



***Subroutines Reference V:
Event Synchronization***

Revision 22.0

DOC10213-1LA

Subroutines Reference V: Event Synchronization

First Edition

John Breithaupt and
Glenn Morrow

*This guide documents the software operation
of the Prime Computer and its supporting
systems and utilities as implemented at
Master Disk Revision Level 22.0 (Rev. 22.0).*

Prime Computer, Inc., Prime Park, Natick, MA 01760

Copyright Information

The information in this document is subject to change without notice and should not be construed as a commitment by Prime Computer, Inc. Prime Computer, Inc. assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Copyright © 1988 by Prime Computer, Inc., Prime Park, Natick, Massachusetts 01760

PRIME, PR1ME, PRIMOS, and the PRIME logo are registered trademarks of Prime Computer, Inc. 50 Series, 400, 750, 850, 2250, 2350, 2450, 2455, 2550, 2655, 2755, 4050, 4150, 6350, 6550, 9650, 9655, 9750, 9755, 9950, 9955, 9955II, DISCOVER, EDMS, FM+, INFO/BASIC, INFORM, Prime INFORMATION, Prime INFORMATION CONNECTION, Prime INFORMATION EXL, MDL, MIDAS, MIDASPLUS, MXCL, PRIME EXL, PRIME MEDUSA, PERFORM, PERFORMER, PRIME/SNA, PRIME TIMER, PRIMAN, PRIMELINK, PRIMENET, PRIMEWAY, PRIMEWORD, PRIMIX, PRISAM, PRODUCER, Prime INFORMATION/pc, PST 100, PT25, PT45, PT65, PT200, PT250, PW153, PW200, PW250, RINGNET and SIMPLE are trademarks of Prime Computer, Inc.

Printing History

First Edition (DOC-10213-1LA) 1988 for Revision 22.0

Credits

Editorial: Thelma Henner

Design: Carol Smith

Project Support: Nancy Lewis, Joan Karp, Helen Raizen, Mei Ng, Evelyn Tate

Illustration: Julie Cyphers, Anne Marie Fantasia, Roseanne Dickey

Document Preparation: Mary Mixon, Kathy Normington

Composition: Julie Cyphers, Sharon Temple

Production: Judy Gordon

How To Order Technical Documents

To order copies of documents, or to obtain a catalog, a price list:

United States Only: Call Prime Telemarketing, toll free, at 1-800-343-2533, Monday through Friday, 8:30 a.m. to 5:00 p.m. (EST).

International: Contact your local Prime subsidiary or distributor.

Customer Support Center

Prime provides the following toll-free numbers for customers in the United States needing service:

1-800-322-2838 (Massachusetts)
1-800-541-8888 (Alaska and Hawaii)
1-800-343-2320 (within other states)

For other locations, contact your Prime representative.

Surveys and Correspondence

Please comment on this manual using the Reader Response Form provided in the back of this book. Address any additional comments on this or other Prime documents to:

Technical Publications Department
Prime Computer, Inc.
500 Old Connecticut Path
Framingham, MA 01701

About This Book	xi
------------------------	----

**Part I
Overview**

1 Overview of Event Synchronization	1-1
Introduction	1-1
Timers	1-2
InterServer Communications	1-2
Include Files	1-3
Retrieving Error Messages	1-4

**Part II
Event Synchronizers and Event Groups**

2 Event Synchronizers	2-1
Introduction	2-1
Event Synchronizer Subroutines	2-2
Include Files for Synchronizers	2-4
Subroutine Descriptions	2-4
SYN\$CREA	2-5
SYN\$POST	2-7
SYN\$WAIT	2-9
SYN\$TMWT	2-11
SYN\$RTRV	2-13
SYN\$DEST	2-15
3 Event Groups	3-1
Introduction	3-1
Event Group Subroutines	3-1
Subroutine Descriptions	3-4
SYN\$GCRE	3-5
SYN\$MVTO	3-7
SYN\$REMV	3-9

SYN\$GWT	3-11
SYN\$GTWT	3-13
SYN\$GRTR	3-15
SYN\$GDST	3-18
4 Retrieving Information About Event Synchronization	4-1
Introduction	4-1
Subroutine Descriptions	4-1
SYN\$CHCK	4-2
SYN\$GCHK	4-4
SYN\$INFO	4-6
SYN\$LSIG	4-8
SYN\$LIST	4-10
SYN\$GLST	4-12
Part III Timers	
5 Timers	5-1
Introduction	5-1
Timer Subroutines	5-1
Include Files for Timers	5-4
Subroutine Descriptions	5-5
TMR\$CREA	5-6
TMR\$DEST	5-8
TMR\$SABS	5-9
TMR\$SINT	5-11
TMR\$SREP	5-13
TMR\$CANL	5-15
TMR\$GTMR	5-16
TMR\$LIST	5-19

Part IV

InterServer Communications

6	General Discussion of InterServer Communications	6-1
	Introduction	6-1
	A Choice of Message Types	6-2
	Programming Considerations	6-3
7	Server Names	7-1
	Introduction	7-1
	Server Name Subroutines	7-2
	Low Level Names	7-2
	Cataloging Low Level Names	7-3
	ISN\$C	7-5
	ISN\$L	7-7
	ISN\$RC	7-8
	ISN\$UC	7-9
	SRS\$GN	7-10
	SRS\$GP	7-11
	SRS\$LN	7-13
8	Establishing a Session	8-1
	Introduction	8-1
	Session Requests and Session Numbers	8-1
	Establishing a Session	8-2
	IS\$RS	8-3
	IS\$GRQ	8-6
	IS\$AS	8-9
	IS\$GRS	8-12

9	Session Configuration	9-1
	Introduction	9-1
	The Default Session Configuration	9-1
	The Session Configuration Block	9-2
	Global Session Parameters	9-3
	Local Session Parameters	9-5
	Configuring Synchronizers	9-7
	Session Synchronizers List	9-9
	Using Synchronizers	9-10
10	Message Exchange	10-1
	Introduction	10-1
	Message Exchange Subroutines	10-2
	Creating a Message	10-3
	IS\$AB	10-5
	IS\$FB	10-7
	IS\$SM	10-9
	IS\$RM	10-12
11	Session Termination and Exceptions	11-1
	Introduction	11-1
	Termination Subroutines	11-1
	The ExceptionPending Synchronizer	11-2
	IS\$TS	11-3
	IS\$GE	11-5
	IS\$CE	11-7
12	Connect Messages	12-1
	Introduction	12-1
	A Connect Message Exchange	12-2

13	Remote Sessions	13-1
	Introduction	13-1
	Remote User ID	13-1
	Network Events	13-2
14	Retrieving Session Information	14-1
	Introduction	14-1
	IS\$GSO	14-2
	IS\$GSA	14-4
	IS\$GSS	14-7
	IS\$STA	14-10

Appendices

A	Quick Reference to Calling Sequences	A-1
B	Sample Programs	B-1
	Programs Accessing Synchronizers and Groups	B-1
	Programs Accessing Timers	B-6
	Programs Using InterServer Communications	B-8
C	Status Codes	C-1
	Synchronizer Status Codes	C-1
	Synchronizer What_Happened Codes	C-2
	Timer Status Codes	C-2
	ISC Status Codes	C-3
	ISC Response Codes Returned by IS\$GRS	C-4
	ISC Exception Codes Returned by IS\$GE	C-5
	ISC Phase Codes Returned by IS\$GSS	C-5
	SRS Status Codes	C-5
D	Limits	D-1
	Synchronizer and Timer Limits	D-1
	ISC Limits	D-1

E	Data Structures	E-1
	Synchronizer Information Record	E-1
	Absolute Timer Information Record	E-1
	Interval Timer Information Record	E-1
	Repetitive Timer Information Record	E-2
	Attribute Identity Block	E-2
	Initiator Authentication Block	E-2
	Low Level Name	E-3
	Message Specifier	E-3
	Session Configuration Block	E-4
	Session Statistics Block	E-5
	Session Status Block	E-5
	Session Synchronizers List	E-6
	Target Authentication Block	E-6
F	Data Type Equivalents	F-1


Indexes

Index of Subroutines by Function	FX-1
Index of Subroutines by Name	SX-1
Index	X-1



About This Book

Overview of This Series



The *Subroutines Reference* series consists of five volumes. These volumes are organized by the functions that the subroutines perform. A brief summary of the contents of each volume is given below.

Volume I


Introduction to the five-volume series. It describes the nature and functions of Prime's standard subroutines and subroutine libraries. It explains how subroutines can be called from programs written in Prime's programming languages: C, CBL, FORTRAN IV, FORTRAN 77, Pascal, PL/I, BASIC/VM, and PMA.



Volume II


Describes file system subroutines, including subroutines for the search rules facility. It also describes subroutines for the manipulation of EPFs in the execution environment and the use of command environment functions.

Volume III



Describes system subroutines. The subroutines covered in this volume are general system calls to the operating system and the standard system library. This volume includes memory allocation and System Information and Metering (SIM) subroutines.

Volume IV



Describes the Input/Output Control System (IOCS) libraries and other I/O-related subroutines. It documents both device-dependent and device-independent subroutines and Sync and Async device-driver subroutines. It also documents Application libraries, the SORT libraries, and MATHLIB, the FORTRAN Matrix library. Many of these subroutines are of special interest for FORTRAN programmers.

Volume V

Describes event synchronization. It documents subroutines used to create and manipulate event synchronizers, and two facilities that use event synchronizers: timers and the InterServer Communications (ISC) facility.

Specifics of This Volume

Volume V describes three features of the PRIMOS® operating system: event synchronizers, timers, and the InterServer Communications facility for message exchange between processes.

- Part I of this book, consisting of Chapter 1, offers a general overview of event synchronization and a brief introduction to its use by two PRIMOS facilities: the Timers facility and the InterServer Communications (ISC) facility. It also provides programming information that is applicable to all programs that use event synchronization.
- Part II, consisting of Chapters 2 through 4, describes in detail how to create, use, destroy, and retrieve information about event synchronizers and event groups.
- Part III, consisting of Chapter 5, describes how to create, use, destroy, and retrieve information about timers.
- Part IV, consisting of Chapters 6 through 14, describes in detail the InterServer Communications (ISC) facility, which makes it possible for processes that are running simultaneously to exchange messages. These processes may be running on the same system or on two different systems connected by PRIMENET™. Message exchange is coordinated by using event synchronizers.

The following three indexes enable the reader to find information quickly:

- The Index of Subroutines by Function, a list of all of the subroutines in the five-volume series, grouped by the general types of function that they perform. Use this index to find out which subroutines perform a particular function, and then use the Index of Subroutines by Name to locate the individual subroutines.
- The Index of Subroutines by Name, an alphabetical list by name of all of the subroutines in the five-volume series. It lists the volume, chapter, and page number of the reference material for each subroutine.
- The Index, a list of the topics treated in this volume. Use this index to find out where in this volume a particular topic, process, or term is described.

Suggested References

The other volumes of the *Subroutines Reference* document set are the following:

Subroutines Reference I: Using Subroutines (DOC10080-2LA)

Subroutines Reference II: File System (DOC10081-1LA) and updates:

Update for Rev. 21.0 (UPD10081-11A)

Update for Rev. 22.0 (UPD10081-12A)

Subroutines Reference III: Operating System (DOC10082-1LA) and updates:

Update for Rev. 21.0 (UPD10082-11A)

Update for Rev. 22.0 (UPD10082-12A)

Subroutines Reference IV: Libraries and I/O (DOC10083-1LA) and updates:

Update for Rev. 21.0 (UPD10083-11A)

Update for Rev. 22.0 (UPD10083-12A)

The five volumes of the *Subroutines Reference* and their current updates can be ordered as a set as DCP10068.

The *PRIMOS User's Guide* (DOC4130-5LA) contains information on system use, directory structure, the condition mechanism, CPL files, ACLs, and how to load and execute files with external subroutines. Language programmers will also need the reference guides for their particular languages.

The following related Prime publications are also available and helpful to the programmer:

PRIMOS Commands Reference Guide (DOC3108-7LA)

Programmer's Guide to BIND and EPFs (DOC8691-1LA) and its update:

Update for Rev. 22.0 (UPD8691-11A)

Advanced Programmer's Guide, Volume 0: Introduction and Error Codes (DOC10066-3LA)

Advanced Programmer's Guide, Volume I: BIND and EPFs (DOC10055-1LA)

Advanced Programmer's Guide, Volume II: File System (DOC10056-2LA)

Advanced Programmer's Guide, Volume III: Command Environment (DOC10057-1LA)

System Administrator's Guide, Volume I: System Configuration (DOC10131-2LA)

System Administrator's Guide, Volume II: Communication Lines and Controllers (DOC10132-2LA).

System Administrator's Guide, Volume III: System Access (DOC10133-2LA).

User's Guide to Prime Network Services (DOC10115-1LA)

Operator's Guide to Prime Networks (DOC10114-1LA)

PRIMENET Planning and Configuration Guide (DOC7532-3LA)

System Architecture Reference Guide (DOC9473-3LA)

Prime Documentation Conventions

The following conventions are used in command formats, statement formats, and in examples throughout this document. Examples illustrate the uses of these commands and statements in typical applications.

<i>Convention</i>	<i>Explanation</i>	<i>Example</i>
UPPERCASE	In subroutine descriptions, words in uppercase indicate the names of commands, options, statements, or keywords. Enter them in either uppercase or lowercase.	SYN\$CREA
<i>italics</i>	Words in italics represent input or output parameters of subroutines, or other variables.	<i>sync_identifier</i>
Parentheses ()	In subroutine calls, you must enter parentheses exactly as shown.	call is\$ce(num, code)
Hyphen –	Wherever a hyphen appears as the first character of an option, it is a required part of that option.	ICE –SERVER
Boldface	New terms appear in boldface.	server

Part I, Overview

Overview of Event Synchronization

Introduction

This volume describes event synchronization and two facilities that use event synchronization: the Timers facility and the InterServer Communications (ISC) facility. **Event synchronization** is a feature of PRIMOS that makes it possible to coordinate the execution of a process with specific events exterior to the process. **Timers** use event synchronization to enable processes to make their own execution time-dependent; a running process sets a timer and is informed when that timer elapses. The InterServer Communications facility uses event synchronization to permit a running process to exchange messages with another concurrently running process. Timers and ISC are included in Rev. 22.0 and subsequent revisions of PRIMOS.

Event synchronization is made possible by **event synchronizers**. An event synchronizer is an indicator on which PRIMOS **posts a notice** for each occurrence of a particular event. This notice informs the process that the event has occurred.

While awaiting notification of an event, the process can either suspend or continue its execution. If the process has suspended its execution, notification of an event causes it to resume processing. If the process has not suspended its execution, PRIMOS posts a notice on a synchronizer associated with the process when the event occurs. This notice is available to the process, but does not interrupt its ongoing processing.

A process can create its own event synchronizers and associate each synchronizer with a specific event. When the event occurs, PRIMOS posts a notice on the associated synchronizer.

A process can group two or more synchronizers into an **event group**. When an event occurs, PRIMOS posts a notice on the associated synchronizer that is within the event group. The process can check the event group to determine whether a notice has been posted on any of its member synchronizers. If a notice has been posted on a member synchronizer, the process can also determine which synchronizer has been notified. If there are no notices on any of the event synchronizers within the group, the process can either continue its own operations, or wait on the event group. Waiting on an event group suspends the process' operations until a notice is posted on any one of the synchronizers in the group.

Note

The facilities described in this volume can synchronize the execution of two or more user processes running within a single server. For example, one process within the server can post a notice on a synchronizer to notify other user processes within the server that an event occurred. At Rev. 22.0, PRIMOS associates each process with its own server when the process is initialized; therefore, synchronization within a server is possible only for PRIMIX™ child processes. A PRIMIX child process is always a member of its parent's server.

Timers

Timers are a PRIMOS facility that provide for time-dependent process synchronization. Timers can post notices on event synchronizers after an elapsed interval, periodically at a fixed interval of time, or at a particular time on the system clock. User processes can request timers to post notices at any appropriate time or intervals of time.

InterServer Communications

InterServer Communications (ISC) is a PRIMOS facility that provides for message exchange between two concurrently running servers. ISC uses the term **server** to refer to a single process or a group of closely cooperating processes. PRIMOS places each terminal or phantom process in its own server when the process is initialized. A PRIMIX child process, however, is placed in its parent's server. A process can determine its own server name and can also make it possible for other processes to look up its server name. You must know the server name of another server before you can exchange messages with that server. Server name operations are described in Chapter 7.

ISC provides each server with several event synchronizers that are associated with specific events. It uses these synchronizers to inform each server of the operations performed by the other server. These operations include the establishment of a link between the servers and the exchange of messages. This permits a server to either suspend processing or perform other operations while awaiting a specific ISC event.

Message exchange occurs within a **session**. A session is a one-to-one link between two active servers. To create this link, one server requests the session and the other accepts the session request. Only two servers can participate in a session; each server can both send and receive multiple messages with the other server participating in the session. A server that is not a participant in the session cannot read messages or otherwise interact with the session. The session continues until either of the servers explicitly terminates the session, or a server is terminated (for example, by a user logout).

Each server can participate in multiple concurrent sessions. A session can involve two servers on the same system, or two servers on different PRIMENET nodes. ISC automatically handles the interface with PRIMENET, making an ISC session across PRIMENET nodes appear identical to ISC processing within the same node.

ISC supports several different kinds of messages:

- **Normal Messages** are messages sent during an established session. A Normal message can consist of two parts; a control part for short messages and a data part for longer messages. A Normal message consisting of both parts can be up to 32,886 characters in length. Information of any type can be sent in either the control part or the data part of a message.
- **Expedited Messages** are messages sent during an established session. Expedited messages are short messages that consist of a control part only. Expedited messages and Normal messages are placed in separate queues. This enables a server to read all of its Expedited messages first, then read its Normal messages.
- **Connect Messages** are messages sent while establishing or terminating a session. You can use these short messages rather than fully establishing a session if only a single brief message exchange is required.

Include Files

The SYSCOM directory contains a number of **include files** that a program must include if it calls subroutines for event synchronizers, timers, or ISC. The include files define key values that correspond to numeric values returned or expected by the subroutines. A program should reference the key values defined in these files, rather than their numeric equivalents. The include files provide templates for data structures used by some subroutines.

Event synchronizers, timers, and ISC each have their own include files. Different versions of each include file are provided for different programming languages. For example, the include file for event synchronizers exists in separate versions for FTN, PL/I, Pascal, C, and PMA.

Table 1-1 lists the include files in the SYSCOM directory, and the subroutines that use each file. In the include file names, replace *language* with the suffix for the language in which the program is written.

Table 1-1
SYSCOM Include Files

<i>Include File</i>	<i>Contents</i>
<code>SYNC_CODES.INS.<i>language</i></code>	Codes for SYN\$ subroutines
<code>TIMERMIK.INS.<i>language</i></code>	Codes for TMR\$ subroutines
<code>ISC_KEYS.INS.<i>language</i></code>	Codes for IS\$ subroutines
<code>SRS_CODES.INS.<i>language</i></code>	Codes for SRS\$ subroutines
<code>ISC_STRUCTURES.INS.<i>language</i></code>	Structure templates for IS\$ subroutines

These SYSCOM include files are described more fully in the chapters on synchronizers, timers, and ISC.

Retrieving Error Messages

Always use `ER$PRINT` and `ER$TEXT`, rather than `ERRPR$` or `ERTXT$`, to retrieve error messages for timers, event synchronizers, or ISC. `ER$PRINT` prints a message on a terminal, and `ER$TEXT` returns a message to a variable.

`ER$PRINT` and `ER$TEXT` retrieve `ERRD` error messages and subsystem-specific error messages. To retrieve error messages, `ER$PRINT` and `ER$TEXT` first look for a specified message file in the `SYSOVL` directory. If the specified message file does not exist in `SYSOVL`, `ER$PRINT` and `ER$TEXT` retrieve the error message from PRIMOS internal tables, which are in English.

Programs that call `ER$PRINT` and `ER$TEXT` must include the SYSCOM include file `ErrorMsgHdlr.INS.language`, where *language* is the suffix for the language of the calling program.

For more information about `ER$PRINT` and `ER$TEXT`, see the *Subroutines Reference III: Operating System*.

Part II, Event Synchronizers and Event Groups

Event Synchronizers

Introduction

Event synchronizers make it possible to synchronize user processes with facilities of the PRIMOS operating system. These facilities include timers and the InterServer Communications (ISC) subsystem. Timers and ISC use event synchronizers to signal to user processes that certain events occurred. The user processes can base their subsequent actions on the occurrence or nonoccurrence of these events. See Part III for information about timers, and Part IV for information about ISC.

Event synchronizers can be used to synchronize PRIMOS facilities and user processes as follows:

- When a PRIMOS facility wants to indicate to a user process that an event occurred, it can **post a notice** on an event synchronizer.
- If a process needs to know whether a particular event occurred before it takes further action, the process can retrieve a notice of an event on the event synchronizer, if one was posted.
- If a process has nothing to do until a particular event occurs, the process can wait until there is a notice of an event on a particular synchronizer. The process can wait for the event indefinitely, or it can set a limit to the length of time that it will wait.

An event synchronizer is private to a single server and cannot be accessed by other servers.

Some user processes may need to know whether any one of a group of possible events occurred. Such a process can access a group of event synchronizers, known as an **event group**, to determine whether there is a notice of an event on any of the synchronizers in the group. The process can wait for notices to be posted on event groups, either indefinitely or for a limited length of time.

Note

Synchronizers can also be used to synchronize the activities of processes within the same server. The only multiprocess servers currently supported are servers running PRIMIX where child processes are running.

Processes must call subroutines to create, use, and destroy event synchronizers and event groups. This chapter describes the subroutines for event synchronizers. Chapter 3 describes subroutines for event groups.

Event Synchronizer Subroutines

Table 2-1 lists the event synchronizer subroutines. See Table 3-1 for a list of the event group subroutines. Programs written in FTN must use the six-character subroutine names listed.

Table 2-1
Event Synchronizer Subroutines

<i>Name</i>	<i>Function</i>
SYN\$CREA SYN\$CR	Creates a synchronizer
SYN\$POST SYN\$PO	Posts a notice on a synchronizer
SYN\$WAIT SYN\$WT	Waits on a synchronizer
SYN\$TMWT SYN\$TW	Performs a timed wait on a synchronizer
SYN\$RTRV SYN\$RV	Retrieves a notice from a synchronizer
SYN\$DEST SYN\$DE	Destroys a synchronizer

The following paragraphs summarize the functions that synchronizer subroutines perform.

Creating Synchronizers

To create an event synchronizer, a process calls the subroutine SYN\$CREA. SYN\$CREA assigns an identifier to the synchronizer. This identifier becomes an input parameter to other subroutines that access the synchronizer. The process that calls SYN\$CREA specifies the number of notices that are to be on the synchronizer when it is created. This number must be greater than or equal to 0.

Posting Notices on Synchronizers

When a process in a multiprocess server needs to indicate to other processes within that server that a particular event occurred, it can call the subroutine SYN\$POST. SYN\$POST posts a notice on a synchronizer within the same server as the process that calls SYN\$POST. If no processes are waiting on or performing a timed wait on the synchronizer (see below), posting the notice increases the synchronizer's notice count by 1. If processes are waiting, posting the notice causes one process to resume execution. At PRIMOS Rev. 22.0, synchronization within a server is possible only for PRIMIX child processes.

Waiting on Synchronizers

When a process needs to wait until a particular event occurs before taking further action, it can call the subroutine SYN\$WAIT to wait on a particular event synchronizer. If there are no notices on the event synchronizer, the process that calls SYN\$WAIT suspends its own operation until another process posts a notice on the synchronizer. If there are notices on the synchronizer, the process that calls SYN\$WAIT continues its operations. When a process attempts to wait on a synchronizer that has a notice count of 1 or more, it decreases the synchronizer's notice count by 1 and continues to run. Any number of processes within a server can wait on a synchronizer.

Performing a Timed Wait on Synchronizers

A process that needs to wait on a synchronizer may need to set a limit to the amount of time that it will wait. Such a process can call the subroutine SYN\$TMWT to perform a timed wait on the synchronizer. A process that performs a timed wait on a synchronizer suspends its own operation until another process posts a notice on the synchronizer or until an interval of time passes. The process that calls SYN\$TMWT specifies the length of this interval. When a process attempts to perform a timed wait on a synchronizer that has a notice count of 1 or more, it decreases the notice count by 1 and continues to run. Any number of processes within a server can perform a timed wait on a synchronizer.

Retrieving a Notice from an Event Synchronizer

A process that wants to find out whether an event occurred, but does not want to wait on a synchronizer, can call the subroutine SYN\$RTRV to retrieve a notice from a synchronizer. If the synchronizer's notice count is 1 or more, the retrieving process decreases the count by 1. If the synchronizer has no outstanding notices, SYN\$RTRV does nothing to the synchronizer and indicates to the calling process that there are no notices on the synchronizer.

