching center where a block enters a high-level network.

HLN exit address (HEX). The high-level designator of the switching center where a block leaves a high-level network.

hold. A facility that allows multiple entries to share data, and to serialize access to the data. The data can be a database record, or any named data resource. This facility can be used to serialize conflicting processes. See also record hold and resource hold.

host variable. See DB2 host variable

I

information block. See SLC link data block.

initial storage allocation (ISA). An area of storage acquired at initial entry to a high-level language program. ALCS provides a separate ISA for each entry (if required). The ISA is part of entry storage.

initiation queue. In message queuing, a local queue on which the queue manager puts trigger messages. You can define an initiation queue to ALCS, in order to start an ALCS application automatically when a trigger message is put on the queue. See trigger message.

input/output control block (IOCB). A control block that represents an ALCS internal “task”. For example, ALCS uses an IOCB to process a DASD I/O request.

input queue. In message queuing with ALCS, you can define a local queue to ALCS in order to start an ALCS application automatically when a message is put on that queue. ALCS expects messages on the input queue to be in PPMSG message format. See PPMSG.

installation-wide exit. The means specifically described in an IBM software product’s documentation by which an IBM software product may be modified by a customer’s system programmers to change or extend the functions of the IBM software product. Such modifications consist of exit routines written to replace an existing module of an IBM software product, or to add one or more modules or subroutines to an IBM software product for the purpose of modifying (including extending) the functions of the IBM software product. Contrast with user exit.

instruction step. One mode of operation of the ALCS trace facility. Instruction step is a conversational trace facility that stops the traced application program before the execution of each processor instruction.

Interactive System Productivity Facility (ISPF). An IBM licensed program that serves as a full-screen editor and dialog manager. ISPF provides a means of generating standard screen panels and interactive dialog between the application programmer and terminal user.

interchange address (IA). In ALC, the 1-byte address of a terminal interchange. Different terminal interchanges connected to the same ALC link have different interchange addresses. Different terminal interchanges connected to different ALC links can have

the same interchange address. See also terminal interchange

International Programmed Airlines Reservation System (IPARS). A set of applications for airline use. The principal functions are reservations and message switching.

IPARS for ALCS. The ALCS shipment includes IPARS as a sample application, and installation verification aid for ALCS.

K

KCN. Abbreviation for an SLC channel number. See SLC channel.

keypointable. See application global area.

keypoint B (CTKB). A record that contains dynamic system information that ALCS writes to DASD when it is updated so that ALCS can restart from its latest status.

L

Language Environment*. A common run-time environment and common run-time services for z/OS high level language compilers.

level. See ECB level.

line number (LN). (1) In ALC, the 1-byte address of an ALC link. Different links connected to the same communication controller have different line numbers. Different links connected to different communication controllers can have the same line number.
(2) Synonym for symbolic line number.

Link Control — Airline (LICRA). The name of a programming request for price quotation (PRPQ) to the IBM 3705 Emulation Program (EP/VS). This modifies EP/VS to support SLC networks.

link control block (LCB). See SLC link control block.

link data block (LDB). See SLC link data block.

link trace. See SLC link trace.

local DXCREI index (LDI). The first byte of a communication resource indicator (CRI).

local queue. In message queuing, a queue that belongs to the local queue manager. A local queue can contain a list of messages waiting to be processed. Contrast with remote queue.

lock. A serialization mechanism whereby a resource is restricted for use by the holder of the lock. See also hold.

log. See ALCS update log.

logging. The process of writing copies of altered database records to a sequential file. This is the method used to provide an up-to-date copy of the database should the system fail and the database have to be restored. The database records are logged to the ALCS update log file.

logical end-point identifier (LEID). In NEF2 and ALCI environments, a 3-byte identifier assigned to an ALC terminal.

logical unit type 6.2 (LU 6.2). The SNA logical unit type that supports general communication between programs in a distributed processing environment; the SNA logical unit type on which Common Programming Interface – Communications (CPI-C) is built.

log in. TPF term for establishing routing between a terminal and an application.

log on. Establish a session between an SNA terminal and an application such as ALCS. See also routing.

logon mode. In VTAM, a set of predefined session parameters that can be sent in a BIND request. When a set is defined, a logon mode name is associated with the set.

logon mode table. In VTAM, a table containing several predefined session parameter sets, each with its own logon mode name.

long message transmitter (LMT). A part of the IPARS application that is responsible for blocking and queuing printer messages for output. Also called XLMT.

long-term pool. An ALCS record class – one of the classes that reside on the real-time database. Within this class, there is one record type for each DASD record size. All long-term pool-file records are also allocatable pool records. ALCS application programs can use long-term pool records for long-lived or high-integrity data. See pool file, real-time database, record class, and record type.

L0, L1, L2, L3, ..., L8. Assembler symbols (and defined values in C) for the storage block sizes and record sizes that ALCS supports. See DASD record and storage block size.

M

macro trace block. See entry macro trace block and system macro trace block.

Mapping of Airline Traffic over IP (MATIP). A protocol for transporting traditional airline messages over an IP (Internet Protocol) network. Internet RFC (Request for Comments) number 2351 describes the MATIP protocol.

MBI exhaustion. The condition of an SLC link when a sender cannot transmit another message because all 7 SLC message labels are already “in use”; that is, the sender must wait for acknowledgement of a message so that it can reuse the corresponding message label. See also SLC link, SLC message label, and SLC message block indicator.

message. For terminals with an Enter key, an input message is the data that is sent to the host when the Enter key is hit. A response message is the data that is returned to the terminal. WTTY messages have special “start/end of message” character sequences. One or more input and output message pairs make up a transaction.

message block indicator. See SLC message block indicator.

message label. See SLC message label.