Destroying Event Synchronizers

When a synchronizer is no longer needed, a user process can call the subroutine SYN\$DEST to destroy the synchronizer. Before destroying an event synchronizer, be sure to cancel any timers that are to post notices on it. If an event synchronizer is destroyed while a timer is set to post notices on it, notices could eventually be posted inappropriately on another event synchronizer.

The INITIALIZE_COMMAND_ENVIRONMENT (ICE) command with the `-SERVER` option deallocates all event synchronizers and timers belonging to a user process. The ICE command with the `-SERVER` option also logs out any child processes belonging to the user process. The ICE command without the `-SERVER` option does not affect event synchronizers or timers. See the *PRIMOS Commands Reference Guide* for information about the ICE command.

Include Files for Synchronizers

The SYSCOM directory contains a number of **include files** for programs that invoke synchronizer subroutines. The include files define certain key values that correspond to numeric values returned by synchronizer subroutines. These key values serve as status codes and other arguments. The include files also define templates for data structures used by synchronizer subroutines.

There are separate include files for different programming languages. A program that calls synchronizer subroutines should include the synchronizer include file for its language and use the key values defined in the file, rather than their numeric equivalents.

The names of the include files for synchronizers are of the form

`SYNC_CODES.INS.language`

where *language* is an abbreviation specifying the programming language. For example, the synchronizer include file for PL/I is

`SYNC_CODES.INS.PL1`

Before writing a program that invokes synchronizer subroutines, consult the include file for the program's language to determine the correct key values for that language.

Note

Because of the name length limits of the FTN language, the key values in the FTN key file are substantially different from those in other language files. Appendix C lists the FTN keys used as returned codes.

Subroutine Descriptions

This section contains descriptions of the subroutines that create, use, and destroy event synchronizers. In the subroutine descriptions, the data type declarations (DCL statements), status codes, and *what_happened* codes are in PL/I format.

SYN\$CREA

SYN\$CR

Creates an event synchronizer.

Usage

```
DCL SYN$CREA ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(15));
```

```
CALL SYN$CREA (count, sync_identifier, code);
```

Parameters

count

INPUT. The number of notices that the event synchronizer is to have when it is created. This number must be greater than or equal to 0.

sync_identifier

OUTPUT. The identifier of the synchronizer that SYN\$CREA created. This value can be NullSyncNum or a value greater than 0. The value of NullSyncNum is defined in the synchronizer include file.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$CREA was completed without error. Any other code indicates that SYN\$CREA failed to create a synchronizer and assigned the value NullSyncNum to *sync_identifier*.

SYN_SC\$NoResources

The system is not able to allocate resources for an event synchronizer. The process that calls SYN\$CREA may be able to solve this problem by releasing kernel resources that it is not currently using, such as synchronizers, groups, or timers, and then reexecuting the call.

SYN_SC\$InvNoticeCount

The value specified for the parameter *count* is invalid, because it is negative.

SYN_SC\$MaxSyncsAlloc

The maximum number of synchronizers or groups allowed for a server has already been allocated and no more can be created. To solve this problem, destroy an event synchronizer or an event group on the server.

SYN\$CREA

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$CREA creates a new event synchronizer and returns a value to *sync_identifier* to represent it. To access the new synchronizer, a process must reference the value that SYN\$CREA returns to *sync_identifier*.

The process that calls SYN\$CREA specifies the initial notice count of the synchronizer. This count represents the number of times that a particular event has already occurred at the time when the synchronizer is created. In most cases, a process should specify an initial notice count of 0 in its call to SYN\$CREA.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$POST

SYN\$PO

Posts a notice on an event synchronizer.

Usage

```
DCL SYN$POST ENTRY (FIXED BIN(15), FIXED BIN(15));
```

```
CALL SYN$POST (sync_identifier, code);
```

Parameters

sync_identifier

INPUT. The identifier of the event synchronizer on which SYN\$POST is to post a notice.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$POST was completed without error.

SYN_SC\$InvSyncNum

sync_identifier does not specify a valid event synchronizer. Either the specified synchronizer does not exist or the maximum number of pending notices for that synchronizer has been exceeded. See Appendix D.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$POST posts a notice on the event synchronizer specified by *sync_identifier*. SYN\$POST can post notices on synchronizers that stand alone or on synchronizers that are in event groups. See Chapter 3 of this volume for more information about event groups.

If there are no processes waiting on the event synchronizer specified by *sync_identifier*, SYN\$POST increases the synchronizer's notice count by 1. If at least one process is waiting on or performing a timed wait on this synchronizer, SYN\$POST allows a process that is waiting on the synchronizer to run but does not change the synchronizer's notice count.

SYN\$POST

Note

Currently, SYN\$POST is of use primarily in servers running under PRIMIX where child processes are running. SYN\$POST makes it possible to synchronize the activities of the processes within such a server.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$WAIT

SYN\$WT

Waits on an event synchronizer that is not in an event group.

Usage

```
DCL SYN$WAIT ENTRY (FIXED BIN(15), FIXED BIN(15));
```

```
CALL SYN$WAIT (sync_identifier, code);
```

Parameters

sync_identifier

INPUT. The identifier of the event synchronizer on which the process is to wait.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$WAIT was completed without error.

SYN_SC\$InvSyncNum

sync_identifier does not specify a valid event synchronizer.

SYN_SC\$SyncInGroup

The synchronizer specified by *sync_identifier* is currently part of an event group. SYN\$WAIT did nothing because it cannot wait on a synchronizer that is part of an event group.

SYN_SC\$NoResources

Resources are not available to enable the process to wait on the synchronizer specified by *sync_identifier*.

SYN_SC\$WaitHasAborted

The wait was ended by a software interrupt. When this happens, the handler of the software interrupt usually reexecutes the call to the subroutine; in this case, the status SYN_SC\$WaitHasAborted is not reported. In some cases, however, the handler may not reexecute the subroutine, allowing SYN_SC\$WaitHasAborted to be reported.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

SYN\$WAIT

Discussion

SYN\$WAIT checks the notice count of a synchronizer and bases its subsequent actions on whether there are notices on the synchronizer.

If there are notices on the synchronizer when the process calls SYN\$WAIT, SYN\$WAIT decreases the synchronizer's notice count by 1 and allows the process that called SYN\$WAIT to continue to run.

If there are no notices on the synchronizer, SYN\$WAIT causes the calling process to wait until a notice is posted on the synchronizer. When a notice is posted on the synchronizer, SYN\$WAIT allows the calling process to run again.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$TMWT

SYN\$TW

Performs a timed wait on an event synchronizer that is not in an event group.

Usage

```
DCL SYN$TMWT ENTRY (FIXED BIN(15), FIXED BIN(31), FIXED BIN(15),  
                    FIXED BIN(15));
```

```
CALL SYN$TMWT (sync_identifier, wait_time, what_happened, code);
```

Parameters

sync_identifier

INPUT. The identifier of the event synchronizer on which the process that calls SYN\$TMWT is to perform a timed wait.

wait_time

INPUT. The greatest amount of time that the calling process will wait for a notice to be posted on the event synchronizer. Specify *wait_time* in milliseconds. Currently, the system rounds the *wait_time* value down to the next lower multiple of 100 milliseconds. *wait_time* cannot be 0.

what_happened

OUTPUT. SYN\$TMWT returns this code only when it returns the code SYN_SC\$OK. *what_happened* indicates why SYN\$TMWT is enabling the process to run again. The possible codes are:

SYN_WHC\$Notice

The synchronizer had notices when SYN\$TMWT was called, or a notice was posted on the synchronizer before *wait_time* elapsed.

SYN_WHC\$TimeOut

There were no notices on the synchronizer when SYN\$TMWT was called and the amount of time specified by *wait_time* passed without a notice being posted on the event synchronizer.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$TMWT was completed without error.

SYN\$TMWT

SYN_SC\$InvSyncNum

sync_identifier does not specify a valid event synchronizer.

SYN_SC\$InvTimeInt

The value of *wait_time* is invalid because it is not greater than 0.

SYN_SC\$SyncInGroup

The synchronizer whose number is specified by *sync_identifier* is in an event group. SYN\$TMWT did nothing because it cannot wait on a synchronizer that is part of an event group.

SYN_SC\$NoResources

Resources are not available to enable the process to perform a timed wait on the synchronizer specified by *sync_identifier*.

SYN_SC\$WaitHasAborted

The wait was ended by a software interrupt. When this happens, the handler of the software interrupt usually reexecutes the call to the subroutine; in this case, the status SYN_SC\$WaitHasAborted is not reported. In some cases, however, the handler may not reexecute the call to the subroutine, allowing SYN_SC\$WaitHasAborted to be reported.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$TMWT checks the notice count of a synchronizer and bases its subsequent actions on whether there are notices on the synchronizer.

If there is at least one notice on the synchronizer when a process calls SYN\$TMWT, SYN\$TMWT decreases the synchronizer's notice count by 1, returns SYN_WHC\$Notice to *what_happened*, and allows the process to continue to run.

If there are no notices on a synchronizer when a process calls SYN\$TMWT, SYN\$TMWT causes the process that calls it to wait until a notice is posted on the synchronizer, or until a specified amount of time passes. The process that calls SYN\$TMWT specifies this amount of time. When a notice is posted on the synchronizer or the interval of time elapses, SYN\$TMWT enables the process that called it to run again.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$RTRV

SYN\$RV

Retrieves a notice from an event synchronizer that is not in an event group.

Usage

```
DCL SYN$RTRV ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(15));
```

```
CALL SYN$RTRV (sync_identifier, what_happened, code);
```

Parameters

sync_identifier

INPUT. The identifier of the event synchronizer from which SYN\$RTRV is to retrieve a notice.

what_happened

OUTPUT. A code that tells the caller of SYN\$RTRV whether or not SYN\$RTRV retrieved a notice from the synchronizer. The possible codes are:

SYN_WHC\$Notice

SYN\$RTRV retrieved a notice from the synchronizer.

SYN_WHC\$NoNotice

There were no notices on the synchronizer for SYN\$RTRV to retrieve.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$RTRV was completed without error.

SYN_SC\$InvSyncNum

sync_identifier does not specify a valid event synchronizer.

SYN_SC\$SyncInGroup

SYN\$RTRV cannot retrieve a notice from the synchronizer specified by *sync_identifier* because the synchronizer is a part of an event group.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

SYN\$RTRV

Discussion

A process should call SYN\$RTRV when it wants to note whether an event occurred before taking some action, but does not want to wait. SYN\$RTRV never causes the process that calls it to wait.

If there is a notice on the synchronizer, SYN\$RTRV decreases the synchronizer's notice count by 1. If there is no notice on the synchronizer, SYN\$RTRV does nothing to the synchronizer.

SYN\$RTRV returns to *what_happened* a code indicating whether or not it retrieved a notice.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$DEST

SYN\$DE

Destroys an event synchronizer.

Usage

```
DCL SYN$DEST ENTRY (FIXED BIN(15), FIXED BIN(15));
```

```
CALL SYN$DEST (sync_identifier, code);
```

Parameters

sync_identifier

INPUT. The identifier of the event synchronizer that SYN\$DEST is to destroy.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$DEST was completed without error.

SYN_SC\$InvSyncNum

sync_identifier does not specify a valid event synchronizer.

SYN_SC\$SyncInGroup

The synchronizer specified by *sync_identifier* cannot be destroyed because it is in an event group. Before the process can destroy the synchronizer, it must call the subroutine SYN\$REMV to remove the synchronizer from the event group. See Chapter 3 of this volume for information about event groups.

SYN_SC\$SyncHasWaiter

The synchronizer specified by *sync_identifier* cannot be destroyed because one or more processes are waiting on it.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

SYN\$DEST

Discussion

SYN\$DEST destroys an event synchronizer, but does not destroy references to this synchronizer in the database of the process that calls SYN\$DEST. The process that calls SYN\$DEST should destroy these references. SYN\$CREA can reassign the identifier of a destroyed synchronizer to a new synchronizer.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

Event Groups

Introduction

An event group is a group of event synchronizers. The event group enables a user process to determine whether any one of a group of events has taken place. PRIMOS facilities, such as timers and ISC, can post notices on any of the synchronizers in an event group, as well as on synchronizers that are not in a group. Each synchronizer in the event group can represent a different event. User processes can use event groups in the following ways:

- If a user process needs to know whether any one of a group of events occurred before it takes further action, the process can access the event group to determine whether the notice of an event has been posted on any of the synchronizers in the group.
- If a user process has nothing to do until one of the events in the group occurs, the process can wait until a notice is posted on one of the synchronizers in the event group. The process can wait indefinitely, or it can set a limit to the amount of time that it will wait.

This chapter describes the system subroutines that user processes can call to create, use, and destroy event groups.

Event Group Subroutines

Table 3-1 lists the subroutines that create, use, and destroy event groups. Programs written in FTN must use the six-character subroutine names listed.

Table 3-1
Subroutines Dealing with Event Groups

<i>Name</i>	<i>Function</i>
SYN\$GCRE SYN\$GC	Creates an event group
SYN\$MVTO SYN\$MV	Moves a synchronizer into an event group
SYN\$REMV SYN\$RM	Removes a synchronizer from an event group
SYN\$GWT SYN\$GW	Causes a process to wait on an event group
SYN\$GTWT SYN\$GT	Causes a process to perform a timed wait on an event group
SYN\$GRTR SYN\$GR	Retrieves a notice from an event group
SYN\$GDST SYN\$GD	Destroys an event group

The following sections summarize the functions that event group subroutines perform.

Creating Event Groups

To create an event group, a user process calls the subroutine SYN\$GCRE. SYN\$GCRE assigns an identifier to the group. This identifier becomes an input parameter to other subroutines that access the group. SYN\$GCRE creates the group empty, with no synchronizers in it. Before a process can use an event group that it created, the process must move event synchronizers into the event group (see SYN\$MVTO below).

SYN\$GCRE also specifies the number of priority levels that the group can use. The priority levels of an event group determine the order in which the group returns notices of events to processes.

Except when a process specifies a particular priority level (see SYN\$GRTR below), a notice is returned from a synchronizer at the highest priority level where there are synchronizers with notices. Notices are returned from lower priorities only when there are no notices at higher priorities.

The process that creates the event group specifies the number of different priority levels that the group can have. The priority levels are numbered, with 1 representing the highest priority.

An event group is private to a single server and cannot be accessed by other servers.

Moving Synchronizers Into Event Groups

To be more useful than an individual synchronizer, an event group must have at least two event synchronizers in it. To move an event synchronizer into an event group, a process calls the subroutine SYN\$MVTO. If a synchronizer is already part of a group, SYN\$MVTO removes the synchronizer from this current group before moving it into a different group.

In its call to SYN\$MVTO, a process specifies a **For Client Use** field for the synchronizer. The For Client Use field is returned to a process when the process receives a notice of an event from the synchronizer. The For Client Use field can enable the process to identify the particular event associated with the synchronizer. For example, the For Client Use field might contain characters that identify the event for which a notice was posted, or a pointer to a record that contains information about the event.

In its call to SYN\$MVTO, a process also specifies the **priority level** into which SYN\$MVTO is to move the synchronizer. A user process should move synchronizers into different priority levels based on the order in which it wants to respond to different events.

Removing an Event Synchronizer from an Event Group

When an event synchronizer is no longer needed in an event group, a process can call the subroutine SYN\$REMOV to remove it from the group. When the process removes the synchronizer, any notices on that synchronizer remain with it. After a synchronizer has been removed from a group, it can be accessed individually.

Waiting on Event Groups

When a process needs to wait until a notice of an event is posted on one of the synchronizers in a group, it can call the subroutine SYN\$GWT to wait on the group.

If there is a notice on any of the synchronizers within the group when the process calls SYN\$GWT, SYN\$GWT decreases by 1 the notice count of a synchronizer at the highest priority level where there are synchronizers with notices, returns to the waiting process the identifier and For Client Use field of this synchronizer, and allows this process to continue to run.

If there are no notices on any of the synchronizers in the group, the process that calls SYN\$GWT suspends its own operations until another process posts a notice on one of the synchronizers in the group. When a process posts a notice on a synchronizer within the group, SYN\$GWT returns the identifier and the For Client Use field of this synchronizer to the process that called SYN\$GWT, and allows this process to run again.

Performing a Timed Wait on an Event Group

A process can set a limit to the amount of time that it will wait for a notice to be posted on an event group by calling the subroutine SYN\$GTWT to perform a **timed wait** on the group. The calling process specifies the maximum amount of time that it will wait.

If there is a notice on any of the synchronizers in the group when the process calls SYN\$GTWT, SYN\$GTWT decreases by 1 the notice count of a synchronizer at the highest priority level where there are synchronizers with notices, returns the identifier and the For Client Use field of this synchronizer to the process that called SYN\$GTWT, tells the calling process that there was a notice on the group, and allows the calling process to continue to run.

If there are no notices on any of the synchronizers in the group when a process calls SYN\$GTWT, the process waits until a notice is posted on a synchronizer within the event group, or until an interval of time passes. The process that calls SYN\$GTWT specifies this interval of time. If a notice is posted on one of the synchronizers in the group before the interval of time elapses, SYN\$GTWT returns the identifier and For Client Use field of this synchronizer to the calling process, allows the calling process to run again, and indicates to the process that a notice was posted. If the interval of time passes without a notice being posted on a synchronizer in the group, SYN\$GTWT allows the calling process to run again and indicates to the process that the interval of time elapsed.

Retrieving Notices from Event Groups

A process that wants to find out whether there is a notice on any of the synchronizers in the group but does not under any circumstances want to wait can call the subroutine SYN\$GRTR. SYN\$GRTR indicates to the calling process whether there are notices on the group, and returns a notice to the process if one has been posted. SYN\$GRTR never causes the process that calls it to wait.

SYN\$GRTR can retrieve notices from the entire group, or only from synchronizers at a certain priority level. The latter capability is useful when a process wants to retrieve a notice of events at a certain priority level.

Destroying Event Groups

When an event group is no longer needed, a user process can call the subroutine SYN\$GDST to destroy the group. When a process destroys a group, the synchronizers within the group remain as individual synchronizers. The individual synchronizers retain the notices that they had when the event group was destroyed.

Subroutine Descriptions

This section contains descriptions of the subroutines that create, use, and destroy event groups. In the subroutine descriptions, the data type declarations and the possible values for the parameters *code* and *what_happened* are in PL/I format.

SYN\$GCRE

SYN\$GC

Creates an event group.

Usage

```
DCL SYN$GCRE ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(15));
```

```
CALL SYN$GCRE (priority_levels, group_identifier, code);
```

Parameters

priority_levels

INPUT. The number of different priority levels within this event group. This number can range from 1 to MaxPriorities, with 1 representing the highest priority. The value of MaxPriorities is defined in the include file for synchronizers.

group_identifier

OUTPUT. The identifier of the event group that SYN\$GCRE created. If the status code is not SYN_SC\$OK, SYN\$GCRE returns NullSyncNum to *group_identifier*. The value of NullSyncNum is defined in the include file for synchronizers.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$GCRE was completed without error. Any code other than SYN_SC\$OK indicates that SYN\$GCRE failed to create an event group and assigned the value NullSyncNum to *group_identifier*.

SYN_SC\$NoResources

The system is not able to allocate resources for a new event group. The calling process may be able to solve this problem by releasing resources that it is not currently using, such as other synchronizers or timers.

SYN_SC\$InvPriority

The value specified by *priority_levels* is not within the valid range for priorities, which is from 1 to MaxPriorities.

SYN_SC\$MaxSyncsAlloc

The maximum number of synchronizers or groups allowed for this server has already been created on this server. To enable the calling process to create an event group, destroy an event synchronizer or an event group on the server.

SYN\$GCRE

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$GCRE creates an event group and returns the group's identifier to the argument *group_identifier*. The identifier is used as an input parameter to other subroutines that access the event group. The process that calls SYN\$GCRE specifies in *priority_levels* the number of different priority levels that the group is to have.

SYN\$GCRE creates an event group with no synchronizers in it. Processes must call SYN\$MVTO to move synchronizers into event groups.

Note

The use of priority levels can lower the speed at which groups return notices through SYN\$GRTR, SYN\$GWT, and SYN\$GTWT. The lower the priority level of a synchronizer, the more processing time is required to return a notice from that synchronizer.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$MVTO

SYN\$MV

Moves an event synchronizer into an event group.

Usage

```
DCL SYN$MVTO ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(15),
                    (3) FIXED BIN(15), FIXED BIN(15));
```

```
CALL SYN$MVTO (group_identifier, sync_identifier, priority_level,
               for_client_use, code);
```

Parameters

group_identifier

INPUT. The identifier of the event group into which the event synchronizer specified by *sync_identifier* is to be moved.

sync_identifier

INPUT. The identifier of the event synchronizer to be moved into the event group specified by *group_identifier*.

priority_level

INPUT. The priority level at which SYN\$MVTO is to move the event synchronizer into the event group. This value can range from 1, the highest priority level, to the value specified for *priority_levels* in the call to SYN\$GCRE that created the event group.

for_client_use

INPUT. A value that is returned to a process when a notice is returned to it from synchronizer *sync_identifier*. The calling process specifies the *for_client_use* value. PRIMOS does not use or alter the *for_client_use* value in any way. See the descriptions of the subroutines SYN\$GWT, SYN\$GTWT, and SYN\$GRTR for information about when subroutines return the For Client Use field to a process.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$MVTO was completed without error.

SYN_SC\$InvGroupNum

group_identifier does not specify a valid event group.

SYN\$MVTO**SYN_SC\$InvSyncNum**

sync_identifier does not specify a valid event synchronizer.

SYN_SC\$SyncHasWaiter

The event synchronizer specified by *sync_identifier* cannot be moved into an event group because a process is currently waiting on the synchronizer.

SYN_SC\$InvPriority

The value of *priority_level* is outside the range of legal priority levels for the event group specified by *group_identifier*. The range of legal priority levels for the event group is set by the process that calls SYN\$GCRE to create the group.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$MVTO moves an event synchronizer into an existing event group. If the synchronizer has notices, it retains them when SYN\$MVTO moves it into the event group. Thus, executing SYN\$MVTO could enable one or more processes that are waiting or performing a timed wait on the event group to run again. If the synchronizer is already in an event group when SYN\$MVTO is executed, SYN\$MVTO removes the synchronizer from its original group, even if processes are waiting on this group. A synchronizer cannot be in more than one event group at a time.

The process that calls SYN\$MVTO specifies the priority level into which SYN\$MVTO moves the synchronizer. More than one event synchronizer can be at the same priority level. The calling process should move synchronizers into priority levels based on the order in which it would like to process events, given that they occur at the same time.

The For Client Use field provides a way to associate a name or other information with each synchronizer in a group. For example, the For Client Use field can serve as a pointer to a structure that contains information about a particular event. Because a process can receive information about an event from the For Client Use field, it does not have to maintain a table to associate synchronizer identifiers with particular events.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$REMV

SYN\$RM

Removes an event synchronizer from an event group.

Usage

```
DCL SYN$REMV ENTRY (FIXED BIN(15), FIXED BIN(15));
```

```
CALL SYN$REMV (sync_identifier, code);
```

Parameters

sync_identifier

INPUT. The identifier of the event synchronizer that SYN\$REMV is to remove from an event group.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$REMV was completed without error.

SYN_SC\$InvSyncNum

sync_identifier does not specify a valid event synchronizer.

SYN_SC\$SyncNotInGroup

The synchronizer specified by *sync_identifier* is not in an event group. The calling process can ignore this error if all that matters is that the synchronizer not be in an event group.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$REMV removes an event synchronizer from an event group, making it possible for processes to access the synchronizer individually. When SYN\$REMV removes the synchronizer, any notices on that synchronizer are also removed from the group and remain with the synchronizer.

Note

In a multiprocess server, removing the last synchronizer from a group could leave one or more processes waiting on a now empty group.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$GWT

SYN\$GW

Waits on an event group.

Usage

```
DCL SYN$GWT ENTRY (FIXED BIN(15), FIXED BIN(15), (3) FIXED BIN(15),  
                  FIXED BIN(15));
```

```
CALL SYN$GWT (group_identifier, sync_identifier, for_client_use, code);
```

Parameters

group_identifier

INPUT. The identifier of the event group on which the process that calls SYN\$GWT is to wait.

sync_identifier

OUTPUT. The identifier of the event synchronizer from which SYN\$GWT is returning a notice. This is a synchronizer at the highest priority level where any synchronizers have notices.

for_client_use

OUTPUT. The For Client Use field associated with the synchronizer whose number SYN\$GWT returns to *sync_identifier*. The value of *for_client_use* is specified by the process that called SYN\$MVTO to move the synchronizer into the group.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$GWT was completed without error.

SYN_SC\$InvGroupNum

group_identifier does not specify a valid event group.

SYN_SC\$WaitHasAborted

The calling process was waiting on the event group, but a software interrupt aborted its wait. When this happens, the handler of the software interrupt usually reexecutes the call to the subroutine; in this case, the status SYN_SC\$WaitHasAborted is not seen by the calling program. In some cases, however, the handler may not reexecute the subroutine, allowing SYN_SC\$WaitHasAborted to be seen. When this error occurs, SYN\$GWT returns NullSyncNum to *sync_identifier* and NullFCU to *for_client_use*.