Message Queue Interface (MQI). The programming interface provided by the IBM WebSphere MQ message queue managers. This programming interface allows application programs to access message queuing services.

message queue manager. See queue manager.

message queuing. A programming technique in which each program within an application communicates with the other programs by putting messages on queues. This enables asynchronous communication between processes that may not be simultaneously active, or for which no data link is active. The message queuing service can assure subsequent delivery to the target application.

message switching. An application that routes messages by receiving, storing, and forwarding complete messages. IPARS for ALCS includes a message switching application for messages that conform to ATA/IATA industry standards for interline communication *ATA/IATA Interline Communications Manual,* DOC.GEN/1840.

mirrored pair. Two units that contain the same data and are referred to by the system as one entity.

monitor-request macro. Assembler language macro provided with ALCS, corresponding to TPF “SVC-type” or “control program” macros. Application programs use these macros to request services from the online monitor.

MQ Bridge. The ALCS MQ Bridge allows application programs to send and receive messages using WebSphere MQ for z/OS queues, without the need to code MQ calls in those programs. The MQ Bridge installation-wide monitor exits USRMQB0, USRMQB1, USRMQB2, and USRMQB3 allow you to customize the behaviour of the MQ Bridge to suit your applications.

MQSeries*. A previous name for WebSphere MQ.

multibyte character. A mixture of single-byte characters from a single-byte character set and double-byte characters from a double-byte character set.

multiblock message. In SLC, a message that is transmitted in more than one link data block. See link data block.

Multiple Virtual Storage/Data Facility Product (MVS/DFP*). An MVS licensed program that isolates applications from storage devices, storage management, and storage device hierarchy management.

Multisystem Networking Facility (MSNF). An optional feature of VTAM that permits these access methods, together with NCP, to control a multiple-domain network.

N

namelist. In message queuing, a namelist is an object that contains a list of other objects.

native file address. For migration purposes ALCS allows two or more file addresses to refer to the same database or general file record. The file address that ALCS uses internally is called the native file address.

NCP Packet Switching Interface (NPSI). An IBM licensed program that allows communication with X.25 lines.

NetView*. A family of IBM licensed programs for the control of communication networks.

NetView operator identifier (NetView operator ID). A 1- to 8-character name that identifies a NetView operator.

NetView program. An IBM licensed program used to monitor a network, manage it, and diagnose network problems.

NetView resource. A NetView operator ID which identifies one of the following:

- A NetView operator logged on to a terminal.
- A NetView operator ID automation task. One of these tasks is used by ALCS to route RO CRAS messages to the NetView Status Monitor Log (STATMON).

network control block (NCB). A special type of message, used for communication between a transaction processing system and a high-level network (HLN). For example, an HLN can use an NCB to transmit information about the network to a transaction processing system.

For a network that connects using SLC, an NCB is an SLC link data block (LDB). Indicators in the LDB differentiate NCBs from other messages.

For a network that connects using AX.25, NCBs are transmitted across a dedicated permanent virtual circuit (PVC).

Network Control Program (NCP). An IBM licensed program resident in an IBM 37xx Communication Controller that controls attached lines and terminals, performs error recovery, and routes data through the network.

Network Control Program Packet Switching Interface (NPSI). An IBM licensed program that provides a bridge between X.25 and SNA.

Network Control Program/Virtual Storage (NCP/VS). An IBM licensed program. ALCS V2 uses the EP/VS component of NCP/VS to access SLC networks.

Network Extension Facility (NEF). The name of a programming request for price quotation (PRPQ P09021) that allows management of ALC networks by NCP; now largely superseded by ALCI.

Network Terminal Option (NTO). An IBM licensed program that converts start-stop terminal device communication protocols and commands into SNA and VTAM communication protocols and commands. ALCS uses NTO to support World Trade Teletypewriter (WTTY).

O

object. In message queuing, objects define the attributes of queue managers, queues, process definitions, and namelists.

offline. A function or process that runs independently of the ALCS online monitor. For example, the ALCS diagnostic file processor is an offline function. See also ALCS offline program.

online. A function or process that is part of the ALCS online monitor, or runs under its control. For example, all ALCS commands are online functions. See also ALCS online monitor.

open. Allocate a sequential file data set to ALCS and open it (MVS `OPEN` macro). For general sequential files this is a function of the `TOPNC` monitor-request macro (or equivalent C function). ALCS automatically opens other sequential files during restart.

operator command. See ALCS command. Can also refer to non-ALCS commands, for example, MVS or VTAM commands.

ordinal. See communication resource ordinal and record ordinal.

P

package. See DB2 package

package list. See DB2 package list

padded ALC. A transmission code that adds one or more bits to the 6-bit airline line control (ALC) transmission code so that each ALC character occupies one character position in a protocol that uses 7- or 8-bit transmission codes. See also airlines line control.

padded SABRE. Synonym for padded ALC.

passenger name record (PNR). A type of record commonly used in reservation systems. It contains all the recorded information about an individual passenger.

path. The set of components providing a connection between a processor complex and an I/O device. For example, the path for an IBM 3390 DASD volume might include the channel, ESCON Director, 3990 Storage Path, 3390 Device Adapter, and 3390 internal connection. The specific components used in a particular path are dynamic and may change from one I/O request to the next. See balanced path.

pathlength. The number of machine instructions needed to process a message from the time it is received until the reponse is sent to the communication facilities.

performance monitor. An online function that collects performance data and stores it in records on the ALCS real-time database. It can produce online performance reports based on current data and historical data.

pilot. See data file.

pool directory update (PDU). A facility of TPF that recovers long-term pool file addresses without running

Recoup. PDU identifies and makes available all long-term pool-file records that have been released.

pool file. Short-term pool, long-term pool, and allocatable pool. Within each pool file class, there is one record type for each record size; for example, short-term pool includes the record type L1STPOOL (size L1 short-term pool records).

Each pool-file record type contains some records that are in-use and some that are available. There is a dispense function that selects an available record, changes its status to in-use, and returns the file address. Also, there is a release function that takes the file address of an in-use pool-file record and changes the record status to available.

To use a pool-file record, a program must:

1. Request the dispense function. This returns the file address of a record. Note that the record contents are, at this stage, unpredictable.
2. Write the initial record contents, using the file address returned by step 1.
3. Save the file address returned by step 1.
4. Read and write the record to access and update the information as required. These reads and writes use the file address saved in step 3.