SYN\$GWT

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$GWT checks whether there are notices on any synchronizers within an event group.

If there are notices, SYN\$GWT decreases by 1 the notice count of a synchronizer at the highest priority level where at least one synchronizer has a notice, and allows the calling process to continue to run. SYN\$GWT also returns to the calling process the identifier of the synchronizer whose notice it is returning and the For Client Use field associated with this synchronizer.

If there are no notices on the group, SYN\$GWT causes the calling process to wait until a notice is posted on a synchronizer in the group. When a notice is posted, SYN\$GWT allows the process to resume execution. Note that SYN\$GWT causes the calling process to wait on an entire group. SYN\$GWT cannot cause a process to wait for a notice to be posted on an individual synchronizer or priority level.

If the status code is not SYN_SC\$OK, SYN\$GWT returns NullSyncNum to *sync_identifier* and NullFCU to *for_client_use*.

A process would call SYN\$GWT when it has nothing to do and wants to wait for a notice to be posted on one of the synchronizers within a group.

Note

A server should never attempt to wait on an event group with no synchronizers in it, unless there is more than one process in the server. If a process waits on an event group with no synchronizers, it will wait until another process moves synchronizers into the group and posts notices on them.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$GTWT

SYN\$GT

Performs a timed wait on an event group.

Usage

DCL SYN\$GTWT ENTRY (FIXED BIN(15), FIXED BIN(31), FIXED BIN(15),
FIXED BIN(15), (3) FIXED BIN(15), FIXED BIN(15));

CALL SYN\$GTWT (*group_identifier*, *wait_time*, *what_happened*,
sync_identifier, *for_client_use*, *code*);

Parameters

group_identifier

INPUT. The identifier of the event group on which SYN\$GTWT is to perform a timed wait.

wait_time

INPUT. The longest amount of time that the calling process is to wait for a notice to be posted on one of the synchronizers in the event group. Specify *wait_time* in milliseconds. Currently, the system rounds the *wait_time* value down to the next lower multiple of 100 milliseconds.

what_happened

OUTPUT. SYN\$GTWT returns this code only when it returns the code SYN_SC\$OK. It tells the calling process whether a notice was returned or *wait_time* elapsed. The possible codes are:

SYN_WHC\$Notice

The group had notices when SYN\$GTWT was called, or a notice was posted on the group before *wait_time* elapsed.

SYN_WHC\$TimeOut

There were no notices on the group when SYN\$GTWT was called and the amount of time specified by *wait_time* elapsed without a notice being posted on the group.

sync_identifier

OUTPUT. The identifier of the event synchronizer from which a notice is being returned.

for_client_use

OUTPUT. The For Client Use field associated with the event synchronizer whose number is returned to *sync_identifier*. The value of *for_client_use* is specified by the process that called SYN\$MVTO to move the synchronizer into the group.

SYN\$GTWT*code*

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$GTWT was completed without error.

SYN_SC\$InvGroupNum

group_identifier does not specify a valid event group.

SYN_SC\$InvTimeInt

The value of *wait_time* is not valid. *wait_time* must not be 0. Any other value is treated as positive.

SYN_SC\$WaitHasAborted

The wait was ended by a software interrupt. When this happens, the handler of the software interrupt usually causes the call to the subroutine to be reexecuted; in this case, the status SYN_SC\$WaitHasAborted is not seen by the calling program. In some cases, however, the handler may not reexecute the subroutine, allowing SYN_SC\$WaitHasAborted to be seen. When this error occurs, SYN\$GTWT returns NullSyncNum for *sync_identifier* and NullFCU to *for_client_use*.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$GTWT checks whether there are notices on any synchronizer within the event group.

If there are notices when the process calls SYN\$GTWT, SYN\$GTWT decreases by 1 the notice count of a synchronizer at the highest priority level where at least one synchronizer has notices and allows the calling process to continue to run. SYN\$GTWT returns to the calling process the identifier of the synchronizer whose notice count it decreased. SYN\$GTWT also returns the For Client Use field associated with this synchronizer.

If there are no notices, SYN\$GTWT causes the calling process to wait until a notice is posted on one of the synchronizers within the event group, or until the amount of time specified by *wait_time* passes. When a notice is posted or the time interval elapses, SYN\$GTWT allows the process to resume execution. Note that SYN\$GTWT can only cause the calling process to wait on the entire group. SYN\$GTWT cannot cause a process to wait for a notice to be posted on a particular synchronizer in a group or at a particular priority level.

A process would call this routine when it has nothing to do and wants to wait for a notice to be posted on one of the synchronizers in the group, but also wants to do something else if no notices are posted within a certain amount of time.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$GRTR

SYN\$GR

Retrieves a notice from an event group.

Usage

```
DCL SYN$GRTR ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(15),  
                    FIXED BIN(15), (3) FIXED BIN(15), FIXED BIN(15));
```

```
CALL SYN$GRTR (group_identifier, priority_level, what_happened,  
               sync_identifier, for_client_use, code);
```

Parameters

group_identifier

INPUT. The identifier of the event group from which SYN\$GRTR is to retrieve a notice.

priority_level

INPUT. The priority level at which SYN\$GRTR is to retrieve notices. *priority_level* must be either AllPriorities or within the range of valid priority levels for this event group. Specify AllPriorities to retrieve a notice from a synchronizer at the highest priority level where there are synchronizers with notices. Specify a particular priority level to retrieve a notice only from a synchronizer at that level.

what_happened

OUTPUT. Indicates whether SYN\$GRTR retrieved a notice. SYN\$GRTR returns a value to *what_happened* when it returns SYN_SC\$OK to *code*. The possible values are:

SYN_WHC\$Notice

SYN\$GRTR retrieved a notice.

SYN_WHC\$NoNotice

SYN\$GRTR did not retrieve a notice.

sync_identifier

OUTPUT. The identifier of the event synchronizer from which SYN\$GRTR retrieved a notice.

for_client_use

OUTPUT. The For Client Use field associated with the event synchronizer from which SYN\$GRTR retrieved a notice. See the description of SYN\$MVTO for information about how to specify a For Client Use field.

SYN\$GRTR*code*

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$GRTR was completed without error.

If the status code is not SYN_SC\$OK, SYN\$GRTR returns NullSyncNum to *sync_identifier* and NullFCU to *for_client_use*.

SYN_SC\$InvGroupNum

group_identifier does not specify a valid event group.

SYN_SC\$InvPriority

The value of *priority_level* is not AllPriorities and is not within the range of priority levels that are valid for event group *group_identifier*. SYN\$GCRE specifies the range of valid priority levels for a group when it creates the group.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$GRTR retrieves a notice from an event group or from a specific priority level of an event group, if such a notice is available. SYN\$GRTR never causes the process that calls it to wait.

When SYN\$GRTR retrieves a notice, it always does the following:

- Returns the identifier of the synchronizer with the notice to *sync_identifier*
- Returns the For Client Use field associated with this synchronizer to *for_client_use*
- Returns SYN_WHC\$Notice to *what_happened*

If *priority_level* is set to AllPriorities, and if there are notices on any synchronizer in the group, SYN\$GRTR decreases by 1 the notice count of a synchronizer with a notice at the highest priority level where at least one synchronizer has a notice.

If *priority_level* specifies a particular priority level, and if there are notices on any of the synchronizers at that priority level, SYN\$GRTR decreases by 1 the notice count of a synchronizer with a notice at *priority_level*.

If SYN\$GRTR does not find a notice on the event group or at the specified priority level, SYN\$GRTR returns SYN_WHC\$NoNotice to *what_happened*, NullSyncNum to *sync_identifier*, and NullFCU to *for_client_use*.

A process should call SYN\$GRTR when it wants to retrieve a notice from any of a set of event synchronizers but has other work to do and under no circumstances wants to wait. For example, the calling process might be performing a relatively low priority task that requires lengthy

processing. Before the process finishes the low priority task, it might want to retrieve any higher priority event notices that may have been posted since it began to process the low priority task.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$GDST**SYN\$GDST****SYN\$GD**

Destroys an event group.

Usage

DCL SYN\$GDST ENTRY (FIXED BIN(15), FIXED BIN(15));

CALL SYN\$GDST (*group_identifier*, *code*);

Parameters***group_identifier***

INPUT. The identifier of the event group that SYN\$GDST is to destroy.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$GDST was completed without error.

SYN_SC\$InvGroupNum

group_identifier does not specify a valid event group.

SYN_SC\$SyncHasWaiter

SYN\$GDST could not destroy the event group because at least one process was waiting on the group.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$GDST destroys an event group. A process should destroy an event group when it no longer needs the group. SYN\$GCRE can assign the identifier of a destroyed event group to a new group. For this reason, when the calling process destroys an event group, it should remove all references to the group identifier from its databases.

SYN\$GDST does not destroy the synchronizers within an event group. Synchronizers that were in a destroyed group retain the notice counts that they had at the time when the group was destroyed.

Note

When SYN\$GDST is called to destroy a group that contains many synchronizers, it could block other synchronizer operations of the calling server. To avoid delaying these operations, remove each synchronizer from a group before calling SYN\$GDST to destroy the group.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

Retrieving Information About Event Synchronization

Introduction

A user process can call subroutines to get information about particular event synchronizers and event groups. Table 4-1 lists these subroutines and the types of information that they return. Programs written in FTN must use the six-character subroutine names listed.

Table 4-1
Subroutines Returning Information About
Synchronizers, Groups, and Timers

<i>Name</i>	<i>Information Returned</i>
SYN\$CHCK SYN\$CK	Number of notices; number of waiting processes
SYN\$GCHK SYN\$GK	Number of notices on a group at one or all priority levels; if all levels, also returns number of waiting processes
SYN\$INFO SYN\$IF	Whether synchronizer is in group, and if it is, the group number, the priority level, and the For Client Use field
SYN\$LSIG SYN\$LG	List of the synchronizers in group and total number
SYN\$LIST SYN\$LS	List of the synchronizers in server and total number
SYN\$GLST SYN\$GL	List of the groups in server and total number

Subroutine Descriptions

This section contains descriptions of the subroutines that return information about event synchronizers and event groups. In the subroutine descriptions, the data type declarations and the possible values for the parameter *code* are in PL/I format.

SYN\$CHCK**SYN\$CHCK****SYN\$CK**

Returns the total number of notices or the number of waiting processes on an event synchronizer.

Usage

DCL SYN\$CHCK ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(15),
FIXED BIN(15));

CALL SYN\$CHCK (*sync_identifier*, *notices*, *waiters*, *code*);

Parameters*sync_identifier*

INPUT. The identifier of the event synchronizer about which SYN\$CHCK is to return information.

notices

OUTPUT. The current notice count of the synchronizer whose number is specified by *sync_identifier*. The notice count will be greater than or equal to 0.

waiters

OUTPUT. The number of processes currently waiting on the synchronizer whose number is specified by *sync_identifier*. The number of waiters will be greater than or equal to 0.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$CHCK was completed without error.

SYN_SC\$InvSyncNum

sync_identifier does not specify a valid event synchronizer.

SYN_SC\$SyncInGroup

The synchronizer specified by *sync_identifier* is in an event group. SYN\$CHCK cannot check on the status of synchronizers in event groups.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$CHCK returns the synchronizer's current notice count to *notices* and the number of processes currently waiting on the synchronizer to *waiters*. Because a synchronizer cannot have both notices and waiters, only one of these two values can be greater than zero.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$GCHK**SYN\$GCHK****SYN\$GK**

Returns the count of notices or waiters on an event group.

Usage

DCL SYN\$GCHK ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(15),
FIXED BIN(15), FIXED BIN(15));

CALL SYN\$GCHK (*group_identifier*, *priority_level*, *notices*, *waiters*, *code*);

Parameters***group_identifier***

INPUT. The identifier of the event group about which the calling process wants to obtain information.

priority_level

INPUT. Specify the priority level for which SYN\$GCHK is to return the count of notices. To get the count of notices or waiters on the entire group, specify AllPriorities. The synchronizer include files define the value of AllPriorities.

notices

OUTPUT. The count of notices on the synchronizers in this group at the priority level specified by *priority_level*. If *priority_level* is AllPriorities, SYN\$GCHK sets *notices* to the total of notices on all synchronizers in the event group.

waiters

OUTPUT. The number of processes waiting on this event group. *waiters* is always set to 0 if *priority_level* is not AllPriorities.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$GCHK was completed without error.

SYN_SC\$InvGroupNum

group_identifier does not specify a valid event group.

SYN_SC\$InvPriority

The value specified by *priority_level* is not AllPriorities or within the range of priority levels that are valid for this event group. The value of AllPriorities is defined by the synchronizer include file.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$GCHK returns information about the current state of a particular event group.

If *priority_level* is set to AllPriorities, SYN\$GCHK returns the total number of notices on the group to *notices*, and the number of processes currently waiting on the group to *waiters*. If *priority_level* is not set to AllPriorities, SYN\$GCHK returns to *notices* the number of notices on all synchronizers at the priority level specified by *priority_level*. In this case, SYN\$GCHK returns 0 to *waiters*, because a process can wait only on an entire event group, and not on the synchronizers within a group at a particular priority level.

Because a group cannot have notices and waiters at the same time, only one of the values *waiters* or *notices* can be nonzero.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$INFO**SYN\$INFO****SYN\$IF**

Returns information about an event synchronizer's membership in an event group.

Usage

DCL SYN\$INFO ENTRY (FIXED BIN(15), PTR OPTIONS(SHORT), FIXED BIN(15));

CALL SYN\$INFO (*sync_identifier*, *information*, *code*);

Parameters*sync_identifier*

INPUT. The identifier of the event synchronizer about which SYN\$INFO is to return information.

information

INPUT → OUTPUT. A pointer to a record to which SYN\$INFO returns information about the current state of the event synchronizer specified by *sync_identifier*. See *Discussion* below for information about the structure of this record.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$INFO was completed without error. If the status is not SYN_SC\$OK, any information returned is meaningless.

SYN_SC\$InvSyncNum

sync_identifier does not specify a valid event synchronizer.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$INFO indicates whether a particular event synchronizer is in a group; if it is, SYN\$INFO also returns the group identifier, priority level, and For Client Use field.

SYN\$INFO returns information to the record named by the pointer *information*. This record has the following structure:

```

dcl 1 SyncInfoRec based,
    2 InGroup bit(1) aligned,
    2 GroupNum fixed bin(15),
    2 Priority fixed bin(15),
    2 ForClient (3) fixed bin (15);
```

The elements of this record contain the following information:

InGroup

Set to TRUE if the synchronizer is in a group, or to FALSE (= 0) if it is not. SYN\$INFO sets the field to TRUE by setting its most significant bit to 1. If the synchronizer is not in a group, the other elements in the record do not contain meaningful information.

GroupNum

A 16-bit integer specifying the event group to which the synchronizer belongs.

Priority

A 16-bit integer specifying the priority level of the synchronizer within the group.

ForClient

Three 16-bit integers containing the contents of the synchronizer's For Client Use field.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$LSIG**SYN\$LSIG****SYN\$LG**

Lists the total number of synchronizers within this group and the identifier of each synchronizer.

Usage

```
DCL SYN$LSIG ENTRY (FIXED BIN(15), FIXED BIN(15), (*) FIXED BIN(15),  
                    FIXED BIN(15), FIXED BIN(15));
```

```
CALL SYN$LSIG (group_identifier, size, list, count, code);
```

Parameters***group_identifier***

INPUT. The identifier of the event group whose event synchronizers SYN\$LSIG is to list.

size

INPUT. The size of the array *list* to which SYN\$LSIG is to return the numbers of the event synchronizers in the event group.

list

INPUT → OUTPUT. An array to which SYN\$LSIG returns the identifiers of the event synchronizers in the event group. The program that calls SYN\$LSIG must allocate memory for this array.

count

OUTPUT. The total count of event synchronizers in the event group.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$LSIG was completed without error.

SYN_SC\$InvGroupNum

group_identifier does not specify a valid event group.

SYN_SC\$ListTooSmall

The array *list* is not large enough to hold the identifiers of all the synchronizers in this event group, and does not contain valid information. However, the information returned to *count* is valid. To correct this error, the caller should allocate memory for a larger array based on the value of *count*.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$LSIG returns the total count of event synchronizers in the event group to *count* and the identifiers of these synchronizers to the elements *list[1]* through *list[count]*. Use SYN\$INFO to get information about a particular synchronizer.

Note

The execution time of SYN\$LSIG is proportional to the number of synchronizers in the event group. When the number of synchronizers in the group is large, SYN\$LSIG should be used with caution, because other synchronizer functions will be delayed while SYN\$LSIG is executing.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$LIST

SYN\$LIST

SYN\$LS

Lists the total count of synchronizers within this server and the identifier of each synchronizer.

Usage

```
DCL SYN$LIST ENTRY (FIXED BIN(15), (*) FIXED BIN(15), FIXED BIN(15),  
                    FIXED BIN(15));
```

```
CALL SYN$LIST (size, list, count, code);
```

Parameters

size

INPUT. The size of array *list*, to which SYN\$LIST is to return the identifiers of the event synchronizers in this server.

list

INPUT → OUTPUT. An array to which SYN\$LIST returns the identifiers of the synchronizers within the server. The program that calls SYN\$LIST must allocate memory for this array.

count

OUTPUT. The total count of event synchronizers within this server.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$LIST was completed without error.

SYN_SC\$ListTooSmall

The array *list* is not large enough to hold the numbers of all the synchronizers within this server, and does not contain valid information. However, the information returned to *count* is valid. To correct this error, the caller should allocate memory for a larger array based on the value of *count*.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$LIST returns the total count of event synchronizers within this server to *count* and the identifiers of these synchronizers to the elements *list[1]* through *list[count]*.

Caution

The execution time of SYN\$LIST is proportional to the number of synchronizers and event groups belonging to the server. When the number of synchronizers and groups belonging to the server is large, SYN\$LIST should be used with caution, because other synchronizer functions will be delayed while SYN\$LIST is executing.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

SYN\$GLST

SYN\$GLST

SYN\$GL

Lists the total count of groups within this server and the identifier of each group.

Usage

```
DCL SYN$GLST ENTRY (FIXED BIN(15), (*) FIXED BIN(15), FIXED BIN(15),  
                    FIXED BIN(15));
```

```
CALL SYN$GLST (size, list, count, code);
```

Parameters

size

INPUT. The size of the array *list* to which SYN\$GLST is to return the identifiers of the event groups within this server.

list

INPUT → OUTPUT. An array to which SYN\$GLST returns the identifiers of the event groups within the server. The program that calls SYN\$GLST must allocate memory for this array.

count

OUTPUT. The total count of event groups within this server.

code

OUTPUT. The synchronizer status code. The possible codes are:

SYN_SC\$OK

The call to SYN\$GLST was completed without error.

SYN_SC\$ListTooSmall

The array *list* is not large enough to hold the identifiers of all the event groups within this server, and does not contain valid information. However, the information returned to *count* is valid. To correct this error, the caller should allocate memory for a larger array based on the value of *count*.

SYN_SC\$InternalError

An unexpected system error occurred. Report any occurrence of this error to the System Administrator.

Discussion

SYN\$GLST returns the total count of event groups within this server to *count* and a list of the group identifiers to array elements *list[1]* through *list[count]*.

Caution

The execution time of SYN\$GLST is proportional to the number of synchronizers and event groups belonging to the server. When the number of synchronizers and groups belonging to the server is large, SYN\$GLST should be used with caution, because other synchronizer functions will be delayed while SYN\$GLST is executing.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

Timers

Introduction

A user process may need a notice to be posted on a synchronizer at a particular time or times. Such a process can create a **timer** to post the notice or notices. There are three types of timers:

- An **absolute timer** posts a notice on an event synchronizer at a particular date and time of day.
- An **interval timer** posts a notice on an event synchronizer after a fixed amount of time passes.
- A **repetitive timer** posts a notice on a synchronizer periodically, at a fixed interval of time.

A timer posting a notice on a synchronizer has the same effect as a process posting a notice. If processes are waiting on the synchronizer, one of the processes starts to run. If no processes are waiting on the synchronizer, the timer increases the synchronizer's notice count by 1. If the synchronizer is part of an event group, one of the processes waiting on the event group starts to run. If the event group has no waiting processes, the timer increases the synchronizer's notice count by 1.

Note

If notices are not retrieved from an event synchronizer on which a repetitive timer is posting notices, the event synchronizer will eventually overflow and become invalid.

Processes call subroutines to create, set, cancel, and destroy timers. A timer is private to a single server and cannot be accessed by other servers.

Timer Subroutines

Table 5-1 lists the subroutines that create, destroy, set, and cancel timers. Programs written in FTN must use the six-character subroutine names listed.

Table 5-1
Timer Subroutines

<i>Name</i>	<i>Function</i>
TMR\$CREA TMR\$CR	Creates a timer
TMR\$DEST TMR\$DE	Destroys a timer
TMR\$\$SABS TMR\$\$SA	Sets an absolute timer
TMR\$\$SINT TMR\$\$SI	Sets an interval timer
TMR\$\$SREP TMR\$\$SR	Sets a repetitive timer
TMR\$CANL TMR\$CN	Cancels a timer
TMR\$GTMR TMR\$TI	Returns the timer type and information about the current state of the timer
TMR\$LIST TMR\$LS	Lists the identifiers of the timers within a server and the total number of timers within the server

The following paragraphs describe the functions that timer subroutines perform.

Creating Timers

To create a timer, a process calls the subroutine TMR\$CREA. TMR\$CREA assigns an identifier to the timer. This identifier becomes an input parameter to other subroutines that access the timer. The calling process specifies the type of timer it is creating (absolute, interval, or repetitive).

Setting Timers

A process can call the subroutine TMR\$\$SABS to set an absolute timer, TMR\$\$SINT to set an interval timer, and TMR\$\$SREP to set a repetitive timer. Each subroutine can set timers of only one type. A timer must be set in order to post a notice.

When a process calls a subroutine to set a timer, it specifies the event synchronizer on which the timer is to post a notice, as well as the following information:

- For absolute timers, the date and time of day when the timer is to expire
- For interval timers, the interval of time after which the timer is to expire
- For repetitive timers, the interval of time at which the timer is to post notices periodically

Cancelling Timers

A process can call the subroutine TMR\$CANL to cancel a timer at any time after the timer is created. A cancelled timer cannot post notices, but it can be reset by TMR\$\$ABS, TMR\$\$INT, or TMR\$\$REP.

Note

A process should cancel a timer before it destroys a synchronizer on which the timer is set to post a notice.

Destroying Timers

When a timer is no longer needed, the process that created the timer can call the subroutine TMR\$DEST to destroy it.

The INITIALIZE_COMMAND_ENVIRONMENT (ICE) command with the -SERVER option deallocates all timers and event synchronizers belonging to a user process. The ICE command with the -SERVER option also logs out any child processes belonging to the user process. The ICE command without the -SERVER option does not affect timers or event synchronizers. See the *PRIMOS Commands Reference Guide* for information about the ICE command.

States of Interval and Absolute Timers

An absolute or interval timer is always in one of three states: initial, pending, or expired. An absolute or interval timer enters:

- The **initial state** when a process creates or cancels it.
- The **pending state** when a process sets it.
- The **expired state** when the specified amount of time elapses. An absolute or interval timer posts a notice on a synchronizer when it passes from the pending state to the expired state. It remains in the expired state until a process sets, destroys, or cancels it.

When a process calls a subroutine to set, cancel, or destroy an absolute or interval timer, the subroutine returns an **Expired Status** value that is TRUE if the timer is in the expired state or FALSE if it is not. The Expired Status value indicates whether a synchronizer was notified before the timer was cancelled or destroyed.

States of Repetitive Timers

A repetitive timer is always in the initial state or the pending state. A repetitive timer enters the initial state when a process creates or cancels it. It enters the pending state when a process sets it and it remains in the pending state when the fixed interval of time elapses. A repetitive timer posts notices on a synchronizer each time the specified interval of time elapses. The Expired Status of a repetitive timer is always FALSE.

Include Files for Timers

The SYSCOM directory contains a number of **include files** for programs that invoke timer subroutines. Timer include files define certain key values that correspond to numeric values returned or expected by timer subroutines. These key values serve as status codes and other arguments. Timer include files also define templates for data structures used by timer subroutines.

Each programming language has its own separate include file. A program that calls timer subroutines should include the timer include file for its language and use the key values defined in the file, rather than their numeric equivalents.

The names of the include files for timers are of the form

TIMERMIK.INS.*language*

where *language* is an abbreviation specifying the programming language. For example, the timer include file for Pascal is

TIMERMIK.INS.PASCAL

Before writing a program that invokes timer subroutines, consult the include file for the program's language to determine the correct key values for that language.

Note

Because of the name length limits of the FTN language, the key values in the FTN key file are substantially different from those in other language files. The FTN timer status codes are listed in Appendix C.

Subroutine Descriptions

This section contains descriptions of the subroutines that create, use, and destroy timers. In the subroutine descriptions, the data type declarations and the possible values for *code* are in PL/I format.

TMR\$CREA**TMR\$CREA****TMR\$CR**

Creates a timer.

Usage

DCL TMR\$CREA ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(15));

CALL TMR\$CREA (*type*, *timer_identifier*, *code*);

Parameters***type***

INPUT. The kind of timer that TMR\$CREA is to create. TMR\$CREA can create absolute, interval, and repetitive timers. Specify a key value defined by the timer include file for the language of the calling program. The key value should specify the type of timer that is to be created. For example, the PL/I include file defines the key values ABSOLUTE, INTERVAL, and REPETITIVE.

timer_identifier

OUTPUT. The identifier of the timer that TMR\$CREA created.

code

OUTPUT. The timer status code. The possible codes are:

TMR_SC\$OK

The call to TMR\$CREA was completed without error.

TMR_SC\$NoResources

TMR\$CREA did not create a timer because the system does not have the resources to support another timer.

TMR_SC\$MaxTimerAlloc

TMR\$CREA did not create a timer because this server already had the maximum number of timers allowed to it.

Discussion

TMR\$CREA creates an absolute, interval, or repetitive timer and returns an identifier by which the timer can be accessed. TMR\$CREA does not set the timer or specify which event synchronizer the timer is to notify when it expires.

The value returned to *timer_identifier* is used as an input parameter to other subroutines that access the timer.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

TMR\$DEST**TMR\$DEST****TMR\$DE**

Destroys a timer.

Usage

DCL TMR\$DEST ENTRY (FIXED BIN(15), BIT(1) ALIGNED, FIXED BIN(15));

CALL TMR\$DEST (*timer_identifier*, *expired_status*, *code*);

Parameters***timer_identifier***

INPUT. The identifier of the timer that TMR\$DEST is to destroy.

expired_status

OUTPUT. A value indicating whether the timer is in the expired state. For absolute and interval timers, TMR\$DEST returns TRUE if the timer is in the expired state or FALSE if the timer is in any other state. TMR\$DEST sets the value to 0 to indicate FALSE; it sets the value's most significant bit to 1 to indicate TRUE. For repetitive timers, TMR\$DEST always returns FALSE.

code

OUTPUT. The timer status code. The possible codes are:

TMR_SC\$OK

The call to TMR\$DEST was completed without error.

TMR_SC\$InvTimer

TMR\$DEST did not destroy a timer because *timer_identifier* does not specify a valid timer.

Discussion

TMR\$DEST destroys the timer whose number is specified by *timer_identifier*. TMR\$CREA can reassign the number of a destroyed timer to a new timer.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

TMR\$\$SABS

TMR\$\$SA

Sets an absolute timer.

Usage

```
DCL TMR$$SABS ENTRY (FIXED BIN(15), FIXED BIN(15), 1,2 FIXED BIN(31),  
2 FIXED BIN(31), BIT(1) ALIGNED, FIXED BIN(15));
```

```
CALL TMR$$SABS (timer_identifier, sync_identifier, expiration_time,  
expired_status, code);
```

Parameters

timer_identifier

INPUT. The identifier of the absolute timer that TMR\$\$SABS is to set.

sync_identifier

INPUT. The identifier of the event synchronizer on which the absolute timer is to post a notice when it expires. Use SYN\$CREA to create an event synchronizer.

expiration_time

INPUT. The date and time of day when the absolute timer is to expire. Specify this value in **absolute time**. Absolute time is expressed as the elapsed date and time, in milliseconds, since midnight of January 1, 1901, in Greenwich Mean Time. The system rounds the *expiration_time* value down to the next lower whole minute. Use the subroutine TMR\$LOCALCONVERT to convert local time to absolute time. For information about TMR\$LOCALCONVERT, see the *Subroutines Reference III: Operating System*.

expired_status

OUTPUT. A value indicating whether the timer was in the expired state when TMR\$\$SABS was called. TMR\$\$SABS returns TRUE if the timer is in the expired state or FALSE if the timer is in any other state. TMR\$\$SABS sets the value to 0 to indicate FALSE; it sets the value's most significant bit to 1 to indicate TRUE.

code

OUTPUT. The timer status code. The possible codes are:

TMR_SC\$OK

The call to TMR\$\$SABS was completed without error.

TMR\$SABS

TMR_SC\$InvTimer

TMR\$SABS did not set the timer because *timer_identifier* does not specify a valid timer.

TMR_SC\$InvTimeParameter

The expiration time specified by *expiration_time* is not valid because it is not greater than the current system time.

Discussion

TMR\$SABS is called to set an absolute timer to expire at a particular date and time of day. The caller specifies the event synchronizer on which the timer is to post a notice when it expires. TMR\$SABS can reset an absolute timer while it is in the pending or expired state. You do not need to cancel a pending timer before you reset it.

If the system time changes while an absolute timer is pending, the absolute timer will expire at the absolute time based on the new system time.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

TMR\$\$SINT

TMR\$\$SI

Sets an interval timer.

Usage

```
DCL TMR$$SINT ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(31),  
                     BIT(1) ALIGNED, FIXED BIN(15));
```

```
CALL TMR$$SINT (timer_identifier, sync_identifier, expiration_interval,  
                expired_status, code);
```

Parameters

timer_identifier

INPUT. The identifier of the interval timer that TMR\$\$SINT is to set.

sync_identifier

INPUT. The identifier of the event synchronizer on which the interval timer is to post a notice when it expires.

expiration_interval

INPUT. The interval of time after which the timer is to expire. This interval must be expressed in milliseconds. The system rounds the value down to the next lower interval of 100 milliseconds. *expiration_interval* must not be equal to 0.

expired_status

OUTPUT. A value indicating whether the timer was in the expired state when TMR\$\$SINT was called. TMR\$\$SINT returns TRUE if the timer is in the expired state or FALSE if the timer is in any other state. TMR\$\$SINT sets the value to 0 to indicate FALSE; it sets the value's most significant bit to 1 to indicate TRUE.

code

OUTPUT. The timer status code. The possible codes are:

TMR_SC\$OK

The call to TMR\$\$SINT was completed without error.

TMR_SC\$InvTimer

TMR\$\$SINT did not set an interval timer because the value of *timer_identifier* does not specify a timer within this server.

TMR\$SINT

TMR_SC\$InvTimerParameter

expiration_interval is not valid because it is 0.

Discussion

TMR\$SINT sets an interval timer to expire after an interval of time passes and to post a notice on a specified event synchronizer when the timer expires.

TMR\$SINT can reset an interval timer while it is in the pending state; there is no need to cancel the timer before resetting it. If the system time is changed while an interval timer is in the pending state, the timer expires when it would have expired if the system time had not been changed.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

TMR\$\$REP

TMR\$\$SR

Sets a repetitive timer.

Usage

```
DCL TMR$$REP ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(31),  
                    FIXED BIN(15));
```

```
CALL TMR$$REP (timer_identifier, sync_identifier, expiration_interval, code);
```

Parameters

timer_identifier

INPUT. The identifier of the repetitive timer that TMR\$\$REP is to set.

sync_identifier

INPUT. The identifier of the event synchronizer on which the timer is to post notices periodically.

expiration_interval

INPUT. The interval of time at which the timer is to post notices periodically. Specify this interval in milliseconds. The system rounds the value down to the next lower multiple of 100 milliseconds. *expiration_interval* must not be 0.

code

OUTPUT. The timer status code. The possible codes are:

TMR_SC\$OK

The call to TMR\$\$REP was completed without error.

TMR_SC\$InvTimer

The value *timer_identifier* does not specify a timer within this server.

TMR_SC\$InvTimerParameter

The value specified by *expiration_interval* is not valid because it is 0.

TMR\$\$REP

Discussion

TMR\$\$REP sets a repetitive timer to post a notice periodically, at a fixed interval of time, and specifies the event synchronizer on which the timer is to post the notice. After TMR\$\$REP sets the repetitive timer, the timer remains in the pending state until TMR\$CANL cancels it. When TMR\$\$REP sets a repetitive timer while it is in the pending state, TMR\$\$REP cancels the current timing interval and the timer starts timing the new interval.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

TMR\$CANL

TMR\$CN

Cancels a timer.

Usage

DCL TMR\$CANL ENTRY (FIXED BIN(15), BIT(1) ALIGNED, FIXED BIN(15));

CALL TMR\$CANL (*timer_identifier*, *expired_status*, *code*);

Parameters

timer_identifier

INPUT. The identifier of the timer that TMR\$CANL is to cancel.

expired_status

OUTPUT. A value indicating whether the timer has already expired when it is cancelled. For absolute and interval timers, TMR\$CANL returns TRUE if the timer is in the expired state or FALSE if the timer is in any other state. TMR\$CANL sets the value to 0 to indicate FALSE; it sets the value's most significant bit to 1 to indicate TRUE. For repetitive timers, *expired_status* is always FALSE.

code

OUTPUT. The timer status code. The possible codes are:

TMR_SC\$OK

The call to TMR\$CANL was completed without error.

TMR_SC\$InvTimer

TMR\$CANL did not cancel the timer because *timer_identifier* does not specify a valid timer.

Discussion

TMR\$CANL cancels a pending absolute, interval, or repetitive timer. TMR\$CANL cancels all future periodic expirations of a repetitive timer. The *expired_status* value of the timer indicates whether the timer has already notified its synchronizer at the time when TMR\$CANL cancels the timer.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

TMR\$GTMR**TMR\$GTMR****TMR\$TI**

Returns information about a timer.

Usage

```
DCL TMR$GTMR ENTRY (FIXED BIN(15), PTR OPTIONS(SHORT),  
                    FIXED BIN(15));
```

```
CALL TMR$GTMR (timer_identifier, information, code);
```

Parameters*timer_identifier*

INPUT. The identifier of the timer about which TMR\$GTMR is to return information.

information

INPUT → OUTPUT. A pointer to a record to which TMR\$GTMR returns information about the timer.

code

OUTPUT. The timer status code. The possible codes are:

TMR_SC\$OK

The call to TMR\$GTMR was completed without error.

TMR_SC\$InvTimer

TMR\$GTMR did not return information because the value of *timer_identifier* does not specify a valid timer.

Discussion

TMR\$GTMR returns information about a timer to a record pointed to by *information*. The program that calls TMR\$GTMR must allocate eight words of memory for the record and pass the address of the record to TMR\$GTMR. Absolute, interval, and repetitive timers require different types of records. The programmer must allocate the proper type of record for the timer specified in the call to TMR\$GTMR.

The record for absolute timers has the following format:

```
/* Declaration for Absolute Timers*/
dcl 1 AbsTimerInfo based,
    2 Timer fixed bin(15),
    2 Sync fixed bin(15),
    2 State fixed bin(15),
    2 Kind fixed bin(15),
    2 ExpirationTime like AbsoluteTime;
```

The fields of the record for absolute timers receive the following items of information:

Timer

The identifier of the timer about which TMR\$GTMR returns information

Sync

The identifier of the event synchronizer that the timer is to notify when it expires

State

The state of the timer (initial, pending, or expired) when TMR\$GTMR is executed

Kind

The kind of timer

ExpirationTime

The time when the timer is to expire, in absolute time. Absolute time is the elapsed date and time since midnight of January 1, 1901, expressed in milliseconds. The timer include files declare the absolute time data type.

The record for interval timers has the following format:

```
/* Declaration for Interval Timers*/
dcl 1 IntTimerInfo based,
    2 Timer fixed bin(15),
    2 Sync fixed bin(15),
    2 State fixed bin(15),
    2 Kind fixed bin(15),
    2 RemainingTime fixed bin(31);
```

TMR\$GTMR

The **Timer**, **Sync**, **State**, and **Kind** fields of the interval timer record receive the same information as the corresponding fields in the absolute timer record. The following field is unique to the interval timer record:

RemainingTime

Contains the amount of time remaining until the interval timer expires, at the time when TMR\$GTMR is executed.

The repetitive timer record has the following format:

```
/* Declaration for Repetitive Timers*/  
dcl 1 RepTimerInfo based,  
    2 Timer fixed bin(15),  
    2 Sync fixed bin(15),  
    2 State fixed bin(15),  
    2 Kind fixed bin(15),  
    2 RepState,  
    3 RemainingTime fixed bin(31),  
    3 SucceedingIntervals fixed bin(31);
```

The **Timer**, **Sync**, **State**, and **Kind** fields of the repetitive timer record receive the same information as the corresponding fields in the absolute timer record. The **RepState** field is unique to the repetitive timer record, and consists of the following elements:

RemainingTime

The amount of time remaining until the timer next posts a notice.

SucceedingIntervals

The length of the interval of time at which the timer periodically posts a notice.

The timer can be in any state when TMR\$GTMR is executed. TMR\$GTMR does not cancel or in any way affect the timer about which it returns information. Note that the state of a timer may change immediately after the call to TMR\$GTMR.

Effective for PRIMOS Revision 22.0 and subsequent revisions.

TMR\$LIST

TMR\$LS

Lists the total number of timers within this server and their identifiers.

Usage

```
DCL TMR$LIST ENTRY (FIXED BIN(15), (*) FIXED BIN(15), FIXED BIN(15),  
                    FIXED BIN(15));
```

```
CALL TMR$LIST (size, list, count, code);
```

Parameters

size

INPUT. The size of the caller allocated array *list* to which TMR\$LIST is to return the identifiers of the timers within this server.

list

OUTPUT. An array to which TMR\$LIST returns the identifiers of the timers within the server. The program that calls TMR\$LIST must allocate memory for this array.

count

OUTPUT. The total count of timers within this server.

code

OUTPUT. The timer status code. The possible codes are:

TMR_SC\$OK

The call to TMR\$LIST was completed without error.

TMR_SC\$ListTooSmall

The array *list* is not large enough to hold the identifiers of all the timers within this server, and the information in *list* is not valid. However, the information returned to *count* is valid. To correct this error, the caller should allocate memory for a larger array based on the value of *count*.

Discussion

TMR\$LIST returns the identifiers of all the timers within the server. It also returns the total count of timers within the server.

Effective for PRIMOS Revision 22.0 and subsequent revisions.



Part IV, InterServer Communications



General Discussion of InterServer Communications

Introduction

The Prime InterServer Communications (ISC) facility provides two-way message exchange between pairs of concurrently running servers. For example, ISC can be used for the two-way transfer of information between a server associated with a user process and a database server. Another use of ISC would be to pair many servers to a single centralized server that logs status information.

ISC uses the term **server** to refer to a single process or a group of closely cooperating processes. PRIMOS places each terminal or phantom process in its own server when the process is initialized. For example, when a user logs in, the user's process receives its own server name. A PRIMIX child process, however, is placed in its parent's server. ISC message exchange can involve servers for user processes or for system processes. Chapter 7 describes how to determine the server name of a process.

Message exchange occurs within a **session**. A session is a one-to-one link between two active servers. Only two servers can participate in a session; each server can both send and receive multiple messages with the other server participating in the session. A server that is not a participant in the session cannot read messages or otherwise interact with the session.

A server can participate in **multiple concurrent sessions**. A server can establish sessions to several other servers. Two servers can be connected by more than one session. A server can even establish a session with itself.

ISC permits you to perform message exchange between servers on different PRIMENET™ nodes. This is referred to as a **remote session**. During a remote session, ISC automatically handles the interface with PRIMENET, making all ISC processing across PRIMENET nodes appear identical to ISC processing within the same node. Special considerations for this type of ISC processing are found in Chapter 13, Remote Sessions.

Servers can coordinate the sending and receiving of messages by using **synchronizers**. You can configure synchronizers associated with specific ISC events to your server for the duration of the session. When an ISC event occurs, ISC posts a notice on the appropriate synchronizer. For example, when one server sends a message, ISC posts a notice on the other server's ReadyToReceive synchronizer. The posting of this notice informs the server that a message is pending, ready to be received. Responding to a notice posted on an ISC synchronizer is the

responsibility of the programmer. General information about synchronizers can be found in Chapter 2. The ISC synchronizers are further described in Chapter 9.

Phases of an ISC Session

An ISC session consists of three basic phases: session establishment, message exchange (also referred to as data transfer), and session termination.

Session Establishment. To establish a session, one server must send a session request, and the other server must accept the request. Any server can request a session with any other server. The servers can be on the same PRIMENET node, or on different nodes. However, you can request a session only between servers that represent processes that are currently running. Session establishment is described in Chapters 8 and 9.

Message Exchange. Once a session has been established, either server can begin sending messages to the other server. Both servers can concurrently send and receive multiple messages. ISC transmits messages exactly as specified; it does not group multiple messages together or in any way process the contents of a message. Message exchange is described in Chapter 10.

Session Termination. Either server can terminate the session at any time during session establishment or message exchange. Because a session can exist only between processes that are currently logged in, the logout of a process automatically terminates any session in which it participates. If a server or some system event terminates a session, the other server receives an **exception**. An exception is a notification of an unusual event. After processing this exception, the remaining server can continue to process previously queued messages. The remaining server must then also terminate the session. Session termination is described in Chapter 11.

A Choice of Message Types

ISC provides facilities for several different kinds of messages:

Normal Messages are messages sent during an established session. The session default is Normal message service. A Normal message can consist of two parts: a **control part** for short messages (maximum of 128 characters), and a **data part** for longer messages (maximum of 32,630 characters). All ISC message types can have a control part; only Normal messages can have a data part. The control part and data part of a Normal message differ in nature as well as size. A control part is copied from an area in the sending program to an area in the receiving program. A data part is created in the session's message area. Information used to locate this message area is then sent to the other server; when the other server receives the message, ISC uses this information to provide the receiving server with a pointer to the data part message. You can send a Normal message that consists of only a control part, only a data part, or both a control part and a data part. Information of any type can be sent in either the control part or the data part of a message. You can send a Normal message that has neither a control part nor a data part in order to synchronize the sending and receiving servers.

Expedited Messages are messages sent during an established session. The session must be specially configured to support exchange of Expedited messages. Expedited messages are short messages (maximum of 64 characters) that consist of a control part only. An Expedited message is copied from an area in the sending program to an area in the receiving program. Expedited messages and Normal messages are queued separately. This enables you to read all your Expedited messages first, then read your Normal messages.

Connect Messages are messages sent while you are establishing or terminating a session. They are short messages (maximum of 128 characters) that consist of a control part only. A Connect message is copied from an area in the sending program to an area in the receiving program. You can use these short messages rather than fully establishing a session if only a single brief message exchange is required. You can also use a Connect message when terminating a session to indicate why the session is being terminated. Connect messages are further described in Chapter 12.

Programming Considerations

You perform all ISC operations by issuing standard subroutine calls. ISC also provides subroutine calls to determine status and statistical information about an ISC session. Two PRIMOS commands (`LIST_SESSIONS` and `LIST_SERVER_NAMES`) also provide status information. These PRIMOS commands are described in the *PRIMOS Commands Reference Guide*.

In order to call ISC subroutines, you must provide certain information in your program. The following three items (or their equivalents) are usually required in programs that perform ISC operations:

- ISC key files
- An ISC structure file
- An array for brief messages

If your program allocates ISC synchronizers, you must also call subroutines to use these synchronizers.

Key Files

You should include key files in your program for the subroutines that you call. ISC uses two different key files, one for `IS$` subroutines and another for `SRS$` subroutines. In addition, you should include the key file for synchronizer subroutines used with ISC.

These key files contain key values for subroutine status codes and the special codes returned by `IS$GE`, `IS$GRS`, and `IS$GSS`. Key files contain the key value equivalents for the integer values returned by the subroutines. Always use the key values, not the integer values, for keys in your program.

The include files for ISC keys are located in the SYSCOM directory. The following key files are provided:

- ISC_KEYS.INS for IS\$ subroutines (status codes and special codes)
- SRS_CODES.INS for SRS\$ subroutines
- SYNC_CODES.INS for synchronizer subroutines

In every case, there are different include files for different programming languages. For example, the IS\$ key file for PL/I is SYSCOM>ISC_KEYS.INS.PL1. Include in your program the key file(s) with the suffix for your programming language. Key files are provided for FTN, PL/I, Pascal, PMA, and the C programming language. F77 users should use the FTN file.

Note

Because of the name length limits of the FTN language, the key values in the FTN key file are substantially different from those in other language files. The key values for FTN keys are shown in Appendix C.

The key values used by ISC are listed in Appendix C. You can also use the ER\$PRINT and ER\$TEXT subroutines to get the error message text for an individual key. These subroutines are described in the *Subroutines Reference III: Operating System*.

Structure Files

You should include in your program the ISC structures required for such operations as session configuration and message specification. ISC provides template files for these structures in the SYSCOM directory. Each structure file contains a complete set of templates for all ISC structures; the SYSCOM directory contains different structure files for different programming languages. For example, the structure file for PL/I is SYSCOM>ISC_STRUCTURES.INS.PL1. Include in your program the structure file with the suffix for your programming language. Structure files are provided for PL/I, Pascal, and the C programming language. The ISC structures are shown in Appendix E.

The structures in these template files are defined as based variables. That is, they provide templates for structures, but do not allocate any space for the structure itself. You must allocate space for each structure that you use in your program. You then establish a pointer to each of your structures for use by the ISC subroutines.

For example:

```
%include 'syscom>isc_keys.ins.pll';                      /* Include keys */
%include 'syscom>isc_structures.ins.pll';                 /* Include template structures */
dcl is$sta entry(fixed bin(15), ptr, fixed bin(15));      /* Declare subroutine */
dcl mystats like SessionStatisticsBlock;                  /* Associate your structure */
                                                         /* with template structure */
dcl mystatsptr ptr;                                       /* Declare pointer to your structure */
mystatsptr = addr(mystats);
mystats.version = isc_version_number;                     /* Set version number of structure */
call is$sta(number, myllnptr, code);                      /* Call subroutine */
```

Note

All ISC structures contain a **Version** field. When using an include file for ISC structures, use the `ISC_VERSION_NUMBER` key value to set this **Version** field. The `ISC_VERSION_NUMBER` key is found in the `ISC_STRUCTURES.INS` include files. You must set the **Version** field for every ISC structure, including "blank" structures used only for output.

Message Array

Connect messages, Expedited messages, and the control part of Normal messages are all copied from one server to the other. In order to create or receive a message, you must establish a message array in your program. This array should be 128 bytes in length; it can be shorter than 128 bytes if you define a shorter message control part. It must be aligned on a halfword boundary. After creating this array, you must establish a pointer to this array and copy this pointer into a Message Specifier in your program. Your program can contain multiple message arrays and Message Specifiers. The Message Specifier is described in Chapter 10.

Synchronizers

If ISC synchronizers are used to coordinate calls to ISC subroutines, the programmer must retrieve a notice from the appropriate synchronizer and then perform the corresponding operation. For example, ISC posts a notice on the ReadyToReceive synchronizer to inform a server that there is a message ready to be received. Therefore, the server receiving the message must first retrieve the notice from its ReadyToReceive synchronizer, then call `IS$RM` to receive the message. You can retrieve a notice by calling either `SYN$WAIT` or `SYN$RTRV`. You use other subroutines to retrieve notices from a synchronizer that is a member of an event group. Notice retrieval subroutines are described in Chapters 2 and 3; ISC synchronizers are described in Chapter 9.

Server Names

Introduction

Before a process can exchange messages with another process, the process initiating the message exchange session must know the server name of the intended session recipient.

PRIMOS automatically assigns a server name to each process during the login operation. The server name is a computer-generated name that consists of a string of twelve consonants. A user process does not have the same server name during subsequent logins. (PRIMOS uses the system clock to generate server names. Therefore, a server name is always unique for that system, unless the system clock is reset.) Because you must supply the current server name of a process to establish an ISC session, you can only send messages to a process that is currently logged on to the system.

Some processes installed on the system which do not log in and log out, also have server names. These server names are standard names that remain the same, rather than computer-generated names.

A phantom process receives its own server name when it is spawned; this server name persists until the process terminates. A PRIMIX child process is assigned the same server name as its parent process.

The `INITIALIZE_COMMAND_ENVIRONMENT (ICE)` command with no options does not affect sessions, synchronizers, or timers. The ICE command with the `-SERVER` option terminates that user's sessions, and deallocates any synchronizers and timers belonging to that user. ICE `-SERVER` reinitializes the server's `SessionRequestPending` synchronizer. It also logs out any child processes belonging to the user process.

The `SRS$` subroutines and the PRIMOS command `LIST_SERVER_NAMES` enable users to determine the server name of a process. To determine the server name of a process, call `SRS$GN`. To retrieve a list of all active server names on your node, call `SRS$LN` or use the PRIMOS command `LIST_SERVER_NAMES`.

A server name is unique within a node, but a server name is not always unique across nodes. The server name is one component of the server's **Low Level Name (LLN)**, which uniquely identifies the server across nodes. An LLN contains the server name and the name of the node on which that server is running. To establish a message exchange session, you must know the LLN of the recipient.

Server Name Subroutines

Table 7-1 lists the subroutines that identify the server name corresponding to a process and the subroutines used to catalog and look up a server's Low Level Name.