When the information in the record is no longer required, a program must:

5. Delete (clear to zeros) the saved copy of the file address (see step 3).
6. Request the release function.

See also record class. Contrast with fixed file.

pool file directory record (PFDR). The ALCS pool file management routine keeps a directory for each size (L1, L2, ...L8) of short-term pool file records and long-term pool-file records. It keeps these directories in pool file directory records.

pool-file record. ALCS application programs access pool-file records with file addresses similar to those for fixed-file records. To obtain a pool-file record, an application program uses a monitor-request macro (or equivalent C function) that specifies a 2-byte record ID or a pool-file record type.

When the data in a pool-file record is no longer required, the application uses a monitor-request macro (or equivalent C function) to release the record for reuse. See pool file.

pool-file record identifier (record ID). The record ID of a pool-file record. On get file requests (using the `GETFC` monitor-request macro or equivalent C function) the program specifies the pool-file record ID. This identifies whether the pool-file record is a short-term or long-term pool-file record and also determines the record size (L1, L2, ...L8). (Coding the 2-byte record IDs, and the corresponding pool-file record sizes and types, is part of the ALCS generation procedure.) See also record ID qualifier.

pool-file record type. Each collection of short-term and long-term pool-file records of a particular record size (identified by the symbols L1, L2, ..., L8) is a different record type. Each pool-file record type has a different name. For short-term pool-file records, this is L*n*STPOOL, where L*n* is the record size symbol. For long-term pool-file records the name is L*n*LTPOOL.

post processor. See ALCS diagnostic file processor.

PPMSG. ALCS program-to-program message format, used by the ALCS message router to send and receive messages on a message routing path to another system. In PPMSG message format, the routing control parameter list (RCPL) precedes the message text.

primary action code. The first character of any input message. The primary action code Z is reserved for ALCS commands. See secondary action code.

Prime CRAS. The primary display terminal, or NetView ID, that controls the ALCS system. See also computer room agent set (CRAS).

process definition object. In message queuing, an object that contains the definition of a message queuing application. For example, a queue manager uses the definition when it works with trigger messages.

product sensitive programming interface (PSPI). An interface intended for use in customer-written programs for specialized purpose only, such as diagnosing, modifying, monitoring, repairing, tailoring or tuning of ALCS. Programs using this interface may need to be changed in order to run with new product releases or versions, or as a result of service.

program linkage. Mechanism for passing control between separate portions of the application program. See dynamic program linkage and static program linkage.

program nesting level. One of 32 ECB areas used by the ENTER/BACK mechanism for saving return control data.

program-to-program interface. In NetView, a facility that allows user programs to send data to, or receive data from, other user programs. It also allows system and application programs to send alerts to the NetView hardware monitor.

P.1024. A SITA implementation of SLC. See SLC.

P.1124. A SITA implementation of SLC. See SLC.

P.1024A. The SITA implementation of airline line control (ALC).

Q

queue manager. A system program that provides queuing services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns. WebSphere MQ for z/OS is an example of a queue manager.

R

real-time database. The database to which ALCS must have permanent read and write access. As an ALCS generation option, the real-time database can be duplicated in order to minimize the effects of a DASD failure.

real-time sequential file. A sequential data set used only for output. ALCS application programs can write to any real-time sequential file without requiring exclusive access to the data set. See also general sequential file.

real-time tape. TPF term for a real-time sequential file.

receive only (RO). The function of a communication terminal that can receive but not send data. An example is a printer that does not have a keyboard.

receive only CRAS. A printer terminal (or NetView operator ID) that ALCS uses to direct status messages. Commonly known as RO CRAS.

record. A set of data treated as a unit.

record class. The first (highest) level categorization of ALCS DASD records. ALCS defines the following record classes:

Allocatable pool
Application fixed file
Configuration data set
General file
Long-term pool
Short-term pool
System fixed file.

See also record type and record ordinal.

record code check (RCC). The third byte of any record stored in the ALCS database. It is part of the record header.

The RCC field is intended to help detect the incorrect chaining of records which have the same record ID. This is particularly useful for passenger name records (PNRs), of which there are often hundreds of thousands. A mismatch in RCC values shows that the chain is broken, probably as a result of an application program releasing a record too soon. (A false match cannot be excluded, but the RCC should give early warning of a chaining problem.)

record header. A standard format for the first 16 bytes of a record stored on the ALCS database. It contains the following fields:

Record ID
Record code check
Control byte
Application program name
Forward chain
Backward chain.

Not all records contain forward chains and backward chains. Some applications extend the record header by including extra fields. TPFDF uses an extended record header.

record hold. A type of hold that applies to DASD records. Applications that update records can use record hold to prevent simultaneous updates. See also resource hold.

record identifier (record ID). The first two bytes of a record stored on the ALCS database, part of the record header.

The record ID should always be used to indicate the nature of the data in the record. For example, airlines reservations applications conventionally store passenger name records (PNRs) as long-term pool-file records with a record ID of `'PR'`.

When application programs read such records, they can (optionally) request ALCS to check that the record ID matches that which the application program expects.

When application programs request ALCS to dispense pool file records, ALCS uses the record ID to select an appropriate long-term or short-term pool-file record of the requested record size (L1, L2,...,L8). See also record ID qualifier.

record ID qualifier. A number 0 through 9 that differentiates between record types that have the same record ID.

For compatibility with previous implementations of the record ID qualifier, ALCS also accepts the character qualifiers P and O. P (primary) is equivalent to 0, and O (overflow) is equivalent to 1.

record ordinal. The relative record number within a record type. See record class and record type.

record size. See DASD record.

record type. The second level categorization of ALCS DASD records. Within any one record class, the records are categorized into one or more record types. See also record type number, record type symbol, record class and record ordinal.

record type number. A number that identifies a record type.

record type symbol. The character string that identifies a fixed-file record type (#*xxxxx*), a long-term pool-file record type (LsLTPOOL), a short-term pool-file record type (LsSTPOOL), or a general file (GF-*nnn*). The value of the record type symbol is the record type number.

Recoup. A real-time database validation routine which runs online in the ALCS system. (Note that, while the Recoup routines of TPF consist of a number of phases, some online and some offline, the ALCS Recoup is a single online phase that runs, without operator intervention, in any system state.)

Recoup reads selected fixed-file records in the database, and then follows up all chains of pool-file records in the database, noting that these records are in use and giving a warning of any that have been corrupted or released. It then updates the pool file directory records (PFDRs) to show the status of all records.

The ALCS pool file dispense procedure identifies records not in a chain (and so apparently available for reuse) that have not been released.

recoup descriptors. These describe the structure of the entire real-time database.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks. All ALCS application programs must be reentrant.

relational database. A database that is in accordance with the relational model of data. The database is perceived as a set of tables, relationships are represented by values in tables, and data is retrieved by specifying a result table that can be derived from one or more base tables.

release (a pool-file record). To make available a long-term or short-term pool-file record so that it can be subsequently dispensed. An application program requests the release action. See dispense a pool-file record.

release file storage (RFS). The general term for the pool-file release mechanisms that ALCS provides.

remote queue. In message queuing, a queue that belongs to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with local queue.

remote terminal trace. One mode of operation of the ALCS trace facility. Remote terminal trace is a conversational trace facility to interactively trace entries from a terminal other than your own.

reservations. An online application which is used to keep track of seat inventories, flight schedules, and other related information. The reservation system is designed to maintain up-to-date data and to respond within seconds or less to inquiries from ticket agents at locations remote from the computing system.

IPARS for ALCS includes a sample reservations application for airlines.

reserve. Unassign a general sequential file from an entry but leave the file open, so that another (or the same) entry can assign it. Application programs can use the `TRSVC` monitor-request macro (or equivalent C function) to perform this action.

resource. Any facility of a computing system or operating system required by a job or task, and including main storage, input/output devices, processing unit, data sets, and control or processing programs. See also communication resource.

resource entry index (REI). The second and third bytes of a communication resource identifier (CRI).

resource hold. A type of hold that can apply to any type of resource. Applications can define resources according to their requirements, and identify them to ALCS using a unique name. See also record hold.

RO CRAS. See receive only CRAS.

rollback. An operation that reverses all the changes made during the current unit of recovery. After the operation is complete, a new unit of recovery begins.

routing. The connection between a communication resource connected to ALCS (typically a terminal on an SNA or non-SNA network) and an application (running under ALCS or another system). Also sometimes called “logging in”, but this must be distinguished from logging on, which establishes the SNA connection (session) between the terminal and ALCS.

routing control parameter list (RCPL). A set of information about the origin, destination, and characteristics of a message. With each input message, ALCS provides an RCPL in the ECB. An output message that is sent using the ROUTC (routc) service also has an RCPL associated with it.

S

scroll. To move a display image vertically or horizontally to view data that otherwise cannot be observed within the boundaries of the display screen.

secondary action code. The second character of an ALCS command. (ALCS commands are made up of 5 characters: Z followed by a secondary action code.) See primary action code.

sequential file. A file in which records are processed in the order in which they are entered and stored in the file. See general sequential file and real-time sequential file.

serialization. A service that prevents parallel or interleaved execution of two or more processes by forcing the processes to execute serially.

For example, two programs can read the same data item, apply different updates, and then write the data item. Serialization ensures that the first program to start the process (read the item) completes the process (writes the updated item) before the second program can start the process – the second program applies its update to the data item which already contains the first update. Without serialization, both programs can start the process (read the item) before either completes the process (writes the updated item) – the second write destroys the first update. See also assign, lock, and hold.

Serviceability Level Indicator Processing (SLIP). An MVS operator command which acts as a problem determination aid.

short-term pool. An ALCS record class – one of the classes that resides on the real-time database. Within this class, there is one record type for each DASD record size. All short-term pool-file records are also allocatable pool records (they have a special status of "in use for short-term pool"). ALCS application programs can use short-term pool records for short-lived low-integrity data. See pool file, real-time database, record class, and record type.

simplex transmission. Data transmission in one direction only. See also duplex and half-duplex.

sine in/out. Those applications that provide different functions to different end users of the same application can require the user to sine in[5] to the specific functions they require. The sine-in message can, for example, include an authorization code.

single-block message. In SLC, a message that is transmitted in one link data block. See link data block.

single-phase commit. A method in which a program can commit updates to a message queue or relational database without coordinating those updates with updates the program has made to resources controlled by another resource manager. Contrast with two-phase commit.

SLC. See synchronous link control.

SLC channel. A duplex telecommunication line using ATA/IATA SLC protocol. There can be from 1 to 7 channels on an SLC link.

SLC error index byte (EIB). A 1-byte field generated by Line Control – Airline (LICRA) and transferred to ALCS with each incoming link control block and link data block. Certain errors cause LICRA to set on certain bits of the EIB. See also Link Control — Airline (LICRA).

SLC information block. Synonym for SLC link data block.

SLC link. A processor-to-processor or processor-to-HLN connection. ALCS supports up to 255 SLC links in an SLC network.

An SLC link that is in the process of an open, close, start, or stop function is said to be "cycling".

SLC link control block (LCB). A 4-byte data item transmitted across an SLC link to control communications over the link. LCBs are used, for example, to confirm that a link data block (LDB) has arrived, to request retransmission of an LDB, and so on.

SLC link data block (LDB). A data item, transmitted across an SLC link, that contains a message or part of a message. One LDB can contain a maximum of 240 message characters, messages longer than this must be split and transmitted in multiple LDBs. Synonymous with SLC information block.

SLC link trace. A function that provides a record of SLC communication activity. It can either display the information in real time or write it to a diagnostic file for offline processing, or both. Its purpose is like that of an NCP line trace, but for the SLC protocol.

SLC message block indicator (MBI). A 1-byte field in the SLC link data block that contains the SLC message label and the block number. A multiblock message is transmitted in a sequence of up to 16 link data blocks

[5] This spelling is established in the airline industry.

with block numbers 1, 2, 3, ... 16. See also multiblock message, SLC link data block, and SLC message label.

SLC message label. A number in the range 0 through 7, excluding 1. In P.1024, consecutive multiblock messages are assigned SLC message labels in the sequence: 0, 2, 3, ... 6, 7, 0, 2, and so on. In P.1124, single-block messages are (optionally) also included in the sequence. See also P.1024, P.1124 and SLC message block indicator.