Table 7-1
Server Name Subroutines

<i>Name</i>	<i>Function</i>
ISN\$C	Catalogs a server's Low Level Name
ISN\$L	Looks up a server's Low Level Name
ISN\$RC	Recatalogs a server's Low Level Name
ISN\$UC	Uncatalogs a server's Low Level Name
SRS\$GN	Gets the server name of a process
SRS\$GP	Gets the process numbers of all processes associated with a server name
SRS\$LN	Lists the server names on your system

Programs that use ISN\$ subroutines should include the SYSCOM>ERRD.INS error code file with the appropriate language suffix. Programs that use SRS\$ subroutines should include the SYSCOM>SRS_CODES.INS status code file with the appropriate language suffix.

Low Level Names

A low level name has the following format:

```
dcl 1 LowLevelName,  
  2 Version fixed bin(15),  
  2 NodeName char(16) var,  
  2 ServerName char(12) var,  
  2 ForClientUse fixed bin(31),  
  2 Reserved (13) fixed bin(15);
```

The fields are defined as follows:

Version

The version number of the structure. Use the ISC_VERSION_NUMBER key value to set this field. ISC_VERSION_NUMBER is supplied in the ISC_STRUCTURES.INS include files.

NodeName

The PRIMENET node name that identifies the system that is running the server. You can set this field to null in a Low Level Name used exclusively for local (non-remote) sessions. You can use GSNAM\$ to determine the **NodeName** element of your LLN. GSNAM\$ is described in the *Subroutines Reference III: Operating System*.

ServerName

The system-assigned server name. You can use SRS\$GN to determine the **ServerName** element of an LLN.

ForClientUse

An area that you can use for whatever purpose you wish. The contents of this field are not interpreted by ISC.

Reserved

For use by ISC only.

Cataloging Low Level Names

A server can use the ISN\$ subroutines to catalog its own Low Level Name in an easily accessible location. This greatly assists other users who need to know the LLN of the server before they can send a message to that server.

Each server that wishes to be available for ISC message exchange should determine its own LLN and then catalog this LLN in a High Level Name File (HLNF). If you wish, you can catalog your own Low Level Name as part of your login procedure. Each server should catalog its own LLN.

The System Administrator should establish conventions for the location and naming of High Level Name Files. These naming conventions permit users to quickly locate the desired HLNF. The name of the HLNF should be a name readily identified with the server, such as the user ID.

Following these naming conventions, you use ISN\$C to establish an HLNF for your server. Each HLNF contains the LLN of one server. Other users can read your current LLN from your HLNF using the ISN\$L routine.

Note

When you use BIND to link a program that contains ISN\$ subroutines, you must explicitly link the LIB>ISNLIB library.

You should provide ACL protection to each HLNF to enhance system security. Protecting the HLNF against unauthorized deletion or write operations prevents the substitution of the name of a dummy ("Trojan horse") server for the name of a real system server. The server should have full access to its own HLNF; other users should be restricted to LUR access. ACL protection of files is described in the *PRIMOS User's Guide*.

When a process logs out, its LLN becomes invalid. For this reason, you should either:

1. Use ISN\$UC to uncatalog (delete) your HLNF each time you log out, then use ISN\$C to catalog an HLNF containing a new LLN each time you log in.
2. Use ISN\$RC to recatalog (update) your LLN each time you log in.

The former method has the advantage of not leaving an obsolete LLN in your HLNF while your process is logged out. The latter method is especially useful when updating system processes. ISN\$RC can also be used to recatalog an LLN of a running process in order to update the **ForClientUse** field of the LLN.

ISN\$C

Catalogs a server's Low Level Name in a High Level Name File.

Usage

```
DCL ISN$C ENTRY (CHAR(*) VAR, PTR, FIXED BIN(15));
```

```
CALL ISN$C (hlnf, lln, code);
```

Parameters

hlnf

INPUT. The pathname of the High Level Name File to be created by this operation. The directory portion of this pathname should follow an agreed-upon convention for the location of High Level Name Files. The filename portion of this pathname should be a name (such as a user ID) commonly associated with your process. *hlnf* can be an absolute pathname or a relative pathname. If *hlnf* is a relative pathname (for example, **>MYID*) the High Level Name File is created in the currently attached directory; ISN\$C does not use the PRIMOS search rules facility. The pathname cannot contain wildcard characters or a password.

lln

INPUT. A pointer to a Low Level Name structure in your program. ISN\$C copies the Low Level Name from this structure into the High Level Name File.

code

OUTPUT. Standard error code. Possible values include:

E\$OK

The operation completed successfully.

E\$EXST

The specified HLNF cannot be created because an HLNF with this pathname already exists. Instead, use ISN\$RC to recatalog the LLN.

Discussion

ISN\$C takes as input the Low Level Name (LLN) of a server and a pathname for a High Level Name File (HLNF). It creates the specified HLNF, and then copies the Low Level Name into this HLNF. This operation is known as cataloging a Low Level Name. Cataloging your LLN in an HLNF enables other users to look up your LLN. A server must know the LLN of a recipient to initiate ISC operations.

Each HLNF contains one Low Level Name.

ISN\$C

The System Administrator should establish conventions for the location and names of High Level Name Files. By following these conventions, users on the system can catalog an LLN in a location that can be easily located by other users. Users can use ISN\$L to read a server's HLNF to determine the server's Low Level Name.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

ISN\$L

Looks up a server's Low Level Name in a High Level Name File.

Usage

```
DCL ISN$L ENTRY (CHAR(*) VAR, PTR, FIXED BIN(15));
```

```
CALL ISN$L (hlnf, lln, code);
```

Parameters

hlnf

INPUT. The pathname of an existing High Level Name File that contains a Low Level Name. This pathname can be an absolute pathname or a relative pathname. If *hlnf* is a relative pathname (for example, **>MYID*) the HLNf must be in the currently attached directory; ISN\$L does not use the PRIMOS search rules facility. The pathname cannot contain wildcard characters or a password.

lln

INPUT → OUTPUT. A pointer to a Low Level Name structure in your program. ISN\$L uses this pointer to copy a Low Level Name from the High Level Name File into your Low Level Name structure.

code

OUTPUT. Standard error code. Possible values include:

E\$OK

The operation completed successfully.

E\$FNTF

The specified HLNf does not exist.

E\$FER

The specified HLNf pathname refers to a file system object that is not an HLNf.

Discussion

ISN\$L takes as input the pathname of a High Level Name File (HLNF) and returns the Low Level Name cataloged in that HLNf. By using this subroutine, you can look up the LLN of a server.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

ISN\$RC

ISN\$RC

Recatalogs a server's Low Level Name in a High Level Name File.

Usage

DCL ISN\$RC ENTRY (CHAR(*) VAR, PTR, FIXED BIN(15));

CALL ISN\$RC (*hlnf*, *lln*, *code*);

Parameters

hlnf

INPUT. The pathname of an existing High Level Name File. This pathname can be an absolute pathname or a relative pathname. If *hlnf* is a relative pathname (for example, *>MYID) the HLNF must be in the currently attached directory; ISN\$RC does not use the PRIMOS search rules facility. The pathname cannot contain wildcard characters or a password.

lln

INPUT. A pointer to a Low Level Name structure in your program. ISN\$RC copies the Low Level Name from this structure into the High Level Name File, replacing the previous entry.

code

OUTPUT. Standard error code. Possible values include:

E\$OK

The operation completed successfully.

E\$FNTF

The specified HLNF does not exist. Use ISN\$C to create an HLNF and catalog the LLN.

E\$FER

The specified HLNF pathname refers to a file system object that is not an HLNF.

Discussion

ISN\$RC takes as input a pointer to the Low Level Name of a server and the pathname of a High Level Name File. This High Level Name File must have been previously used to catalog a Low Level Name. ISN\$RC replaces the contents of the High Level Name File with the specified Low Level Name.

ISN\$RC is commonly used to update a cataloged Low Level Name following a change to the ForClientUse field of the LLN.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

ISN\$UC

Uncatalogs (deletes) a server's High Level Name File.

Usage

```
DCL ISN$UC ENTRY (CHAR(*) VAR, FIXED BIN(15));
```

```
CALL ISN$UC (hlnf, code);
```

Parameters

hlnf

INPUT. The pathname of an existing High Level Name File. This pathname can be an absolute pathname or a relative pathname. If *hlnf* is a relative pathname (for example, **>MYID*) the HLNf must be in the currently attached directory; ISN\$UC does not use the PRIMOS search rules facility. The pathname cannot contain wildcard characters or a password.

code

OUTPUT. Standard error code. Possible values include:

E\$OK

The operation completed successfully.

E\$FNTF

The specified HLNf does not exist, and therefore cannot be deleted.

E\$FER

The specified HLNf pathname refers to a file system object that is not an HLNf. The file system object was not deleted.

Discussion

ISN\$UC takes as input the pathname of an existing High Level Name File. It deletes the High Level Name File. ISN\$UC can only perform this delete operation if the High Level Name File contains a Low Level Name.

You should perform an uncataloging operation when you terminate a process. If you do not uncatalog the HLNf of a terminated process, other users may unintentionally use the HLNf to look up an obsolete LLN.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

SRS\$GN

Gets the server name of a process.

Usage

```
DCL SRS$GN ENTRY (FIXED BIN(15), FIXED BIN(15), CHAR(12) VAR,  
                  FIXED BIN(15));
```

```
CALL SRS$GN (who, mysync, name, code);
```

Parameters

who

INPUT. The process number of the process whose server name is desired. The PRIMOS command STAT USER lists the numbers of all of the processes currently running on your system. To get the server name of the current process, specify 0.

mysync

OUTPUT. If *who* denotes the calling process, this parameter returns the number of the SessionRequestPending synchronizer associated with this server. For all other processes, this parameter returns NullSyncNum.

name

OUTPUT. The server name of the requested process.

code

OUTPUT. The status code. The possible codes are:

SRS_SC\$OK

The operation completed successfully.

SRS_SC\$NoSuchProcess

SRS\$GN returns this value if *who* is not the process number of a running process.

Discussion

SRS\$GN takes as input the process number of a user process and returns the server name for that process. It also can return the identifier of the server's SessionRequestPending synchronizer. This synchronizer is further described in Chapter 9.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

SRS\$GP

Gets the process numbers of all processes that share a specified server name.

Usage

```
DCL SRS$GP ENTRY (CHAR (*) VAR, FIXED BIN(15), (*) FIXED BIN(15),  
                 FIXED BIN(15), FIXED BIN(15));
```

```
CALL SRS$GP (name, arraysize, array, numproc, code);
```

Parameters

name

INPUT. The name of the server whose member processes are to be listed. You can input the server name as either uppercase or lowercase letters.

arraysize

INPUT. The number of process numbers that you expect to be returned by this subroutine. This should be the same size as the array you specify in *array* to hold the returned process numbers. An error occurs if you specify an array size smaller than the actual number of processes for the server.

array

OUTPUT. An array containing the process numbers returned by this subroutine. You create this array in your program and supply its name to the *array* parameter. PRIMOS copies the process numbers into the array you specified.

numproc

OUTPUT. The number of processes that are members of this server.

code

OUTPUT. The status code. The possible codes are:

SRS_SC\$OK

The operation completed successfully.

SRS_SC\$NoSuchServer

SRS\$GP returns this value if *name* does not specify the server name of an active server.

SRS_SC\$ListTooSmall

SRS\$GP returns this value if *arraysize* is smaller than the number of processes to be returned.

SRS\$GP

Discussion

SRS\$GP takes as input a server name and returns the process numbers of all processes that share that server. Most server names correspond to only a single process, but there are cases where multiple processes share the same server.

If there are more process numbers to be returned than space available in *array*, SRS\$GP returns ListTooSmall and sets *numproc* to indicate the actual number of processes.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

SRS\$LN

Lists the names of all active servers.

Usage

```
DCL SRS$LN ENTRY (FIXED BIN(15), (*) CHAR(12) VAR, FIXED BIN(15),  
                  FIXED BIN(15));
```

```
CALL SRS$LN (numservers, names, numnames, code);
```

Parameters

numservers

INPUT. The number of server names that you wish to retrieve. This subroutine begins retrieving the first server name on the system and continues until it retrieves the number of server names specified in this parameter. If you specify a number larger than the number of active servers, SRS\$LN retrieves the names of all of the servers currently running on the system.

names

OUTPUT. An array of server names retrieved by this subroutine. You create this array in your program and supply its name here. PRIMOS copies the server names into the array. Depending on the value of *numservers*, this array contains either the specified number of server names, or the names of all of the servers currently running on the system.

numnames

OUTPUT. The number of server names retrieved. This parameter is only set if *code* is SRS_SC\$OK.

code

OUTPUT. The status code. The possible codes are:

SRS_SC\$OK

The operation completed successfully.

SRS_SC\$ListTooSmall

SRS\$LN returns this value if *numservers* is smaller than the actual number of active servers. If this status code is returned, *names* records the number of server names specified in *numservers*, and *numnames* is not set.

SRS\$LN

Discussion

SRS\$LN returns a list of the names of active servers on your system. You can either return a complete list of active servers, or list a specified number of active servers. The order in which server names are returned, and which server names will be returned in a partial list are both unpredictable.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

Establishing a Session

Introduction

When you wish to initiate an ISC message exchange session, you must specify the Low Level Name (LLN) of the recipient server, and the parameters to be used during the session. This is known as requesting a session. If the system can fulfill the specified parameters, your session is assigned a session number and your session request is sent to the recipient server. The recipient server can either accept or reject the session request.

To establish a session, the two servers call the subroutines shown in Table 8-1 in the order shown:

Table 8-1
ISC Session Establishment Subroutines

<i>Name</i>	<i>Function</i>
IS\$RS	Initiator requests the session
IS\$GRQ	Recipient gets the session request
IS\$AS	Recipient accepts the session
IS\$GRS	Initiator gets the session request response

This chapter and the following chapter describe how to request a session. This chapter describes the basic session request operations. It assumes all session configuration parameters are set to default values. Chapter 9 describes how to assign nondefault session configuration parameters during session establishment.

Session Requests and Session Numbers

A session is a two-way connection established between two servers. ISC message exchange can only be performed within a session. A session is always between two and only two servers: the server that initiates the session, and the server that is the recipient of the session request. Note that the terms **initiator** and **recipient** refer to the establishment of a session, not the message exchange performed during the session. A session permits two-way communications; both servers can send or receive messages during a session. The session ends for each server when that server terminates the session, or when the server is logged off.

Each server identifies each of its sessions by means of a unique **session number**. ISC automatically generates these session numbers during session establishment. IS\$RS (the Request Session subroutine) returns a session number to the session initiator. IS\$GRQ (the Get Session Request subroutine) returns a session number to the session recipient. There is no relationship between the session number the initiator uses to identify the session and the session number the recipient uses to identify the same session. Each server uses its own session number to perform all subsequent operations during the session.

Each server can conduct multiple concurrent sessions, each session having its own unique session number. A server conducting a session with itself would be assigned two different session numbers, one as the session initiator and one as the session recipient.

Establishing a Session

You request a session using the IS\$RS (Request Session) subroutine. To do so, you specify the Low Level Name of the recipient. (The format of a Low Level Name is described in Chapter 7.) IS\$RS returns a session number, which you use to identify this session for all other ISC subroutines. The session request is queued to the recipient server.

When you call IS\$RS to request a session, ISC posts a notice on the SessionRequestPending synchronizer belonging to the intended session recipient. This informs the session recipient that a session request is available to be received. This synchronizer is further described in Chapter 9.

The session recipient retrieves this notice, then receives the session request using the IS\$GRQ (Get Session Request) subroutine. IS\$GRQ returns a session number to the recipient server. IS\$GRQ also returns authentication information about the session initiator and configuration information about the session. The authentication information includes the user ID, project ID, and ACL groups of the session initiator. (Authentication information may not be available for some servers on remote nodes; refer to Chapter 13 for details.) ISC performs no processing on this authentication information; it simply makes this information available to the session recipient. The session recipient should use this information to determine whether or not to establish a session with the session requestor.

If the session recipient wishes to participate in the requested session, it calls the IS\$AS (Accept Session) subroutine. If the session recipient does not wish to participate in the requested session, it calls the IS\$TS (Terminate Session) subroutine. IS\$TS is described in Chapter 11. Upon successful completion of a call to IS\$AS, the recipient's side of the session is established, and the recipient server can proceed immediately to exchanging messages.

When the session recipient calls IS\$AS, ISC posts a notice on the session initiator's SessionResponsePending synchronizer. This informs the session initiator that a response to the session request has arrived. This synchronizer is further described in Chapter 9.

The session initiator retrieves this notice, then completes session establishment by getting the response to the session request using the IS\$GRS (Get Session Response) subroutine. The initiator can then proceed to exchanging messages.

IS\$RS

Requests the establishment of an ISC message session.

Usage

```
DCL IS$RS ENTRY (PTR OPTIONS(SHORT), PTR OPTIONS(SHORT),  
                PTR OPTIONS(SHORT), FIXED BIN(15),  
                PTR OPTIONS(SHORT), FIXED BIN(15));
```

```
CALL IS$RS (lln, connectmessage, config, number, syncs, code);
```

Parameters

lln

INPUT. A pointer to the Low Level Name (LLN) structure of the intended session recipient. You can determine the LLN of a server by using the ISN\$L subroutine. The LLN pointer returned by that subroutine can be used as the value for this parameter. You can specify a ForClientUse field value as part of the LLN.

connectmessage

INPUT. Either supply a null value or specify a pointer to a Message Specifier in your program. The Message Specifier should contain a pointer to the text of the Connect message. If specified, this parameter sends a brief Connect message to the session recipient. A null value does not send a Connect message. The Message Specifier structure is described in Chapter 10. Connect messages are further described in Chapter 12.

config

INPUT. Either supply a null value or specify a pointer to a Session Configuration Block in your program. You can use the Session Configuration Block to specify local and global parameters for the session: the number and length of message queues, the size of messages and buffers, and the number of synchronizers to establish. A null pointer value sets all of these parameters to default values. These parameters and their default values are further described in Chapter 9.

number

OUTPUT. The initiator's session number assigned to this session.

syncs

INPUT → OUTPUT. A pointer to your Session Synchronizers List. You supply a pointer to a blank Session Synchronizers List in your program. ISC uses this pointer to write identifiers for your synchronizers into your Session Synchronizers List. The Session Synchronizers List is described in Chapter 9.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully.

ISC_SC\$NoRoom

ISC is unable to perform the operation at this time due to system limitations. For example, this code is returned if you have requested a session with a server that already has 63 pending session requests.

ISC_SC\$BadMessage

The Message Specifier is invalid.

ISC_SC\$TooManySessions

You have exceeded your quota for concurrent sessions. In order to request a new session, you must terminate an existing session.

ISC_SC\$InvalidConfig

You have requested an invalid configuration for the session. This error is returned if a configuration parameter value is outside of the range of permitted values. It is also returned if you did not set the reserved configuration parameters to zeros.

ISC_SC\$BadVersion

You have specified an invalid version number in one of the structures used by this subroutine.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

Discussion

IS\$RS is called by the session initiator. It requests an ISC session between two servers. You specify the Low Level Name of the recipient server, and upon successful completion, your session request is sent to the server you specified.

Upon successful completion, IS\$RS returns a session number assigned to this session. The session number returned is the initiator's session number; the recipient server receives its own session number.

IS\$RS permits you to configure the global parameters of the session and your own local parameters for the session. You can explicitly configure each of these parameters, or specify that the default configuration for all of these parameters should be used. For further details on configuration parameters and synchronizers, refer to Chapter 9.

IS\$RS permits you to send information to the other server. You can include information in the ForClientUse field of the Low Level Name, and you can send a Connect message when you call IS\$RS. When the session recipient calls IS\$GRQ to get the session request, it can receive both the Low Level Name and the Connect message. For additional information about Connect messages, refer to Chapter 12.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

IS\$GRQ

Gets a request for the establishment of an ISC session.

Usage

```
DCL IS$GRQ ENTRY (PTR OPTIONS(SHORT), PTR OPTIONS(SHORT),  
                  PTR OPTIONS(SHORT), FIXED BIN(15),  
                  PTR OPTIONS(SHORT), FIXED BIN(15));
```

```
CALL IS$GRQ (lln, connectmessage, config, number, initiator_id, code);
```

Parameters

lln

INPUT → OUTPUT. A pointer to the Low Level Name (LLN) structure of the recipient. This is the LLN supplied by the initiator when requesting this session. You supply a pointer to a blank Low Level Name structure in your program. ISC uses this pointer to write the LLN into your Low Level Name structure. This LLN contains the **ForClientUse** field specified by the session initiator.

connectmessage

INPUT → OUTPUT. A pointer to the Connect message sent by the session initiator. You supply either a null pointer or a pointer to a Message Specifier. ISC uses the pointer you supplied to copy the Connect message into your message array. If you supply a null pointer, no Connect message can be received. The Message Specifier structure is described in Chapter 10. Connect messages are further described in Chapter 12.

config

INPUT → OUTPUT. Either supply a null value or specify a pointer to a blank Session Configuration Block in your program. ISC uses this pointer to copy the global configuration parameter values supplied by the initiator into your Session Configuration Block. Local configuration parameters are not returned. Refer to Chapter 9 for further details.

number

OUTPUT. The recipient's session number assigned to this session.

initiator_id

INPUT → OUTPUT. A pointer to an Initiator Authentication Block, which contains authentication information about the initiator. You supply a pointer to a blank Initiator Authentication Block in your program. ISC uses this pointer to write the initiator's authentication information into your Initiator Authentication Block.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully.

ISC_SC\$LongMessage

You have successfully received a session request and ISC has allocated a session number, but the message array in your program was too small for the Connect message sent with the session request. The Connect message has therefore been truncated.

ISC_SC\$NothingYet

No new session request has arrived.

ISC_SC\$BadMessage

The Message Specifier is invalid. Correct your Message Specifier and call IS\$GRQ again to receive the session request.

ISC_SC\$TooManySessions

You have exceeded your quota for concurrent sessions. The pending session request has been terminated by the system. Before you can get a request for a new session, you must terminate an existing session.

ISC_SC\$BadVersion

You have specified an invalid version number in one of the structures used by this subroutine. All ISC structures must be assigned a version number. Correct the version number and call IS\$GRQ again to receive the session request.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

Discussion

IS\$GRQ is called by the session recipient. It gets a request for an ISC session that was sent to this server. Usually, this subroutine call is issued after a wait on the server's SessionRequestPending synchronizer (described in Chapter 9), so that this subroutine is called upon notification of a Request Session operation. If your server is likely to be involved in multiple concurrent sessions, you probably would not wish to halt all server activity while waiting for a session request. To avoid this, you should place your SessionRequestPending synchronizer in an event group with synchronizers belonging to existing sessions, then wait on the event group.

Upon successful completion, IS\$GRQ returns a session number assigned to this session. You use this session number to perform all subsequent operations during this session.

IS\$GRQ

IS\$GRQ also returns information that was supplied by the session initiator in IS\$RS. It can return a Low Level Name, global configuration parameters, and a Connect message to structures that you defined in your program. You must specify pointers to the appropriate structures to have this information copied into those structures. A pointer to a structure is required for the Low Level Name and Initiator Authentication Block; in the other cases you can specify either a pointer to a structure or a null pointer. If you specify a null pointer, this information is not reported to you.

ISC uses the *initiator_id* pointer to copy information about the session initiator into a blank Initiator Authentication Block in your program. Your program can use this information to determine whether or not to establish a session with the server requesting the session. For example, your program could compare the initiator's user ID with a list of eligible users, or your program could examine the initiator's ACL rights to a particular file (using the CALAC\$ subroutine) to determine if it will accept the session request. The Initiator Authentication Block has the following format:

```
    dcl 1 InitiatorAuthBlock,  
        2 Version fixed bin(15),  
        2 NodeName char(16) var,  
        2 ProjectID char(32) var,  
        2 FullID,  
        3 UserID char(32) var,  
        3 ACLGroupsCount fixed bin(15),  
        3 ACLGroups (32) char(32) var;
```

The fields are defined as follows:

Version

The version number of the structure. Use the ISC_VERSION_NUMBER key value to set this field. ISC_VERSION_NUMBER is supplied in the ISC_STRUCTURES.INS include files.

NodeName

The PRIMENET node name that identifies the system that is running the server.

ProjectID

The ID of the project that the server belongs to. If the session initiator is on a remote node, this field may contain a remote project ID or a null value. Refer to Chapter 13 for details.

FullID

The ID of the initiator's process, including the user ID, a count of the ACL groups that the initiator belongs to, and a list of those ACL groups. If the session initiator is on a remote node, the UserID field may contain a remote ID or a null value. If the UserID field is null, the ACL group information is also null. Refer to Chapter 13 for details.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

IS\$AS

Accepts an ISC session.