SLC transmission status indicator (TSI). A 1-byte field in the SLC link data block that contains the SLC transmission sequence number. See also SLC transmission sequence number.

SLC transmission sequence number (TSN). A number in the range 1 through 31. Consecutive SLC link data blocks transmitted in one direction on one SLC channel are assigned TSNs in the sequence: 1, 2, 3, ... 30, 31, 1, 2, and so on. See also SLC link data block, SLC channel, and SLC transmission status indicator.

SLC Type A traffic. See Type A traffic.

SLC Type B traffic. See Type B traffic.

Société Internationale de Télécommunications Aéronautiques (SITA). An international organization which provides communication facilities for use within the airline industry.

SQL Communication Area (SQLCA). A structure used to provide an application program with information about the execution of its SQL statements.

SQL Descriptor Area (SQLDA). A structure that describes input variables, output variables, or the columns of a result table used in the execution of manipulative SQL statements.

stack. An area of storage that a compiler uses to allocate variables defined in a high-level language. ALCS provides separate stacks for each entry (if needed). The stack is part of entry storage.

standard message format. For input and output messages, a message format which includes a 2-byte field for the message length.

standby. The state of ALCS after it has been initialized but before it has been started. Standby is not considered one of the system states.

static program linkage. Program linkage where the connection between the calling and called program is established before the execution of the program. The connection is established by the assembler, compiler, prelinker, or linkage editor. Static program linkage does not invoke ALCS monitor services. See also dynamic program linkage.

static SQL. See embedded SQL.

statistical report generator (SRG). An offline ALCS utility that is a performance monitoring tool. It takes the data written to the ALCS data collection or diagnostic file processor by the data collection function and produces a variety of reports and bar charts. The SRG is the equivalent of TPF “data reduction”.

STATMON. See NetView resource.

storage block. An area of storage that ALCS allocates to an entry. It is part of entry storage. See storage block sizes.

storage block size. ALCS allows storage blocks of up to 9 different sizes. These are identified in programs by the assembler symbols (or defined C values) L0, L1, L2, ..., L8. Installations need not define all these block sizes but usually define at least the following:

Size L0 contains 127 bytes of user data
Size L1 contains 381 bytes of user data
Size L2 contains 1055 bytes of user data
Size L3 contains 4000 bytes of user data
Size L4 contains 4095 bytes of user data.

The system programmer can alter the size in bytes of L1 through L4, and can specify the remaining block sizes.

storage level. An area in the ECB or a DECB used to hold the address and size of a storage block. See ECB level and DECB level.

storage unit. The ALCS storage manager allocates storage in units called storage units. Entry storage is suballocated within storage units; for example, one storage unit can contain an ECB and several storage blocks attached to that ECB.

ALCS uses three types of storage unit:

- Prime and overflow storage units for entry storage (also called type 1 storage units).
- High-level language storage units for stack storage (also called type 2 storage units).
- Storage units for heap storage for programs (also called type 3 storage units).

The size of a storage unit, and the number of each type of storage unit, is defined in the ALCS generation. See entry storage.

store access. Access which only involves writing (not reading). Compare with fetch access.

striping. A file organization in which logically adjacent records are stored on different physical devices. This organization helps to spread accesses across a set of physical devices.

Structured Query Language (SQL). a standardized language for defining and manipulating data in a relational database.

symbolic line number (SLN). In TPF, a 1-byte address of an ALC link, derived from the line number but adjusted so that all ALC links connected to the TPF system have a different symbolic line number. See also line number.

Synchronous Data Link Control (SDLC). A discipline conforming to subsets of the Advanced Data Communication Control Procedures (ADCCP) of the American National Standards Institute (ANSI) and High-level Data Link Control (HDLC) of the International Organization for Standardization, for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection.

Transmission exchanges can be duplex or half-duplex over switched or nonswitched links. The configuration of the link connection can be point-to-point, multipoint, or loop.

Synchronous Link Control (SLC). A discipline conforming to the ATA/IATA Synchronous Link Control, as described in the ATA/IATA publication *ATA/IATA Interline Communications Manual*, ATA/IATA document DOC.GEN 1840.

syncpoint. An intermediate or end point during processing of a transaction at which the transaction's protected resources are consistent. At a syncpoint, changes to the resources can safely be committed, or they can be backed out to the previous syncpoint.

system error. Error that the ALCS monitor detects. Typically, ALCS takes a dump, called a system error dump, to the ALCS diagnostic file. See also ALCS diagnostic file and ALCS diagnostic file processor. See also system error dump, system error message.

system error dump. (1) A storage dump that ALCS writes to the ALCS diagnostic file when a system error occurs. See also ALCS diagnostic file and system error. (2) The formatted listing of a storage dump produced by the ALCS diagnostic file processor. See also ALCS diagnostic file processor.

system error message. A message that ALCS sends to receive only CRAS when a system error occurs. See also receive only CRAS and system error.

system error option. A parameter that controls what action ALCS takes when it detects a system error. See also system error.

system fixed file. An ALCS record class – one of the classes that reside on the real-time database. All system fixed-file records are also allocatable pool records (they have a special status of “in use for system fixed file”).

System fixed-file records are reserved for use by ALCS iteslf. See real-time database, record class, and record type.

system macro trace block. There is one system macro trace block. Each time an entry issues a monitor-request macro (or equivalent C function), ALCS records information in the system macro trace block.

This information includes the ECB address, the macro request code, the name of the program that issued the macro, and the displacement in the program. The ALCS diagnostic file processor formats and prints the system macro trace block in ALCS system error dumps. See also entry macro trace block.

System Modification Program/Extended (SMP/E). An IBM licensed program used to install software and software changes on MVS systems. In addition to providing the services of SMP, SMP/E consolidates installation data, allows flexibility in selecting changes to be installed, provides a dialog interface, and supports dynamic allocation of data sets.

Systems Application Architecture* (SAA*). A set of software interfaces, conventions, and protocols that provide a framework for designing and developing applications with cross-system consistency.

Systems Network Architecture (SNA*). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of networks.

system sequential file. A class of sequential data sets used by ALCS itself. Includes the ALCS diagnostic file, the ALCS data collection file, and the ALCS update log file or files.

system state. The ALCS system can run in any of the following system states: IDLE, CRAS, message switching (MESW), and normal (NORM).

Each state represents a different level of availability of application functions. Altering the system state is called “cycling the system”. See also standby.

system test compiler (STC). An offline ALCS utility that compiles data onto data files for loading on to the real-time database. STC also builds test unit tapes (TUTs) for use by the system test vehicle (STV).

system test vehicle (STV). An online ALCS function that reads input messages from a general sequential file test unit tape (TUT) and simulates terminal input. STV intercepts responses to simulated terminals and writes them to the ALCS diagnostic file.

T

terminal. A device capable of sending or receiving information, or both. In ALCS this can be a display terminal, a printer terminal, or a NetView operator identifier.

terminal address (TA). In ALC, the 1-byte address of an ALC terminal. Different terminals connected to the same terminal interchange have different terminal addresses. Different terminals connected to different terminal interchanges can have the same terminal address. See also terminal interchange.

terminal circuit identity (TCID). Synonym for line number.

terminal hold. When an ALCS application receives an input message, it can set terminal hold on for the input terminal. Terminal hold remains on until the application sets it off. The application can reject input from a terminal that has terminal hold set on. Also referred to as AAA hold.

terminal interchange (TI). In ALC, synonym for terminal control unit.

terminate. (1) To stop the operation of a system or device. (2) To stop execution of a program.

test unit tape (TUT). A general sequential file that contains messages for input to the system test vehicle (STV). TUTs are created by the system test compiler (STC).

time available supervisor (TAS). An ALCS or TPF function that creates and dispatches low priority entries.

time-initiated function. A function initiated after a specific time interval, or at a specific time. In ALCS this is accomplished by using the `CRETC` monitor-request macro or equivalent C function. See create service.

TP profile. The information required to establish the environment for, and attach, an APPC/MVS transaction program on MVS, in response to an inbound allocate request for the transaction program.

trace facility. See ALCS trace facility, generalized trace facility, and SLC link trace.

transaction. The entirety of a basic activity in an application. A simple transaction can require a single input and output message pair. A more complex transaction (such as making a passenger reservation) requires a series of input and output messages.

Transaction Processing Facility (TPF). An IBM licensed program with many similarities to ALCS. It runs native on IBM System/370 machines, without any intervening software (such as MVS). TPF supports only applications that conform to the TPF interface. In this book, TPF means Airline Control Program (ACP), as well as all versions of TPF.

Transaction Processing Facility Database Facility (TPFDF). An IBM licensed program that provides database management facilities for programs that run in an ALCS or TPF environment.

Transaction Processing Facility/Advanced Program to Program Communications (TPF/APPC). This enables LU 6.2 for TPF.

Transaction Processing Facility/Data Base Reorganization (TPF/DBR). A program which reorganizes the TPF real-time database.

Transaction Processing Facility/MVS (TPF/MVS). Alternative name for ALCS V2.

Transaction program identifier (TP_ID). A unique 8-character token that APPC/MVS assigns to each instance of a transaction program. When multiple instances of a transaction program are running sumultaneously, they have the same transaction program name, but each has a unique TP_ID.

transaction scheduler name. The name of an APPC/MVS scheduler program. The ALCS transaction scheduler name is ALCSx000, where x is the ALCS system identifier as defined during ALCS generation.

transfer vector. An ALCS application program written in assembler, SabreTalk, or C, can have multiple entry points for dynamic program linkage. These entry points are called transfer vectors. Each transfer vector has a separate program name.

transmission status indicator. See SLC transmission status indicator.

transmission sequence number. See SLC transmission sequence number.

trigger event. In message queuing, an event (such as a message arriving on a queue) that causes a queue manager to create a trigger message on an initiation queue.

trigger message. In message queuing, a message that contains information about the program that a trigger monitor is to start.

trigger monitor. In message queuing, a continuously-running application that serves one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. When ALCS acts as a trigger monitor, it uses the information in the trigger message to start an

ALCS application that serves the queue on which a trigger event occurred.

triggering. In message queuing, a facility that allows a queue manager to start an application automatically when predetermined conditions are met.

TSI exhaustion. The condition of an SLC channel when a sender cannot transmit another SLC link data block (LDB) because the maximum number of unacknowledged LDBs has been reached. The sender must wait for acknowledgement of at least one LDB so that it can transmit further LDBs. See also SLC channel, SLC link data block, SLC transmission sequence number, and SLC transmission status indicator.

two-phase commit. A protocol for the coordination of changes to recoverable resources when more than one resource manager is used by a single transaction. Contrast with single-phase commit.

type. See record type.

Type A traffic. ATA/IATA conversational traffic – that is, high-priority low-integrity traffic transmitted across an SLC or AX.25 link.

Type B application-to-application program (BATAP). In any system (such as ALCS) that communicates with SITA using AX.25 or MATIP, this is the program which receives and transmits type B messages.

Type B traffic. ATA/IATA conventional traffic – that is, high-integrity, low-priority traffic transmitted across an SLC or AX.25 link or a MATIP TCP/IP connection.

type 1 pool file dispense mechanism. The mechanism used in ALCS prior to V2 Release 1.3 (and still available in subsequent releases) to dispense both short-term and long-term pool-file records.

type 1 storage unit. Prime or overflow storage unit for entry storage. See storage unit.

type 2 pool file dispense mechanisms. The mechanisms available since ALCS V2 Release 1.3 to dispense pool-file records (the mechanisms are different for short-term and long-term pool-file records).

IBM recommends users to migrate to type 2 dispense mechanisms as part of their migration process.

type 2 storage unit. High-level language storage unit for stack storage. See storage unit.

type 3 storage unit. Storage unit for heap storage for programs. See storage unit.

U

unit of recovery. A recoverable sequence of operations within a single resource manager (such as WebSphere MQ for z/OS or DB2 for z/OS). Compare with unit of work.

unit of work. A recoverable sequence of operations performed by an application between two points of consistency. Compare with unit of recovery.