Usage

DCL IS\$AS ENTRY (FIXED BIN(15), PTR OPTIONS(SHORT),
PTR OPTIONS(SHORT), PTR OPTIONS(SHORT),
FIXED BIN(15));

CALL IS\$AS (*number*, *connectmessage*, *config*, *syncs*, *code*);

Parameters

number

INPUT. Your session number for the session to be accepted. You get this session number as output from the IS\$GRQ subroutine.

connectmessage

INPUT. Either supply a null value or specify a pointer to a Message Specifier in your program. The Message Specifier should contain a pointer to the Connect message. If specified, this parameter sends a brief Connect message to the session initiator. A null value does not send a Connect message. The Message Specifier structure is described in Chapter 10. Connect messages are further described in Chapter 12.

config

INPUT. Either supply a null value or specify a pointer to a Session Configuration Block in your program. You can use the Session Configuration Block to specify local parameters for your server in this session. The global session parameters have already been established by the initiator using IS\$RS. These local parameters include which synchronizers to establish and their threshold values. Synchronizers are discussed in Chapter 9. A null value sets all of these parameters to default values.

syncs

INPUT → OUTPUT. A pointer to the Session Synchronizers List. You supply a pointer to a blank Session Synchronizers List in your program. IS\$AS creates the synchronizers you specified in *config*, then uses this pointer to write identifiers for these synchronizers into your Session Synchronizers List. Refer to Chapter 9 for a description of this structure.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully.

IS\$AS

ISC_SC\$NoRoom

ISC is unable to perform the operation at this time due to system limitations. Terminate one of your other sessions to free resources. If this problem is recurrent, request that the System Administrator enlarge your allotment of dynamic segments.

ISC_SC\$BadMessage

The Message Specifier is invalid.

ISC_SC\$InvalidConfig

You have requested an invalid configuration for the session. This error is returned if you have specified a queue threshold value that is outside of the range of permitted values or a threshold for a queue that is not configured. It is also returned if you did not set the **Reserved** local configuration parameter to zeros.

ISC_SC\$NoSession

The session number you specified does not refer to an existing session.

ISC_SC\$Established

You have already accepted this session; the current operation is therefore ignored.

ISC_SC\$Terminated

You cannot accept this session because the session has already been terminated, either by the initiator or by the system.

ISC_SC\$BadVersion

You have specified an invalid version number in one of the structures used by this subroutine.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

Discussion

IS\$AS is called by the session recipient. It responds to a request for an ISC session between two servers. You use this subroutine to accept a session request; to reject a session request, use IS\$TS, which terminates the session. To either accept or reject a session, specify the session number assigned to the session, which you received from IS\$GRQ. Upon successful completion of a call to IS\$AS, the session is fully established and available for message exchange.

Before calling IS\$AS, you establish pointers to a Session Configuration Block and a Session Synchronizers List in your program. Upon successful completion of IS\$AS, ISC establishes the local configuration parameters that you specified in the Session Configuration Block, and returns identifiers to your synchronizers in the Session Synchronizers List. If you specify a null value for the Session Configuration Block pointer, IS\$AS establishes the default local

configuration parameters and returns identifiers for the default synchronizers to the Session Synchronizers List. You can specify your own local parameters even if the session initiator has specified a default configuration. Do not specify local parameters that are not supported by the global session configuration.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

IS\$GRS

Gets a response to an ISC session request operation. Upon successful completion, an ISC session is fully established.

Usage

```
DCL IS$GRS ENTRY (FIXED BIN(15), PTR OPTIONS(SHORT),  
                  PTR OPTIONS(SHORT), FIXED BIN(15),  
                  PTR OPTIONS(SHORT), FIXED BIN(15));
```

```
CALL IS$GRS (number, reserved, recipient_id, response, connectmessage, code);
```

Parameters

number

INPUT. The session number that you use to identify this session. This session number was returned by the IS\$RS subroutine that you used to request the session.

reserved

INPUT. A field reserved for system use. Its value must be null.

recipient_id

INPUT → OUTPUT. A pointer to authentication information about the session recipient. You establish a pointer to a blank Target Authentication Block in your program. Upon successful completion, ISC writes the recipient's user ID and node name into this Target Authentication Block. Your program can use this authentication information to confirm that you have established a session with the correct recipient.

response

OUTPUT. A code indicating the status of the session. ISC sets this response code only when IS\$GRS completes successfully, as indicated by the *code* parameter. The possible PL/I values for *response* are:

ISC_RC\$Accepted

The recipient accepted your session request; the session is fully established.

ISC_RC\$ServerTerminate

The recipient did not accept your session request. Instead, the recipient responded with a Terminate Session operation. The recipient may have supplied further information in the message pointed to by *connectmessage*.

ISC_RC\$SystemTerminate

The system was unable to establish the session because the recipient you requested is inaccessible. For example, the recipient server may have exceeded its maximum number of concurrent sessions.

FTN values for *response* are listed in Appendix C.

connectmessage

INPUT → OUTPUT. Either specify a null value or supply a pointer to a Message Specifier in your program. ISC uses this pointer to copy a Connect message sent by the session recipient into a message array in your program. This Connect message could have been sent by either an accept session (IS\$AS) or terminate session (IS\$TS) operation. A null value does not receive a Connect message. The Message Specifier structure is described in Chapter 10. Connect messages are further described in Chapter 12.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully. The status of the session is indicated by the *response* parameter.

ISC_SC\$LongMessage

The operation completed successfully. However, the allocated message array was too small for the Connect message returned by this operation. The Connect message has therefore been truncated.

ISC_SC\$NothingYet

No response to the session request has arrived.

ISC_SC\$BadMessage

The Message Specifier is invalid.

ISC_SC\$NoSession

The session number you specified does not refer to an existing session.

ISC_SC\$Established

You have already performed an IS\$GRS operation for this session; the current operation is therefore ignored.

ISC_SC\$BadVersion

You have specified an invalid version number in one of the structures used by this subroutine.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

IS\$GRS**Discussion**

When the session recipient responds to a session request, ISC posts a notice on the initiator's SessionResponsePending synchronizer. The posting of a notice on this synchronizer informs the initiator to call IS\$GRS. This synchronizer is further described in Chapter 9. Subroutines for retrieving notices from a synchronizer are described in Chapters 2 and 3; use the subroutines described in Chapter 3 if the synchronizer is a member of an event group.

The initiator calls IS\$GRS to determine the outcome of its session request. Upon successful completion, IS\$GRS returns a response code that indicates the status of the session request. This response code indicates whether the session has been accepted or terminated. These response codes are located in the ISC_KEYS.INS include file for your programming language. The response code values for different languages are listed in Appendix C.

Upon successful completion, IS\$GRS returns information about the session recipient. If the session request has been accepted, the Target Authentication Block information returned by this subroutine can be used to confirm that you have established this session with the intended recipient. Target Authentication Block information is also returned if the other server rejected the session request. The Target Authentication Block has the following format:

```
dcl 1 TargetAuthBlock,  
    2 Version fixed bin(15),  
    2 NodeName char(16) var,  
    2 UserID char(32) var;
```

The fields are defined as follows:

Version

The version number of the structure. Use the ISC_VERSION_NUMBER key value to set this field. ISC_VERSION_NUMBER is supplied in the ISC_STRUCTURES.INS include files.

NodeName

The PRIMENET node name that identifies the system that is running the session recipient.

UserID

The user ID of the session recipient. If you are establishing a session with a remote node, this field may contain a remote ID or may be blank. Refer to Chapter 13 for details.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

Session Configuration

Introduction

When you request a session using IS\$RS, you establish a set of configuration parameters for that session. You can either accept the default configuration parameter values or set your own parameter values. If you want to change any of the session's parameters, you must specify your own parameter values for all of the session's parameters. This chapter describes how to set session parameter values, and how to use the synchronizers established during session configuration.

Session parameters are divided into two groups, **global parameters** and **local parameters**. Global session parameters are set by the session initiator (using IS\$RS) to govern both sides of the session. For example, the number of message queues must be the same for the servers on both sides of the session. Local session parameters are set by each server to govern only its side of the session. For example, each server can specify which synchronizers it wishes to use to coordinate the sending and receiving of session messages. Either server can choose default values or specify nondefault values for its own local parameters, regardless of the choice made by the other server. The initiator establishes local parameters using IS\$RS; the session recipient establishes local parameters using IS\$AS.

The Default Session Configuration

To establish session configuration parameters with default values, you make the *config* pointer in IS\$RS or IS\$AS a null pointer. ISC automatically supplies a full set of default configuration values to the session. The Session Configuration Block in your program (if present) is ignored.

ISC establishes the following session configuration as a default:

- Full support for Normal messages with a maximum data part length of 2048 bytes and a maximum control part length of 128 bytes.
- No support for Expedited messages.
- Send and receive queues with a length of 7, no queue thresholds, and ReadyToSend and ReadyToReceive synchronizers to inform you of the status of these queues.
- A SessionResponsePending synchronizer for the session initiator to inform it that the session recipient has responded to the session request.

- An ExceptionPending synchronizer to inform you that an exception has occurred.

A program example that uses the default session configuration is shown in Appendix B.

The Session Configuration Block

To establish session configuration parameters with nondefault values, you set the *config* pointer in IS\$RS or IS\$AS to point to a Session Configuration Block in your program. This block has the following format:

```

dcl 1 SessionConfigurationBlock,
    2 Version fixed bin(15),                /* local */
    2 SessionServices,
        3 NormalService bit(1),             /* global */
        3 ExpeditedService bit(1),          /* global */
        3 SyncsToBeUsed,
            4 ReadyToSend bit(1),            /* local */
            4 ReadyToSendExpedited bit(1),    /* local */
            4 ReadyToReceive bit(1),          /* local */
            4 ReadyToReceiveExpedited bit(1), /* local */
            4 BufferAvailable bit(1),         /* local */
            4 SessionResponsePending bit(1), /* local */
            4 ExceptionPending bit(1),        /* local */
        3 Reserved bit(7),                  /* local */
    2 QueueLengths,
        3 NormalSend fixed bin(15),          /* global */
        3 NormalReceive fixed bin(15),       /* global */
        3 ExpeditedSend fixed bin(15),       /* global */
        3 ExpeditedReceive fixed bin(15),    /* global */
    2 QueueThresholds,
        3 NormalSend fixed bin(15),          /* local */
        3 NormalReceive fixed bin(15),       /* local */
        3 ExpeditedSend fixed bin(15),       /* local */
        3 ExpeditedReceive fixed bin(15),    /* local */
    2 MaxControlLength fixed bin(15),        /* global */
    2 MaxDataLength fixed bin(15),           /* global */
    2 MaxExpeditedLength fixed bin(15),      /* global */
    2 ExistingSessionID fixed bin (15),      /* local */
    2 MessageArea,
        3 BlockSize fixed bin(15),           /* global */
        3 NumberOfBlocks fixed bin(15),      /* global */
        3 Reserved fixed bin(15);           /* global */

```


You can automatically include a template for this structure in your program by using the SYSCOM>ISC_STRUCTURES.INS include file appropriate for your programming language, as described in Chapter 6. Note that if you establish a pointer to a Session Configuration Block, rather than taking default values, you must supply appropriate values for *all* of the parameters in that Session Configuration Block, including those that you wish to set to default values.

You must specify constant values for certain parameters of the Session Configuration Block. These parameters are the **Version** and **Reserved** parameters.

Version

The version number for this structure. You must set this parameter both when using the structure to configure parameters and when using the structure to receive information from ISC. Use the ISC_VERSION_NUMBER to set this field. ISC_VERSION_NUMBER is supplied in the ISC_STRUCTURES.INS include files.

Reserved

There are two reserved parameters, a local parameter, **SessionServices.Reserved**, and a global parameter, **MessageArea.Reserved**. These parameters are reserved for system use. If a nondefault configuration is used, the session initiator must set both reserved parameters to zeros and the session recipient must set the local reserved parameter to zero.

During session establishment, the Session Configuration Block is used as follows:

- The session initiator calls IS\$RS to request a session. It can either use a Session Configuration Block to specify values for all global and local parameters or take the configuration default.
- The session recipient calls IS\$GRQ to get the session request. It can use a Session Configuration Block to see what global parameters have been requested. IS\$GRQ does not copy local configuration parameters into the structure.
- The session recipient calls IS\$AS to accept the session. It can either use a Session Configuration Block to specify values for all of its local parameters or take the configuration default. IS\$AS does not read global parameter values from the structure.

Global Session Parameters

Configuration parameters that are set to the same value for both the session initiator and recipient are referred to as global parameters. Servers use the Session Configuration Block to set both global configuration parameters and local configuration parameters. Global parameter values are set by the session initiator as part of the request session operation (IS\$RS). ISC ignores any global parameter values specified by the session recipient.

Global parameters specify the types and sizes of message queues, the space allocation for message buffers and the maximum sizes for messages. The following are global parameters:

NormalService

A logical value indicating whether or not Normal message service is to be provided on the session. Normal message service provides for the exchange of long messages (called Normal messages), which can consist of a control part and a data part, as described in Chapter 10. Most users will want to configure their session for either Normal message service or Normal message service and Expedited message service. Configuring neither Normal message service nor Expedited message service establishes a session that does not support message exchange, but it allows for the exchange of Connect messages during session establishment. Options for configuring Normal message service are 1 (YES) and 0 (NO). Default is 1 (YES).

ExpeditedService

A logical value indicating whether or not Expedited message service is to be provided on the session. Expedited message service provides for the exchange of short messages (called Expedited messages) on a queue separate from the Normal message queue. Options are 1 (YES) and 0 (NO). Default is 0 (NO).

QueueLengths

ISC establishes two queues for each service: a send and a receive queue for Normal messages and a send and a receive queue for Expedited messages. Each **QueueLengths** parameter configures a matching set of send and receive queues. For example, when you specify a send queue length, ISC configures a send queue of that length (or greater) for the session initiator and a corresponding receive queue of the same length for the session recipient. You use the **QueueLengths** parameters to specify the queue length (number of message slots) to allocate for each of these queues. Values range from 0 through 31 message slots for Normal message service, and 0 through 7 message slots for Expedited message service. If the session is not configured for a service, or if you specify a queue length of 0 for one of the service's queues, no queues for that service are allocated. The default queue length is 7 for all queues.

MaxControlLength

The maximum length for the control part of a Normal message. Valid only if you have specified **NormalService**. An ISC Normal message can consist of two parts, a control part and a data part. The length of a Normal message control part can range from 0 bytes through 128 bytes. The default is 128 bytes. You should define an array in your program that can accommodate a control part of the specified size. This parameter does not govern the maximum size of Expedited messages or Connect messages. The maximum size of an Expedited message is specified in **MaxExpeditedLength**. The maximum size of a Connect message is always 128 bytes.

MaxDataLength

The maximum length for the data part of a Normal message. Valid only if you have specified **NormalService**. An ISC Normal message consists of two parts: a control part and a data part. The length of the data part can range from 0 bytes through 32,630 bytes. A length of 0

prevents the allocation of the space specified in **MessageArea**. Set **MaxDataLength** either to zero or to a multiple of the message area block size. A **MaxDataLength** value that is not a block size multiple is automatically increased to the next block size multiple. The default is 2048 bytes.

MaxExpeditedLength

The maximum length for an Expedited message. Valid only if you have specified **ExpeditedService**. The length of an Expedited message can range from 0 bytes through 64 bytes. The default is 64 bytes.

MessageArea

An area from which message buffers are allocated. These buffers are used for the data part of Normal messages. You should specify a message area large enough to contain the data parts of multiple messages. To specify a message area, you specify the block size and the number of blocks. The block size can range from 128 bytes to 32,630 bytes. The maximum number of blocks you can specify depends on the block size; the total message area may not exceed 262,144 bytes. The default is 100 blocks of 512 bytes each for a total message area of 51,200 bytes.

Local Session Parameters

Session configuration parameters that each server sets for its own use are referred to as local parameters. Each server uses the Session Configuration Block to set its local configuration parameters. The session initiator sets its local parameters as part of the request session operation (IS\$RS). The session recipient sets its local parameters as part of the accept session operation (IS\$AS). The session initiator also uses the Session Configuration Block to set global configuration parameters; the session recipient cannot set global configuration parameters.

The local parameters consist of seven session synchronizers, four queue thresholds, and the ExistingSessionID. Servers must also set the **Version** and **SessionServices.Reserved** parameters to constants when configuring local parameters. The following are the local parameters:

ReadyToSend

A logical value (0 or 1) indicating whether or not to allocate a ReadyToSend synchronizer to this server for the duration of this session. This synchronizer informs the server that there is space available on the Normal message send queue. A value of 1 allocates the synchronizer; a value of 0 does not allocate the synchronizer. The default is 1.

ReadyToSendExpedited

A logical value (0 or 1) indicating whether or not to allocate a ReadyToSendExpedited synchronizer to this server for the duration of this session. This synchronizer informs the server that there is space available on the Expedited message send queue. A value of 1 allocates the synchronizer; a value of 0 does not allocate the synchronizer. The default is 0.

ReadyToReceive

A logical value (0 or 1) indicating whether or not to allocate a ReadyToReceive synchronizer to this server for the duration of this session. This synchronizer informs the server that a message has arrived on the Normal message receive queue. A value of 1 allocates the synchronizer; a value of 0 does not allocate the synchronizer. The default is 1.

ReadyToReceiveExpedited

A logical value (0 or 1) indicating whether or not to allocate a ReadyToReceiveExpedited synchronizer to this server for the duration of this session. This synchronizer informs the server that a message has arrived on the Expedited message receive queue. A value of 1 allocates the synchronizer; a value of 0 does not allocate the synchronizer. The default is 0.

BufferAvailable

A logical value (0 or 1) indicating whether or not to allocate a BufferAvailable synchronizer to this server for the duration of this session. This synchronizer informs the server that message buffer space has become available. The BufferAvailable synchronizer is only notified when space first becomes available after an allocate buffer (IS\$AB) operation failed for lack of available space. A value of 1 allocates the synchronizer; a value of 0 does not allocate the synchronizer. The default is 0.

SessionResponsePending

A logical value (0 or 1) indicating whether or not to allocate a SessionResponsePending synchronizer to this server. This synchronizer informs the session initiator that a response to its session request has arrived. This synchronizer is therefore only meaningful for the server that initiates the session. Session recipients cannot allocate this synchronizer; attempts to do so are ignored. A value of 1 allocates the synchronizer; a value of 0 does not allocate the synchronizer. The default is 1 for the initiator, 0 for the recipient.

ExceptionPending

A logical value (0 or 1) indicating whether or not to allocate an ExceptionPending synchronizer to this server for the duration of this session. This synchronizer informs the server that a session exception has occurred. If this synchronizer is not allocated, ISC posts notices on the server's other synchronizers when a session exception occurs. A value of 1 allocates the synchronizer; a value of 0 does not allocate the synchronizer. The default is 1.

QueueThresholds

These parameters indicate the threshold values for the Normal message send, Expedited message send, Normal message receive, and Expedited message receive queues. A queue threshold defines what type of event causes ISC to post a notice on the corresponding synchronizer. The threshold value specifies the number of messages (on a receive queue) or available message slots (on a send queue) that is required to post a notice on the corresponding synchronizer. If the threshold is 0, ISC posts a notice each time a message appears on your receive queue or an available message slot appears on your send queue. If the threshold is 1 or more, ISC posts a notice when the queue reaches the threshold number of items. ISC posts a notice on the ReadyToReceive synchronizer when the number of pending

messages reaches the threshold number; it does not post notices for additional messages in excess of the threshold number. ISC posts a notice on a ReadyToSend synchronizer when the number of available message slots reaches the threshold number; it does not post notices for additional message slots in excess of the threshold number. If thresholds are used, the usual threshold value is 1 (ISC posts a notice only when the first message or message slot becomes available), but values larger than 1 can be set when you wish to process messages in groups. The minimum threshold value is 0 (threshold not used; ISC posts a notice every time a message is added to the queue or a message slot becomes available); 0 is also the default value. The maximum threshold value is the length of the corresponding queue (a global configuration parameter).

ExistingSessionID

A session number of one of the initiator's existing sessions. If specified, ISC uses the message area of the existing session for message buffering, rather than creating a new message area. If 0, ISC creates a separate message area for the session. The default is 0. This local parameter can only be specified by the session initiator. A session recipient cannot share a message area; ISC ignores any attempt by the session recipient to set this parameter.

Configuring Synchronizers

Each server can configure up to seven ISC synchronizers for each session. These synchronizers inform the server when a particular type of event occurs. For example, the ReadyToReceive synchronizer notifies its server that there is a message available to be received.

Each server can configure its own ISC synchronizers as local parameters during session establishment. Both servers have the option of selecting the default configuration. However, if a server configures any global or local parameters, it must configure all of its parameters, including its synchronizers and queue thresholds. The options for configuring synchronizers are shown in Table 9-1.

Table 9-1
Synchronizer Configuration

<i>Synchronizer Name</i>	<i>Global Parameters Required</i>	<i>Default Configuration</i>	<i>If Not Configured</i>
ReadyToSend	NormalService QueueLengths.NormalSend > 0	allocated	no notice
ReadyToSendExpedited	ExpeditedService QueueLengths.ExpeditedSend > 0	not allocated	no notice
ReadyToReceive	NormalService QueueLengths.NormalReceive > 0	allocated	no notice
ReadyToReceiveExpedited	ExpeditedService QueueLengths.ExpeditedReceive > 0	not allocated	no notice
BufferAvailable	NormalService MaxDataLength > 0 BlockSize > 0 NumberOfBlocks > 0	not allocated	no notice
SessionResponsePending	none	allocated for initiator	no notice
ExceptionPending	none	allocated	notice posted on all other configured synchronizers

The SessionRequestPending Synchronizer

In addition to the seven synchronizers configured as local parameters, PRIMOS automatically provides every server with a SessionRequestPending synchronizer. This synchronizer informs the server that another server has requested a session and is awaiting a response. You can determine the identity of your SessionRequestPending synchronizer by using the SRS\$GN subroutine, described in Chapter 7. Terminating a session does not affect your SessionRequestPending synchronizer. Logging out or issuing an ICE -SERVER command may change the identity of your SessionRequestPending synchronizer.

Deleting Synchronizers

PRIMOS allocates ISC synchronizers for the duration of their useful life within a session. It automatically destroys the SessionResponsePending synchronizer when you successfully call IS\$GRS to establish the session. It automatically destroys your other ISC session synchronizers when you call IS\$TS to terminate the session. You can destroy (or perform any other operation on) non-ISC synchronizers without affecting the ISC session or its synchronizers. Although you can destroy ISC synchronizers during a session, doing so produces unpredictable results and is

not recommended. PRIMOS destroys all of your synchronizers when your process is logged out or when you issue an ICE –SERVER command. The ICE –SERVER command destroys and recreates the SessionRequestPending synchronizer.

Unusable and Unused Synchronizers

If a server attempts to configure an ISC synchronizer which is not supported by the session configuration, the IS\$RS or IS\$AS operation completes successfully, but the synchronizer is not allocated. For example, if a server specifies a ReadyToSendExpedited synchronizer and the global configuration parameters do not support Expedited messages, the synchronizer is not allocated.

If a server configures an ISC synchronizer, it is expected to use that synchronizer. If the server does not use the synchronizer, the number of unretrieved notices on the synchronizer may increase until it reaches 32,767 notices. At that point the synchronizer is inactivated for the duration of the session; you can retrieve notices, but no further notices are posted on that synchronizer.

Session Synchronizers List

The subroutines that you use to configure your session synchronizers (IS\$RS and IS\$AS) return the identifiers for the synchronizers that they configured. ISC writes these identifiers into the Session Synchronizers List structure. This structure has the following format:

```
dcl 1 SessionSyncList,
    2 Version fixed bin(15),
    2 ReadyToSend fixed bin(15),
    2 ReadyToSendExpedited fixed bin(15),
    2 ReadyToReceive fixed bin(15),
    2 ReadyToReceiveExpedited fixed bin(15),
    2 BufferAvailable fixed bin(15),
    2 SessionResponsePending fixed bin(15),
    2 ExceptionPending fixed bin(15);
```

The fields are defined as follows:

Version

The version number of the structure. Use the ISC_VERSION_NUMBER key value to set this field. ISC_VERSION_NUMBER is supplied in the ISC_STRUCTURES.INS include files.

Synchronizer Fields

ISC returns either a synchronizer identifier number or NullSyncNum to each of these seven synchronizer fields. A synchronizer identifier is a system-generated number that uniquely identifies a synchronizer for the duration of a session (or until the synchronizer is destroyed).

A synchronizer identifier is unique within its server. Therefore, if a server participates in multiple concurrent sessions, its identifiers for the synchronizers for each session are different. Synchronizers are numbered from 1. A value of NullSyncNum indicates that the synchronizer was not configured or has been destroyed.

Using Synchronizers

Upon the occurrence of an event, ISC posts a notice on the appropriate synchronizer. This notice indicates that the event has occurred by incrementing the synchronizer's notice count; a notice supplies no other information about the event. Retrieving a notice decrements the synchronizer's notice count.

Retrieving a Notice

If a server wishes to wait for notification of an event, it can call SYN\$WAIT for the synchronizer associated with that event. SYN\$WAIT causes the server to wait until ISC posts a notice to that synchronizer; upon notification, the waiting server automatically retrieves the notice and resumes processing. The server should then respond to the event.

If a server wishes to retrieve a notice when available, it can call SYN\$RTRV. If a notice is present, SYN\$RTRV retrieves the notice and returns a code indicating that fact. The server should then respond to the event. If no notice is present, SYN\$RTRV returns a code indicating that there was no notice to retrieve; the server can then proceed to perform other processing. SYN\$WAIT and SYN\$RTRV are described in Chapter 2; SYN\$GWT and SYN\$GRTR, which wait and retrieve on synchronizers in an event group, are described in Chapter 3.

For example, sending a message (using IS\$SM) posts a notice on the other server's ReadyToReceive synchronizer. If the other server is waiting on that synchronizer, that server automatically retrieves the notice and resumes processing. When processing resumes, the next line of code would ordinarily be a call to IS\$RM to receive the message. If the server is not waiting, it can call a subroutine to retrieve the notice. After the notice is retrieved, the server receives the message by calling IS\$RM. Table 9-2 shows the subroutine associated with each synchronizer. It assumes default queue thresholds.

Table 9-2
Synchronizers and Subroutines

<i>Get Notice on Synchronizer</i>	<i>Then Execute Subroutine</i>
SessionRequestPending	IS\$GRQ
ReadyToSend	IS\$SM
ReadyToSendExpedited	IS\$SM
ReadyToReceive	IS\$RM
ReadyToReceiveExpedited	IS\$RM
BufferAvailable	IS\$AB
SessionResponsePending	IS\$GRS
ExceptionPending	IS\$GE

Retrieving a notice decrements the synchronizer's notice count. It is the programmer's responsibility to maintain the accuracy of this notice count. Therefore, it is extremely important that in every case a notice is retrieved before your program responds to the event that the notice represents.

Using Queue Thresholds

You can use queue threshold values to coordinate notices retrieved and operations performed. (Queue threshold values are set as local session parameters.) The two following examples demonstrate the use of queue thresholds. The first example does not use a queue threshold (that is, threshold = 0). The second example uses a queue threshold.

1. If there is no send queue threshold, the server waits on the ReadyToSend synchronizer before sending each message. It then calls IS\$SM to send the message. This means that the server waits before each send operation. When there is an available send queue slot, the wait ends immediately; when there is no available send queue slot, the server waits until one becomes available. Each call to IS\$SM sends a message; a call to IS\$SM should never fail with a NoRoom error.
2. If there is a send queue threshold of 1, the server calls IS\$SM repeatedly to send messages. When a call to IS\$SM fails with a NoRoom error, the server waits on the ReadyToSend synchronizer. It then calls IS\$SM repeatedly to send messages until the next NoRoom error. This means that the server only waits on the synchronizer when no send queue slot is available; the wait ends when one becomes available. However, sending a message sometimes requires two calls to IS\$SM: a call that fails with a NoRoom error, and a second call after waiting for a send queue slot.

Using Event Groups

ISC synchronizers have all of the features of general-purpose synchronizers, as described in Chapters 2 and 3. Therefore, you can perform waits and timed waits on ISC synchronizers and group ISC synchronizers into event groups.

If a server wishes to wait for any of a number of ISC events, it can create an event group (using `SYN$GCRE`), move several of its synchronizers into the event group (using `SYN$MVTO`) and then wait for a notice to be posted on any of these synchronizers (using `SYN$GWT`). The synchronizers in an event group may belong to different sessions. For example, the server may place the `ReadyToReceive` synchronizers for all of its sessions into a single event group. Another example would be to place the `ReadyToReceive` and `ReadyToReceiveExpedited` synchronizers for a session in an event group and give priority to Expedited messages. Event groups are described in Chapter 3.

Note

Synchronizers placed in an event group are each provided with a `For Client Use` field. This `For Client Use` field is 3 16-bit halfwords in length. You can use this field to identify the session and synchronizer type for each synchronizer in the group, or to provide a pointer to an area containing additional information about the synchronizer. The `For Client Use` field is deleted when you remove the synchronizer from the event group.

You can create additional non-ISC synchronizers during a session by using the synchronizer subroutines described in Chapter 2. You can group ISC synchronizers and non-ISC synchronizers together in the same event group.

Message Exchange

Introduction

Once you have established an ISC session, the two servers can exchange messages. Each server can both send and receive messages, and there is no limit to the number of messages that can be sent during a session. You send a message using IS\$SM; the other server in the session receives the message using IS\$RM. Both servers can send and receive messages simultaneously.

Message exchange can involve Normal messages and Expedited messages. During session establishment, the session initiator configures which of these types of messages will be available. Each type of message has its own queues. If both message types are available, it is recommended that you receive messages from the Expedited message queue before receiving Normal messages.

During message exchange, the following events occur:

- You create the message. If the message has a control part, you write the control part message in an array in your program. If the message has a data part, you allocate a buffer in the session's message area (using IS\$AB), then write the data part of the message into the buffer. You then specify the array and/or buffer in a Message Specifier structure. To send the message, you retrieve a notice from your ReadyToSend synchronizer. This notice indicates that there is available space on your send queue. You then call IS\$SM to send the message, specifying the location of the Message Specifier structure.
- When you send a message, ISC posts a notice on the other server's ReadyToReceive synchronizer. This informs the other server that a message is ready to be received. The other server must perform two operations: first retrieve the notice from the synchronizer, then call IS\$RM to receive the message.
- When you send a message, ISC copies the contents of your control part message array (if present) into a temporary system area. When the other server receives the message, ISC copies the control part of the message from the temporary system area into the message array of the receiving server.
- When you send a message, ISC places location information for the data part message (if present) on your send queue. When the other server receives the message, ISC uses this location information to provide a pointer to the data part message in the recipient's Message Specifier structure.

- After receiving a message that has a data part, the message recipient can free the message buffer (using IS\$FB), or it can reuse the message buffer by writing another message into the buffer and sending that message.

(This listing of events assumes a default session configuration: ReadyToReceive and ReadyToSend synchronizers allocated and no queue thresholds configured. The uses of notices on these synchronizers may differ if you have established queue thresholds. Queue thresholds are described in Chapter 9.)

The size of send and receive queues is established as a global session parameter. The status of these queues is represented by the number of notices posted on synchronizers. Notices posted on a ReadyToReceive synchronizer indicate messages pending on the corresponding receive queue. Notices posted on a ReadyToSend synchronizer indicate available space for sending messages on the corresponding send queue. If no queue thresholds are configured, each notice indicates a single pending message or available message slot. If queue thresholds are configured, a notice indicates a specific number of pending messages or available message slots has been reached. Refer to Chapter 9 for details.

When either server wishes to end the session, the server issues a terminate session operation. This appears as an exception at the other server, informing it of the termination operation. Session termination and exception processing are described in Chapter 11.

Message Exchange Subroutines

Table 10-1 lists the subroutines that allocate and free message buffers and send and receive messages.

Table 10-1
ISC Message Exchange Subroutines

<i>Name</i>	<i>Function</i>
IS\$AB	Allocates a buffer for a message data part
IS\$FB	Frees an allocated data part buffer
IS\$SM	Sends a message
IS\$RM	Receives a message

Creating a Message

Every program that sends or receives a message must contain a Message Specifier. A Message Specifier is a structure that contains the length and location of your message text. You declare a Message Specifier in your program as follows:

```

dcl 1 MessageSpecifier,
    2 Version fixed bin(15),
    2 Control,
        3 SuppliedLength fixed bin(15),
        3 ReturnedLength fixed bin(15),
        3 BufferLocation ptr,
    2 Data,
        3 BufferLength fixed bin(15),
        3 BufferLocation ptr;

```

You declare the Message Specifier in your program and supply the address of that Message Specifier to the *message* or *connectmessage* parameter of the subroutine that sends or receives the message. You can use the same Message Specifier for all messages sent or received by your program, or declare multiple Message Specifiers.

Connect messages, Normal messages, and Expedited messages all use the same Message Specifier structure. A Connect message (sent by IS\$RS, IS\$AS, or IS\$TS) can consist of a control part. A Normal message (sent by IS\$SM) can consist of either a control part, a data part, or both a control part and a data part. An Expedited message (sent by IS\$SM) can consist of a control part. You can send a null message (either Normal or Expedited) by either specifying a null pointer in IS\$SM instead a pointer to a Message Specifier, or specifying a pointer to a Message Specifier and setting both of its **BufferLocation** pointers to null values.

A Message Specifier has the following fields:

Version

The version number of the structure. Use the ISC_VERSION_NUMBER key value to set this field. ISC_VERSION_NUMBER is supplied in the ISC_STRUCTURES.INS include files.

Control

Fields specifying the location and size of a control part message. The control part is a small space allocation (0 to 128 bytes) that you establish as an array in your program. The session initiator establishes the maximum length of the control part for Normal messages and the maximum length of the control part for Expedited messages when requesting the session using IS\$RS. When sending or receiving a message, you must specify the actual length of your message array in the **SuppliedLength** parameter and a pointer to your message array in the **BufferLocation** parameter. When you receive a message, ISC writes the message into the **BufferLocation** array and indicates the actual length of the message in **ReturnedLength**.

Data

Fields specifying the location and size of a data part message. The data part is a large space allocation (0 to 32,630 bytes) that you establish in your session's message area. The session initiator establishes the maximum size of the data part and the total size of the message area when requesting a session using IS\$RS. Before sending a data part message, you must call IS\$AB to allocate a portion of your message area. IS\$AB returns a pointer to a buffer for your message. You use this pointer to write your message into the buffer, then copy this pointer into the **BufferLocation** field of the Message Specifier. When sending a message, you specify the actual length of the message's data part in **BufferLength**; when receiving a message, ISC writes the length of the message's data part into **BufferLength**.

IS\$AB

Allocates a buffer for the data area of a message. Used by either server.

Usage

DCL IS\$AB ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(15))
RETURNS (PTR OPTIONS(SHORT));

buffer = IS\$AB (*number*, *length*, *code*);

Parameters

number

INPUT. The session number that you use to identify this session. This session number was assigned by either the IS\$RS or IS\$GRQ subroutine.

length

INPUT. The length in bytes of the buffer required. Valid lengths are 0 to 32630 bytes. The buffer you allocate should be some multiple of the length of your message area blocks. A length that is not a multiple of the block length is automatically increased to a block length multiple. If you specify a length of zero, this subroutine returns a null pointer.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully. A buffer of the specified size has been allocated.

ISC_SC\$BadSize

The buffer size that you specified is not a valid value.

ISC_SC\$NoSession

The session number you specified does not refer to an existing session.

ISC_SC\$NoMessageArea

The session number you specified does not have an associated message area.

ISC_SC\$NoRoom

There is not enough room in the session's message area to allocate this buffer request. When space becomes available, ISC will notify your BufferAvailable synchronizer (if configured).

IS\$AB**ISC_SC\$NotEstablished**

The session number you specified refers to a session that is not yet fully established.

ISC_SC\$Exception

A session exception has occurred. You cannot allocate a buffer until you have cleared the session exception. Exception processing is discussed in Chapter 11.

ISC_SC\$Terminated

You cannot allocate a buffer because the session has already been terminated.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

buffer

RETURNED VALUE. A pointer to the allocated buffer. This pointer is two 16-bit halfwords in length. Copy this pointer into the **Data.BufferLocation** field of a Message Specifier in your program and use this pointer to write the text of your message into the buffer.

Discussion

IS\$AB allocates a message buffer in the session's message area. You must get a message buffer before sending a message that contains a data part. Only Normal messages can contain a data part. Connect messages and Expedited messages do not have a data part and therefore do not require a buffer allocation. You can get a message buffer either by allocating it (using IS\$AB) or by reusing a message buffer that you received (via IS\$RM) but did not free.

The actual size of the buffer allocated is always either zero (no buffer allocated), or some multiple of the message area block size. The message area block size is a global configuration parameter. If you specify a buffer size that is not a multiple of the block size, ISC increases your allocation to the next whole block size.

This subroutine returns a pointer to the allocated buffer. Use this pointer to write the data part of your message into the buffer. When sending the message, place this pointer value in the **Data.BufferLocation** field of your Message Specifier.

A call to IS\$AB may not be able to allocate a buffer due to the lack of available space in the message area. After this type of buffer allocation failure, ISC posts a notice on the server's BufferAvailable synchronizer when more message area space becomes available. This notice only indicates that buffer space is now available; it does not guarantee that the available space is large enough to allocate the desired buffer. The default configuration does not include the BufferAvailable synchronizer. If no ExceptionPending synchronizer is configured, the occurrence of an exception posts a notice on all of your configured synchronizers, including the BufferAvailable synchronizer.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

IS\$FB

Frees an allocated message buffer in the session's message area.

Usage

DCL IS\$FB ENTRY (FIXED BIN(15), PTR OPTIONS(SHORT), FIXED BIN(15));

CALL IS\$FB (*number*, *buffer*, *code*);

Parameters

number

INPUT. The session number that you use to identify this session. This session number was assigned by either the IS\$RS or IS\$GRQ subroutine.

buffer

INPUT. A pointer to the buffer to be freed. Supply the **Data.BufferLocation** pointer value from the Message Specifier. This pointer value was supplied to the Message Specifier by the receive message (IS\$RM) operation.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully. The buffer has been freed.

ISC_SC\$NoSession

The session number you specified does not refer to an existing session.

ISC_SC\$NoMessageArea

The session number you specified does not have an associated message area.

ISC_SC\$BadBuffer

The *buffer* parameter does not point to a buffer owned by this session.

ISC_SC\$NotEstablished

The session number you specified refers to a session that is not yet fully established.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

IS\$FB

Discussion

IS\$FB frees a message buffer in the session's message area. A message buffer is allocated using IS\$AB.

Sending a message transfers ownership of the message buffer. Only the server receiving a message can free the message buffer for a sent message. However, a server writing a message can use IS\$FB to free a message buffer of an unsent message rather than sending the message.

A server that has received a message buffer has the choice of either freeing the message buffer or reusing the message buffer to send a message. A server can reuse a buffer to send a message back to the the other server in the session or to send a message in another session that shares the same message area.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

IS\$SM

Sends a message to the other server. Used by either server.

Usage

DCL IS\$SM ENTRY (FIXED BIN(15), PTR OPTIONS(SHORT), BIT(1) ALIGNED,
FIXED BIN(15));

CALL IS\$SM (*number*, *message*, *queue*, *code*);

Parameters

number

INPUT. The session number that you use to identify this session. This session number was assigned by either the IS\$RS or IS\$GRQ subroutine.

message

INPUT. A pointer to a Message Specifier in your program. The Message Specifier contains message length information and pointers to the locations that contain the actual message. If you specify a null value for this pointer, ISC sends a null message.

queue

INPUT. A logical value (0 or 1) indicating whether or not to send the message on the Expedited message queue. A value of 0 sends a message on the Normal message queue; a value of 1 sends a message on the Expedited message queue.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully. The message has been sent.

ISC_SC\$NoSession

The session number you specified does not refer to an existing session.

ISC_SC\$NotEstablished

The session number you specified refers to a session that is not yet fully established.

ISC_SC\$NoQueue

A message of the specified type (Normal or Expedited) cannot be sent because no queue has been established for that type of message.

IS\$SM**ISC_SC\$NoRoom**

A message of the specified type (Normal or Expedited) cannot be sent because the send queue for messages of that type is full. You can wait on the appropriate ReadyToSend synchronizer (if configured) until space is available on the queue. The ReadyToSend synchronizers are described in Chapter 9.

ISC_SC\$BadMessage

The Message Specifier is invalid.

ISC_SC\$Exception

A session exception has occurred. You cannot send a message until you have cleared the session exception.

ISC_SC\$Terminated

You cannot send a message on this session because the session has already been terminated, either by the other server or by ISC.

ISC_SC\$BadVersion

You have specified an invalid version number in the Message Specifier.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

Discussion

IS\$SM sends a message from one server in a session to the other server. The message can be either a Normal message or an Expedited message. This subroutine can be used by either server. The other server issues an IS\$RM subroutine to receive the message.

Either server can be the first to send a message once the session has been established. The session recipient can send a message immediately after calling an accept session operation (IS\$AS); it does not have to wait for the initiator to get the session response (using IS\$GRS). The initiator can send a message after calling IS\$GRS.

You can send a message as either a Normal message or an Expedited message (if the session is configured to support these two types of messages). IS\$SM sends Normal messages and Expedited messages on separate queues. These two types of messages are also received on separate queues. Therefore, the other server can receive all of your pending Expedited messages before receiving your Normal messages. This is a useful method for bringing a message to the other server's immediate attention.

To send a message, you specify a pointer to the Message Specifier. The Message Specifier contains the lengths of message buffers and pointers to the control part and the data part of the message. A Normal message can consist of a control part, a data part, both a control part and a data part, or can be a null message containing neither a control part nor a data part. An expedited message can consist of a control part or can be a null message.

You can synchronize the execution of two servers by sending null messages. The Message Specifier for a null message contains null pointers for both the control part and the data part. If no threshold is configured for the receive queue, sending a null message posts a notice on the other server's ReadyToReceive synchronizer.

ISC posts notices on the ReadyToSend and ReadyToSendExpedited synchronizers (if configured) when space becomes available on your send queues. You should retrieve notices from these synchronizers before you use send queue space. Sending a message (of any type or length) occupies one space on a send queue. If the queue has no queue threshold (the default configuration), you should retrieve a notice from the appropriate ReadyToSend synchronizer and then send the message. You can retrieve a notice by either calling SYN\$RTRV for that synchronizer or by waiting on the synchronizer. If the queue has a threshold, you should send messages without retrieving notices until a send operation fails for lack of send queue space. You then wait for ISC to post a notice on the synchronizer, retrieve the notice, and continue sending messages. These options are described in greater detail later in this chapter. It is the programmer's responsibility to maintain the ReadyToSend synchronizers by retrieving a notice (when necessary) before sending a message. These synchronizers and queue thresholds are described in Chapter 9.

Each message that you send occupies one space on your send queue. ISC automatically frees spaces on your send queue as the messages you sent are processed. The number of notices posted on the ReadyToSend synchronizers indicates the availability of space on your send queue. *Do not* interpret the posting of a notice on a ReadyToSend synchronizer as an indication of the status of a previously sent message.

Attempting to send more messages than the number of available spaces on your send queue results in a NoRoom error. If no space is available on your send queue, one possible explanation is that the other server has failed to receive your previous messages. In this case, you must wait until the other server receives one of its pending messages before sending another message on that queue.

After sending a control part message, you can immediately modify or delete your control part message array. You do not have to wait until the other server has received the message. After sending a data part message, you cannot delete your data part message buffer; only the message recipient can free the buffer. After sending a data part, reading or modifying the data part buffer can cause unexpected results and is not recommended.

If the session is configured without an ExceptionPending synchronizer, a notice is posted to your ReadyToSend synchronizer(s) when an exception occurs. The *code* parameter of IS\$SM indicates whether an exception has occurred. Exceptions are further described in Chapter 11.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

IS\$RM

Receives a message. Used by either server.

Usage

**DCL IS\$RM ENTRY (FIXED BIN(15), PTR OPTIONS(SHORT), BIT(1) ALIGNED,
FIXED BIN(15));**

CALL IS\$RM (*number*, *message*, *queue*, *code*);

Parameters

number

INPUT. The session number that you use to identify this session. This session number was assigned by either the IS\$RS or IS\$GRQ subroutine.

message

INPUT → OUTPUT. A pointer to a Message Specifier. You supply a pointer to a Message Specifier in your program. ISC uses the Control.BufferLocation field of the Message Specifier to locate the control part message array and writes the control part of the message into that array. ISC also modifies the parameter values of the Message Specifier. Upon successful completion, the Message Specifier contains the lengths of the message's control and data parts, and pointers to the locations of these parts.

queue

INPUT. A logical value (0 or 1) indicating the queue from which you wish to receive a message. A value of 0 receives a message from the Normal message queue; a value of 1 receives a message from the Expedited message queue. The default is 0.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully. A message of the specified type has been received.

ISC_SC\$LongMessage

A message has been successfully received. However, the length of the control part of the message exceeds the available message array. The control portion of the message has therefore been truncated.

ISC_SC\$NothingYet

No message of the specified type has arrived. This code only tells you about the message type you specified (Normal or Expedited).

ISC_SC\$NoSession

The session number you specified does not refer to an existing session.

ISC_SC\$NotEstablished

The session number you specified refers to a session that is not yet fully established.

ISC_SC\$NoQueue

Messages of the specified type (Normal or Expedited) cannot be received because no queue was configured for that type of message. Queues are configured by the session initiator using IS\$RS.

ISC_SC\$BadMessage

The Message Specifier is invalid.

ISC_SC\$Exception

A session exception has occurred. You cannot receive a message until you have cleared the session exception.

ISC_SC\$Terminated

You cannot receive a message on this session because the session has already been terminated, and all message queues are empty.

ISC_SC\$BadVersion

You have specified an invalid version number in your Message Specifier.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

Discussion

IS\$RM receives a message from the other server in the specified session. The message received by this subroutine was sent by the other server using IS\$SM; the message can be either a Normal message or an Expedited message. Each time you call IS\$RM, you must specify whether you wish to receive a Normal message or an Expedited message.

ISC posts notices on your ReadyToReceive and ReadyToReceiveExpedited synchronizers (if configured) when messages are pending on your receive queues. A message (of any type or length) occupies one space on a receive queue.

ISC posts a notice on your ReadyToReceive synchronizer when a Normal message is ready to be received. Retrieve the notice, then issue a call to IS\$RM and specify a value of 0 for the *queue* parameter.

If the session is configured for Expedited messages, ISC posts a notice on your ReadyToReceiveExpedited synchronizer when an Expedited message is ready to be received.

IS\$RM

Retrieve the notice, then issue a call to IS\$RM and specify a value of 1 for the *queue* parameter.

If the receive queue has no queue threshold (the default configuration), ISC posts a notice for each pending message. You should retrieve a notice from the appropriate ReadyToRetrieve synchronizer and then receive the message. You can retrieve a notice by either calling SYN\$RTRV for that synchronizer or by waiting on the synchronizer.

If the queue has a threshold, the notice informs you that the threshold number of messages are pending on the receive queue. You should retrieve the notice, then receive all of the messages on the queue.

It is the programmer's responsibility to maintain the ReadyToReceive synchronizers by retrieving a notice (when necessary) before receiving a message. These synchronizers and queue thresholds are described in Chapter 9.

When there are messages of both types waiting to be received, you should receive all Expedited messages before receiving Normal messages. You can easily perform this task if you have placed the ReadyToReceive and the ReadyToReceiveExpedited synchronizers in an event group and given the ReadyToReceiveExpedited synchronizer a higher priority. These ISC synchronizers are further described in Chapter 9; retrieving a notice from a synchronizer is described in Chapter 2; event groups are described in Chapter 3.

If the session is configured without an ExceptionPending synchronizer, a notice is posted to your ReadyToReceive synchronizer(s) when an exception occurs. The *code* parameter of IS\$RM indicates whether an exception has occurred. Exceptions are further described in Chapter 11.

If the session is configured without the ReadyToReceive synchronizer(s), you can still use IS\$RM to receive messages. If a message is pending, IS\$RM receives the message; if no message is pending IS\$RM returns a *code* parameter value of ISC_SC\$NothingYet. You should receive all Expedited messages before receiving Normal messages.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

Session Termination and Exceptions

Introduction

When either server wishes to end the message exchange session, that server calls IS\$TS to terminate its side of the session. The other server receives this termination operation as an exception. The server that receives the exception can either immediately terminate its side of the session or clear the exception and continue processing. When all processing is done, the remaining server should also call IS\$TS to terminate its side of the session.

This chapter describes how to terminate a session and how to analyze and clear an exception. An exception can occur when the other server terminates its side of the session, when the system terminates the session, or when a network transmission failure occurs.

An exception prevents a server from proceeding with that session until the exception is cleared. ISC indicates that an exception has occurred by posting a notice on the server's ExceptionPending synchronizer. If the server attempts to perform an ISC operation while an exception is pending, the operation returns an exception status code.

To respond to an exception, you first call IS\$GE to get the exception (determine its nature) and then call IS\$CE to clear the exception. If the exception did not terminate the other server's side of the session, you can clear the exception and proceed with ISC message exchange. If the exception describes a condition that terminated the other server's side of the session, you cannot send any further messages to that server. After clearing an exception, the remaining server can continue to read messages waiting on its receive queues. After reading all queued messages, the remaining server must also call IS\$TS to complete session termination.

Termination Subroutines

Table 11-1 lists the subroutines that terminate a session and respond to an exception.

Table 11-1
Termination Subroutines

<i>Name</i>	<i>Function</i>
IS\$TS	Terminates the caller's side of a session
IS\$GE	Gets an exception
IS\$CE	Clears an exception

The ExceptionPending Synchronizer

ISC provides an optional ExceptionPending synchronizer that permits you to determine if an exception has occurred. If you configured your session with an ExceptionPending synchronizer, an exception posts a notice upon that synchronizer. Other synchronizers are not affected. If, however, you do not have an ExceptionPending synchronizer, ISC treats the exception as an ordinary event and posts a notice to all of your existing ISC synchronizers (except the SessionRequestPending synchronizer). In this case, the number of notices on each synchronizer appears to be incremented by one, due to the exception being posted as an event.

It is suggested that you place your ExceptionPending synchronizer in an event group with other ISC synchronizers for that session and give ExceptionPending a higher priority. Event groups are described in Chapter 3.