Universal Communications Test Facility (UCTF). An application used by SITA for SLC protocol acceptance testing.

update log. See ALCS update log.

user data-collection area. An optional extension to the data-collection area in the ECB. Application programs can use the DCLAC macro to update or read the user data-collection area.

user exit. A point in an IBM-supplied program at which a user exit routine can be given control.

user exit routine. A user-written routine that receives control at predefined user exit points. User exit routines can be written in assembler or a high-level language.

V

version number. In ALCS and TPF, two characters (not necessarily numeric), optionally used to distinguish between different versions of a program. Sometimes also used with other application components such as macro definitions.

virtual file access (VFA). An ALCS caching facility for reducing DASD I/O. Records are read into a buffer, and subsequent reads of the same record are satisfied from the buffer. Output records are written to the buffer, either to be written to DASD – immediately or at a later time – or to be discarded when they are no longer useful.

virtual SLC link. Used to address an X.25 PVC or TCP/IP resource for transmitting and receiving Type B traffic. Some applications (such as IPARS MESW) address communication resources using a symbolic line number (SLN) instead of a CRI. These applications can address X.25 PVC and TCP/IP resources by converting the unique SLN of a virtual SLC link to the CRI of its associated X.25 PVC or TCP/IP resource.

W

WebSphere* MQ for z/OS. An IBM product that provides message queuing services to systems such as CICS, IMS, ALCS or TSO. Applications request queuing services through MQI.

wide character. A character whose range of values can represent distinct codes for all members of the largest extended character set specified among the supporting locales. For the z/OS XL C/C++ compiler, the character set is DBCS, and the value is 2 bytes.

workstation trace. One mode of operation of the ALCS trace facility. Workstation trace controls the remote debugger facility. The remote debugger is a source level debugger for C/C++ application programs.

World Trade Teletypewriter (WTTY). Start-stop telegraph terminals that ALCS supports through Network Terminal Option (NTO).

Z

Z message. See ALCS command.

Bibliography

Note that unless otherwise specified, the publications listed are those for the z/OS platform.

Airline Control System Version 2 Release 4.1

- *Application Programming Guide*, SH19-6948
- *Application Programming Reference – Assembler Language*, SH19-6949
- *Application Programming Reference – C Language*, SH19-6950
- *Concepts and Facilities*, SH19-6953
- *General Information Manual*, GH19-6738
- *Installation and Customization*, SH19-6954
- *Licensed Program Specifications*, GC34-6327
- *Messages and Codes*, SH19-6742
- *Operation and Maintenance*, SH19-6955
- *Program Directory*, GI10-2577
- *ALCS World Wide Web Server User Guide*
- *OCTM User Guide*

MVS

- *Data Areas, Volumes 1 through 5*, GA22-7581 through GA22-7585
- *Diagnosis Reference*, GA22-7588
- *Diagnosis Tools and Service Aids*, GA22-7589
- *Initialization and Tuning Guide*, SA22-7591
- *Initialization and Tuning Reference*, SA22-7592
- *Installation Exits*, SA22-7593
- *IPCS Commands*, SA22-7594
- *IPCS User's Guide*, SA22-7596
- *JCL Reference*, SA22-7597
- *JCL User's Guide*, SA22-7598
- *JES2 Initialization and Tuning Guide*, SA22-7532
- *JES2 Initialization and Tuning Reference*, SA22-7533
- *Program Management: User's Guide and Reference*, SA22-7643
- *System Codes*, SA22-7626
- *System Commands*, SA22-7627
- *System Messages, Volumes 1 through 10*, SA22-7631 through SA22-7640

APPC/MVS

- *MVS Planning: APPC/MVS Management*, SA22-7599
- *MVS Programming: Writing Transaction Programs for APPC/MVS*, SA22-7621

DFSMS

- *Access Method Services for Catalogs*, SC26-7394
- *DFSMSdfp Storage Administration Reference*, SC26-7402
- *DFSMSdss Storage Administration Guide*, SC35-0423
- *DFSMSdss Storage Administration Reference*, SC35-0424
- *DFSMShsm Storage Administration Guide*, SC35-0421
- *DFSMShsm Storage Administration Reference*, SC35-0422
- *Introduction*, SC26-7397

RMF

- *RMF Report Analysis*, SC33-7991
- *RMF User's Guide*, SC33-7990

Data Facility Sort (DFSORT)

- *Application Programming Guide*, SC33-4035
- *Messages, Codes and Diagnostic Guide*, SC26-7050

Language Environment

- *Language Environment Concepts Guide*, SA22-7567
- *Language Environment Customization*, SA22-7564
- *Language Environment Debugging Guide*, GA22-7560
- *Language Environment Programming Guide*, SA22-7561
- *Language Environment Programming Reference*, SA22-7562
- *Language Environment Run-Time Messages*, SA22-7566

z/OS XL C/C++

- *Standard C++ Library Reference*, SC09-4949
- *z/OS XL C/C++ Compiler and Run-Time Migration Guide*, GC09-4913
- *z/OS XL C/C++ Language Reference*, SC09-4815
- *z/OS XL C/C++ Messages*, GC09-4819
- *z/OS XL C/C++ Programming Guide*, SC09-4765
- *z/OS XL C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS XL C/C++ User's Guide*, SC09-4767

COBOL

- *Enterprise COBOL for z/OS and OS/390 Language Reference*, SC27-1408
- *Enterprise COBOL for z/OS and OS/390 Programming Guide*, SC27-1412
- *VisualAge COBOL for OS/390 and VM Language Reference*, SC26-9046
- *VisualAge COBOL for OS/390 and VM Programming Guide*, SC26-9049

PL/I

- *Enterprise PL/I for z/OS and OS/390 Language Reference*, SC27-1460
- *Enterprise PL/I for z/OS and OS/390 Messages and Codes*, SC27-1461
- *Enterprise PL/I for z/OS and OS/390 Programming Guide*, SC27-1457
- *VisualAge PL/I for OS/390 Compile-Time Messages and Codes*, SC26-9478
- *VisualAge PL/I for OS/390 Language Reference*, SC26-9476
- *VisualAge PL/I for OS/390 Programming Guide*, SC26-9473

High Level Assembler

- *Language Reference*, SC26-4940
- *Programmer's Guide*, SC26-4941

CPI-C

- *SAA CPI-C Reference*, SC09-1308

DB2 for z/OS

- *Administration Guide*, SC18-9840
- *Application Programming and SQL Guide*, SC18-9841
- *Codes*, GC18-9843
- *Command Reference*, SC18-9844
- *Installation Guide*, GC18-9846
- *Messages*, GC18-9849
- *SQL Reference*, SC18-9854
- *Utility Guide and Reference*, SC18-9855
- *DB2 for z/OS What's New?*, GC18-9856

ISPF

- *ISPF Dialog Developer's Guide*, SC34-4821
- *ISPF Dialog Tag Language*, SC34-4824
- *ISPF Planning and Customizing*, GC34-4814

WebSphere MQ for z/OS

- *An Introduction to Messaging and Queuing*, GC33-0805
- *Application Programming Guide*, SC34-6595
- *Application Programming Reference*, SC34-6596
- *Command Reference*, SC34-6597
- *Concepts and Planning Guide*, GC34-6582
- *Intercommunication*, SC34-6587
- *Messages and Codes*, GC34-6602
- *MQI Technical Reference*, SC33-0850
- *Problem Determination Guide*, GC34-6600
- *System Administration Guide*, SC34-6585

Tivoli NetView

- *Administration Reference*, SC31-8854
- *Automation Guide*, SC31-8853
- *Installation, Getting Started*, SC31-8872
- *User's Guide*, GC31-8849
- *Security Reference*, SC31-8870

SMP/E

- *Reference*, SA22-7772
- *User's Guide*, SA22-7773

Communications Server IP (TCP/IP)

- *API Guide*, SC31-8788
- *Configuration Guide*, SC31-8775
- *Configuration Reference*, SC31-8776
- *Diagnosis Guide*, SC31-8782
- *Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534
- *IP and SNA Codes*, SC31-8791
- *Messages Volume 1*, SC31-8783
- *Messages Volume 2*, SC31-8784
- *Messages Volume 3*, SC31-8785
- *Messages Volume 4*, SC31-8786
- *Migration Guide*, SC31-8773

TPF

- *Application Programming*, SH31-0132
- *Concepts and Structures*, GH31-0139
- *C/C++ Language Support Users Guide*, SH31-0121
- *General Macros*, SH31-0152
- *Library Guide*, GH31-0146
- *System Macros*, SH31-0151

TPF Database Facility (TPFDF)

- *Database Administration*, SH31-0175
- *General Information Manual*, GH31-0177
- *Installation and Customization*, GH31-0178
- *Programming Concepts and Reference*, SH31-0179
- *Structured Programming Macros*, SH31-0183
- *Utilities*, SH31-0185

TSO/E

- *Administration*, SA22-7780
- *Customization*, SA22-7783
- *System Programming Command Reference*, SA22-7793
- *User's Guide*, SA22-7794

Communications Server SNA (VTAM)

- *IP and SNA Codes*, SC31-8791
- *Messages*, SC31-8790
- *Network Implementation* , SC31-8777
- *Operations*, SC31-8779
- *Programming*, SC31-8829
- *Programmers' LU6.2 Guide*, SC31-8811
- *Programmers' LU6.2 Reference*, SC31-8810
- *Resource Definition Reference*, SC31-8778

Security Server (RACF)

- *RACF General User's Guide*, SA22-7685
- *RACF Messages and Codes*, SA22-7686
- *RACF Security Administrator's Guide*, SA22-7683

Other IBM publications

- *IBM Dictionary of Computing*, ZC20-1699
- *IBM 3270 Information Display System: Data Stream Programmer's Reference*, GA23-0059
- *Input/Output Configuration Program User's Guide and Reference*, SB0F-3741
- *NTO Planning, Migration and Resource Definition*, SC30-3347
- *Planning for NetView, NCP, and VTAM*, SC31-7122
- *SNA Formats*, GA27-3136
- *X.25 NPSI Planning and Installation*, SC30-3470
- *z/Architecture Principles of Operation*, SA22-7832

CD-ROM Softcopy collection kits

- *IBM Online Library: Transaction Processing and Data Collection*, SK2T-0730
- *IBM Online Library: z/OS Collection*, SK3T-4269
- *IBM Online Library: z/OS Software Products Collection*, SK3T-4270
- *The Best of APPC, APPN and CPI-C Collection*, SK2T-2013

SITA publications

- *ACS protocol acceptance tool*, SITA document 032-1/LP-SD-001
- *Communications control procedure for connecting IPARS agent set control unit equipment to a SITA SP*, SITA document P.1024B (PZ.7130.1)

- *P.1X24 automatic testing (with UCTF on DIS)*, SITA document 032-1/LP-SDV-001
- *P.1024 Test Guide*, SITA Document PZ.1885.3
- *Status Control Service Step 2: Automatic Protected Report to ACS*, SITA document 085-2/LP-SD-001
- *Synchronous Link Control Procedure*, SITA Document P.1124

SITA produces a series of books which describe the SITA high level network and its protocols. These may be obtained from:

Documentation Section
SITA
112 Avenue Charles de Gaulle
92522 Neuilly sur Seine
France

Other non-IBM publications

Systems and Communications Reference Manual (Vols 1-7). This publication is available from the International Air Transport Association (IATA). You can obtain ordering information from the IATA web site <http://www.iata.org/> or contact them directly by telephone at +1(514) 390-6726 or by e-mail at Sales@iata.org.

Index

D

E

F

N

O

P

R

S

T

U

V

W

Z

Readers' Comments — We'd Like to Hear from You

Airline Control System Version 2
Concepts and Facilities
Release 4.1

Publication No. SH19-6953-13

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.

Readers' Comments — We'd Like to Hear from You
SH19-6953-13

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

ALCS Development
2455 South Road
P923
Poughkeepsie NY 12601-5400
USA

Fold and Tape | **Please do not staple** | Fold and Tape

SH19-6953-13

Cut or Fold
Along Line

Program Number: 5695-068

SH19-6953-13