IS\$TS

Terminates an ISC session or halts the process of session establishment. Used by either server.

Usage

DCL IS\$TS ENTRY (FIXED BIN(15), FIXED BIN(15), PTR OPTIONS(SHORT),
FIXED BIN(15));

CALL IS\$TS (*number*, reserved, *connectmessage*, *code*);

Parameters

number

INPUT. The session number that you use to identify this session. This session number was assigned by either the IS\$RS or IS\$GRQ subroutine.

reserved

A field reserved for system use. Its value is ignored.

connectmessage

INPUT. Either supply a null value or specify a pointer to a Message Specifier in your program. If specified, this parameter sends a brief Connect message to the other server as part of session termination. A null value does not send a Connect message. The Message Specifier structure is described in Chapter 10. Connect messages are further described in Chapter 12.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully. The session has been terminated.

ISC_SC\$NoRoom

The terminate message cannot be sent due to system limitations.

ISC_SC\$BadMessage

The Message Specifier is invalid.

ISC_SC\$NoSession

The session number you specified does not refer to an existing session.

ISC_SC\$BadVersion

You have specified an invalid version number in your Message Specifier.

IS\$TS

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

Discussion

IS\$TS terminates an ISC session. It can be issued at any time by either server participating in the session. You can use IS\$TS to terminate a session during session establishment or during message exchange. This subroutine must be called by both servers to fully terminate a session.

To terminate a session during session establishment, the session recipient can issue a Terminate Session (IS\$TS), rather than an Accept Session (IS\$AS) subroutine call. When the session initiator calls IS\$GRS, the subroutine indicates that the session has been terminated and returns the IS\$TS Connect message. The session initiator can also call IS\$TS to terminate a session at any point during session establishment.

When you terminate an established session, ISC posts a notice on the other server's ExceptionPending synchronizer. The other server must call IS\$GE to get this exception, then call IS\$CE to clear the exception before it can proceed to read messages remaining on the receive queues. The other server receives the IS\$TS Connect message when it calls IS\$GE to get the exception. After a server calls IS\$TS, it can neither send nor receive further messages. The other server participating in the session can continue to receive messages that are waiting on its receive message queues.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

IS\$GE

Gets an exception. Used by either server.

Usage:

```
DCL IS$GE ENTRY (FIXED BIN(15), FIXED BIN(15), PTR OPTIONS(SHORT),  
                FIXED BIN(15));
```

```
CALL IS$GE (number, exception, connectmessage, code);
```

Parameters

number

INPUT. The session number that you use to identify this session. This session number was assigned by either the IS\$RS or IS\$GRQ subroutine.

exception

OUTPUT. A code indicating the type of exception that occurred. This code is only set upon successful completion of this subroutine call. Possible values are:

ISC_EX\$DeliveryFailure

Possible data loss. You may continue the session and request that the message (or messages) be sent again. This exception is caused by a network failure during a remote ISC session. Refer to Chapter 13 for further details.

ISC_EX\$ServerTerminate

This session has been terminated by the other server.

ISC_EX\$SystemTerminate

This session has been terminated by the system.

connectmessage

INPUT → OUTPUT. A pointer to the Connect message sent by the session terminator. You supply either a null value or a pointer to a Message Specifier. ISC uses the pointer you supplied to copy the Connect message into your message array. The Message Specifier structure is described in Chapter 10. Connect messages are further described in Chapter 12.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully. The *exception* parameter indicates the type of exception received.

IS\$GE

ISC_SC\$LongMessage

An exception has been successfully received. However, the length of the Connect message exceeds the length of the message array. This message has therefore been truncated.

ISC_SC\$NothingYet

No exception is pending on this session.

ISC_SC\$NoSession

The session number you specified does not refer to an existing session.

ISC_SC\$Terminated

You cannot receive an exception on this session because the session has already received a termination exception.

ISC_SC\$BadMessage

The Message Specifier is invalid.

ISC_SC\$NotEstablished

The session number you specified refers to a session that is not yet fully established.

ISC_SC\$BadVersion

You have specified an invalid version number in your Message Specifier.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

Discussion

IS\$GE gets an exception that is pending for the current session. You perform this operation after receiving a notice on your ExceptionPending synchronizer or receiving a status code of ISC_SC\$Exception from a Send Message (IS\$SM), Receive Message (IS\$RM), or Allocate Buffer (IS\$AB) subroutine call. Before you can successfully perform any of these operations, you get the exception, using this subroutine, and then you must clear the exception, using IS\$CE.

If the exception is a ServerTerminate, IS\$GE can receive the Connect message sent by the other server. No Connect message is returned for SystemTerminate or DeliveryFailure exceptions.

Getting an exception enables you to determine the type of exception that occurred. It is possible, though not generally advisable, to use IS\$CE to clear an exception without first using this subroutine to get the exception. It is not necessary to either get or clear an exception if you plan to immediately terminate the session.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

IS\$CE

Clears an exception. Used by either server.

Usage

```
DCL IS$CE ENTRY (FIXED BIN(15), FIXED BIN(15));
```

```
CALL IS$CE (number, code);
```

Parameters

number

INPUT. The session number that you use to identify this session. This session number was assigned by either the IS\$RS or IS\$GRQ subroutine.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully.

ISC_SC\$NothingYet

No pending exception currently exists.

ISC_SC\$NoSession

The session number you specified does not refer to an existing session.

ISC_SC\$Terminated

You cannot clear an exception on this session because you have already cleared a termination exception.

ISC_SC\$NotEstablished

The session number you specified refers to a session that is not yet fully established.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

IS\$CE

Discussion

IS\$CE clears an exception that is pending for the current session. You perform this operation after receiving a notice on your ExceptionPending synchronizer or receiving a status code of ISC_SC\$Exception from a Send Message (IS\$SM), Receive Message (IS\$RM), or Allocate Buffer (IS\$AB) subroutine call.

Normally, you first call IS\$GE to get the exception, then call IS\$CE to clear the exception. Getting the exception enables you to determine the type of exception that occurred, and to receive an optional termination message issued by the other server. It is possible, though not generally advisable, to use IS\$CE to clear an exception without first using IS\$GE to get the exception.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

Connect Messages

Introduction

ISC permits you to send and receive brief messages while establishing or terminating a session. These messages are known as **Connect messages**. The following subroutines permit you to send and receive Connect messages:

<i>Sent By</i>	<i>Received By</i>
IS\$RS	IS\$GRQ
IS\$AS	IS\$GRS
IS\$TS	IS\$GE or IS\$GRS

In order to send or receive a Connect message, your program must contain a Message Specifier, and the **Control.BufferLocation** field of that structure must contain a pointer to an array of a maximum of 128 bytes. Refer to Chapter 10 for further details on how to include this structure in your program.

To send a Connect message, you write the message into your array, put the actual length of the message and pointer to the array in your Message Specifier, then specify a pointer to your Message Specifier in the *connectmessage* parameter of the subroutine.

To receive a Connect message, you specify the location and length of a blank array in your Message Specifier, and specify a pointer to your Message Specifier in the *connectmessage* parameter of the subroutine. ISC copies the Connect message into your message array.

ISC copies a Connect message into a temporary buffer when you send the message, then copies the Connect message from this temporary buffer when the other server receives the Connect message. Therefore, you can send a Connect message and then delete or overwrite your message array. You do not have to wait for the other server to receive the Connect message before reusing the array.

Unlike Normal messages and Expedited messages, a delivery failure of a Connect message in an IS\$TS call is invisible. That is, when you issue an IS\$TS call your session is ended. No acknowledgment that the Connect message was successfully delivered can be returned. A delivery failure can also delete Connect messages sent during session establishment.

A Connect Message Exchange

You can use Connect messages for message exchange, rather than establishing a session and exchanging Normal messages or Expedited messages. This strategy is useful if the two servers only need to send one brief message apiece. A Connect message exchange is performed as follows:

- A server sends a Connect message in IS\$RS (Request Session).
- The other server responds by sending a Connect message in IS\$TS (Terminate Session), which also rejects the session request.

Connect message exchange does not require an established session, and therefore message exchange can be performed with fewer preliminary steps. Connect message exchange does not require the establishment of either Normal or Expedited message service, a message area, queues, or synchronizers (except SessionResponsePending). It therefore requires fewer system resources. However, if there is a possibility that the recipient server will not respond to the session request by terminating (with IS\$TS), but instead accept the session (with IS\$AS), the session initiator should configure the session's global parameters to enable the exchange of Normal messages and/or Expedited messages.

Remote Sessions

Introduction

You can establish a session with a server on the same node, or with a server on another node. A session with a server located on another node is known as a **remote session**. You perform the same ISC operations for local sessions and remote sessions; ISC automatically handles the transmission of messages across the network.

The Low Level Name (LLN) that you use to identify the session recipient contains both the server name and its node name. Each LLN is therefore unique throughout the network. When specifying the LLN for a local session, you can specify the node name as null; when specifying the LLN for a remote session, you must specify the name of the remote node.

ISC supports remote sessions through PRIMENET and an auxiliary process called the `ISC_NETWORK_SERVER`. Both of these facilities must be running on both nodes before you can establish a remote session; neither of these facilities are required for ISC sessions between servers on the same node. ISC uses the `ISC_NETWORK_SERVER` on each node as an intermediary to handle PRIMENET communications. Under normal circumstances, this use of an intermediary server is completely transparent to users of ISC. Establishing and maintaining the `ISC_NETWORK_SERVER` is described in the *Operator's Guide to Prime Networks*.

Remote User ID

When you establish a session, ISC places the user ID of the other server in your authentication block. For a local session, this value is always the user ID. The user ID value for a remote server may be either that server's local ID on that node, a remote ID, or a null value.

- If the remote node does not force user validation, and the server has not established a remote ID, your authentication block receives that server's local user ID, project ID, and ACL group information.
- If the remote server has used the Add Remote ID (ARID) command to establish a remote ID for the current node, your authentication block receives the remote ID information that the server has established for your node. Refer to the *User's Guide to Prime Network Services* for further details on the use of the ARID command.

- If the Network Administrator has configured the remote node to force user validation, and the remote server has not established a remote ID for your node, the authentication block receives null values for the **UserID**, **ProjectID**, and **ACLGroups** fields. Your server must then decide whether or not to establish a session with this anonymous server. Refer to the *PRIMENET Planning and Configuration Guide* for further details on forcing user validation.

Network Events

ISC uses virtual circuits to connect the two PRIMENET nodes involved in a remote session. It establishes a virtual circuit for Normal messages and, if required, a second virtual circuit for Expedited messages. These two virtual circuits are coordinated so that if either one of them becomes inoperable, ISC terminates the session, clearing both virtual circuits. These virtual circuits can experience the following events:

Clear

If the virtual circuit is cleared, both servers receive a `SystemTerminate` exception. The servers can clear this exception and continue to read previously queued messages, but no new messages can be sent by either server. Each of the servers must call `IS$TS` to terminate the session.

Reset

If the virtual circuit is reset, a `DeliveryFailure` exception is posted to the server currently sending a message. This exception indicates that one or more messages being transmitted *may* have been lost. The server can clear this exception and retransmit the message(s). Programmers performing ISC remote sessions should establish procedures for recognizing and recovering from data loss following a `DeliveryFailure` exception. Resetting a virtual circuit does not otherwise affect the contents of send and receive queues.

Retrieving Session Information

Introduction

This chapter describes four subroutines that you can use to monitor ISC message exchange sessions:

Table 14-1
ISC Monitoring Subroutines

<i>Name</i>	<i>Function</i>
IS\$GSO	Gets sessions owned by your server
IS\$GSA	Gets session attributes
IS\$GSS	Gets session status
IS\$STA	Gets statistics about a session

The IS\$GSO subroutine can be called at any time; it returns the session numbers of the server's active sessions (if any).

IS\$GSA, IS\$GSS and IS\$STA can be called during an active session. IS\$GSA and IS\$GSS can also be called during session establishment; IS\$STA cannot be called until the session has been fully established. All three of these subroutines can be called after the other server has terminated, but not after your server has called IS\$TS to terminate the session.

IS\$GSO

Gets a list of sessions owned by this server.

Usage

```
DCL IS$GSO ENTRY (FIXED BIN(15), PTR OPTIONS(SHORT), FIXED BIN(15),  
                  FIXED BIN(15));
```

```
CALL IS$GSO (length, sessions, count, code);
```

Parameters

length

INPUT. The length (in 16-bit halfwords) of the array that is to hold the session numbers of all the currently active sessions for this server.

sessions

INPUT → OUTPUT. A pointer to an array in your program. You supply a pointer to an array of size *length* in your program. ISC uses this pointer to copy the session numbers of your currently active sessions into this array.

count

OUTPUT. A count of the number of active sessions for this server. A count of the actual number of sessions is returned here, even if the array pointed to by *sessions* is not large enough to hold all of the session numbers. A value of 0 indicates that the server currently has no sessions.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully.

ISC_SC\$TooShort

The array length specified in *length* is too small to contain all of the server's session numbers. The total number of active sessions is returned to *count*; no session numbers are returned to *sessions*.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

Discussion

IS\$GSO copies the session numbers of all of your currently active sessions into an array in your program. You must define an array in your program large enough to hold these session numbers, and then specify the length of this array to the subroutine. If the array is not large enough, ISC returns the total number of sessions to *count*, but does not return any session numbers to the array pointed to by *sessions*. If the array is large enough, ISC copies the session numbers into the array pointed to by *sessions* and returns the total number of sessions to *count*.

The list of sessions returned by this subroutine includes sessions that are currently being established or terminated. Sessions are listed in random order.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

IS\$GSA

Gets the attributes of a session.

Usage

```
DCL IS$GSA ENTRY (FIXED BIN(15), PTR OPTIONS(SHORT),  
                  PTR OPTIONS(SHORT), PTR OPTIONS(SHORT),  
                  FIXED BIN(15));
```

```
CALL IS$GSA (number, config, syncs, identity, code);
```

Parameters

number

INPUT. The session number that you use to identify this session. This session number was assigned by either the IS\$RS or IS\$GRQ subroutine.

config

INPUT → OUTPUT. Either a null pointer or a pointer to a Session Configuration Block in your program. A Session Configuration Block contains the global configuration parameters for the session and your server's local configuration parameters. You supply a pointer to a blank Session Configuration Block in your program. ISC uses this pointer to write your configuration parameter values into your Session Configuration Block. If you specify a null pointer, this parameter returns no information.

syncs

INPUT → OUTPUT. Either a null pointer or a pointer to a Session Synchronizers List in your program. A Session Synchronizers List contains the identifiers of the synchronizers established for your server. You supply a pointer to a blank Session Synchronizers List in your program. ISC uses this pointer to write identifiers for your synchronizers into your Session Synchronizers List. If you specify a null pointer, this parameter returns no information.

identity

INPUT → OUTPUT. Either a null pointer or a pointer to an Attribute Identity Block in your program. An Attribute Identity Block provides information about the other server participating in this session. You supply a pointer to a blank Attribute Identity Block in your program. ISC uses this pointer to write the server information into your Attribute Identity Block. If you specify a null pointer, this parameter returns no information.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully.

ISC_SC\$NoSession

The session number you specified does not refer to an existing session.

ISC_SC\$BadVersion

You have specified an invalid version number in one of the structures used by this subroutine.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

Discussion

IS\$GSA can return session configuration information, synchronizer identifiers, and the identity attributes of the other server participating in the session. You can choose to receive only some of this information by specifying a null pointer for one or more of the subroutine's parameters.

IS\$GSA can return information to an Attribute Identity Block. This information includes the attributes of the server at the other end of the session and which server initiated the session. The Attribute Identity Block has the following format:

```
dcl 1 AttributeIdentityBlock,  
    2 Version fixed bin(15),  
    2 NodeName char(16) var,  
    2 UserID char(32) var,  
    2 IAmInitiator bit(1) aligned,  
    2 TargetServerName char(12) var;
```

The fields are defined as follows:

Version

The version number of the structure. Use the ISC_VERSION_NUMBER key value to set this field. ISC_VERSION_NUMBER is supplied in the ISC_STRUCTURES.INS include files.

NodeName

The PRIMENET node name that identifies the system that is running the other server.

UserID

The user ID of the other server. Returns a remote ID if the other server is on another node and has established a remote ID using the ARID command.

IS\$GSA**IAmInitiator**

A logical value. Returns 1 if the server calling IS\$GSA is the session initiator and 0 if it is the session recipient.

TargetServerName

If the server calling IS\$GSA is the session initiator, **TargetServerName** returns the server name of the other (recipient) server. If it is the session recipient, the value of **TargetServerName** is undefined.

This subroutine also returns the session configuration and the identifiers for your session synchronizers. The attributes displayed by this subroutine are the actual session attributes, which may differ slightly from the attributes requested by IS\$RS or IS\$AS. For example, ISC may allocate a queue larger than the one requested in IS\$RS.

If you call this subroutine during session establishment, some items of information may not be established yet or may not be available to the caller. These items are represented as zeros or blanks in the returned structures.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

IS\$GSS

Gets status information about a session.

Usage

DCL IS\$GSS ENTRY (FIXED BIN(15), PTR OPTIONS(SHORT), FIXED BIN(15));

CALL IS\$GSS (*number*, *status*, *code*);

Parameters

number

INPUT. The session number that you use to identify this session. This session number was assigned by either the IS\$RS or IS\$GRQ subroutine.

status

INPUT → OUTPUT. A pointer to a Session Status Block, which contains status information about the current session. You supply a pointer to a blank Session Status Block in your program. ISC uses this pointer to write status information into your Session Status Block.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully.

ISC_SC\$NoSession

The session number you specified does not refer to an existing session.

ISC_SC\$BadVersion

You have supplied an invalid version number in your Session Status Block.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

Discussion

IS\$GSS returns information about the current status of the session in a Session Status Block. The Session Status Block has the following format:

```
dcl 1 SessionStatusBlock,  
  2 Version fixed bin(15),  
  2 Phase fixed bin(15),  
  2 ExceptionsToBeCleared fixed bin(15),  
  2 MessageAreaUsers fixed bin(15),  
  2 CurrentQueueStatus,  
    3 NormalSend fixed bin(15),  
    3 NormalReceive fixed bin(15),  
    3 ExpeditedSend fixed bin(15),  
    3 ExpeditedReceive fixed bin(15);
```

Upon successful completion, ISC assigns values to the fields of this structure. These field values refer to the status of your side of the session, not to the overall session status. All items are counted from 1. The fields are defined as follows:

Version

The version number of the structure. Use the ISC_VERSION_NUMBER key value to set this field. ISC_VERSION_NUMBER is supplied in the ISC_STRUCTURES.INS include files.

Phase

The current phase of your side of the session. This field can take the following PL/I values:

ISC_PC\$Establishing

Session is being established.

ISC_PC\$DataTransfer

Session has been established. This phase begins for the session recipient when it successfully calls IS\$AS. This phase begins for the session initiator when it successfully calls IS\$GRS.

ISC_PC\$Terminating

Session is being terminated. Either the other server has called IS\$TS (Terminate Session), or the system has terminated the other server.

FTN values for these phase codes are listed in Appendix C.

ExceptionsToBeCleared

This field contains the number of outstanding exceptions pending on your server for this session. This exception count is independent of the ExceptionPending synchronizer.

MessageAreaUsers

This field contains the number of your server's sessions that are currently assigned to this session's message area. A message area is shared by using the **ExistingSessionID** field when configuring the session. A value of 1 indicates that this session is the only session using this message area. A value of 0 indicates that no message area is configured for this session.

CurrentQueueStatus

These fields contain the number of messages currently on the receive queues and the number of message slots available on the send queues that are configured to your server for this session. If a queue is not configured, ISC returns a value of 0 to the corresponding **CurrentQueueStatus** field.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

IS\$STA

Gets statistics about the current session.

Usage

DCL IS\$STA ENTRY (FIXED BIN(15), PTR OPTIONS(SHORT), FIXED BIN(15));

CALL IS\$STA (*number*, *statistics*, *code*);

Parameters

number

INPUT. The session number that you use to identify this session. This session number was assigned by either the IS\$RS or IS\$GRQ subroutine .

statistics

INPUT → OUTPUT. A pointer to the Session Statistics Block, which contains current statistics for this session. You supply a pointer to a blank Session Statistics Block in your program. ISC uses this pointer to write statistics gathered since the beginning of the session into your Session Statistics Block.

code

OUTPUT. The status code. The possible codes are:

ISC_SC\$OK

The operation completed successfully.

ISC_SC\$NoSession

The session number you specified does not refer to an existing session.

ISC_SC\$NotEstablished

The specified session has not been fully established.

ISC_SC\$BadVersion

You have supplied an invalid version number in your Session Statistics Block.

ISC_SC\$SoftwareError

An unexpected software error has occurred. Report any occurrence of this error to the System Administrator.

Discussion

IS\$STA can be called only while a server is in the DataTransfer or Terminating phases of a session, as indicated by IS\$GSS. It returns statistics compiled over the duration of the current session to a Session Statistics Block. The Session Statistics Block has the following format:

```
dcl 1 SessionStatisticsBlock,  
  2 Version fixed bin(15),  
  2 NormalMessages,  
    3 Sent fixed bin(31),  
    3 Received fixed bin(31),  
    3 FailedSends fixed bin(31),  
    3 FailedReceives fixed bin(31),  
  2 ExpeditedMessages,  
    3 Sent fixed bin(31),  
    3 Received fixed bin(31),  
    3 FailedSends fixed bin(31),  
    3 FailedReceives fixed bin(31),  
  2 Exceptions,  
    3 Count fixed bin(31),  
  2 Allocations,  
    3 Failed fixed bin(31),  
    3 AverageBufferSize fixed bin(15),  
  2 MessageAreaInfo,  
    3 CurrentAreaUsage fixed bin(15),  
    3 MaxAreaUsage fixed bin(15);
```

Upon successful completion, ISC assigns values to the fields of this structure. These statistics refer to the activities performed by your server during the session, not to overall session activity. All items are counted from 1. The fields are defined as follows:

Version

The version number of the structure. Use the ISC_VERSION_NUMBER key value to set this field. ISC_VERSION_NUMBER is supplied in the ISC_STRUCTURES.INS include files.

NormalMessages and ExpeditedMessages

These fields count the number of times your server has performed the specified operation during the current session. A failed operation is one that could not be performed because there was either no message to receive from the queue, or no space available on the queue to send a message.

Exceptions

This field counts the number of exceptions (delivery failures and terminations) received by your server during this session.

Allocations

These fields count the number of failed buffer allocation attempts, and the average size (in bytes) of the buffers your server has successfully allocated during this session.

MessageAreaInfo

These fields contain the percent of the message area that your server is currently using, and the maximum percent that it has used during this session.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

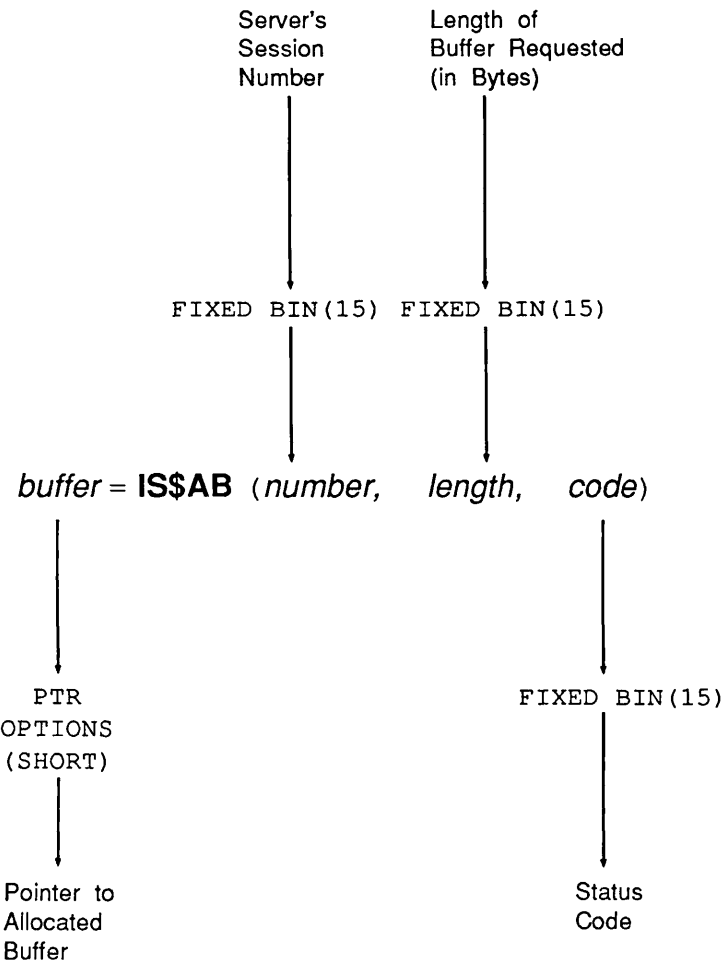
Quick Reference to Calling Sequences

The following figures illustrate the calling sequence for the subroutines described in this volume. Subroutines are listed in alphabetical order.

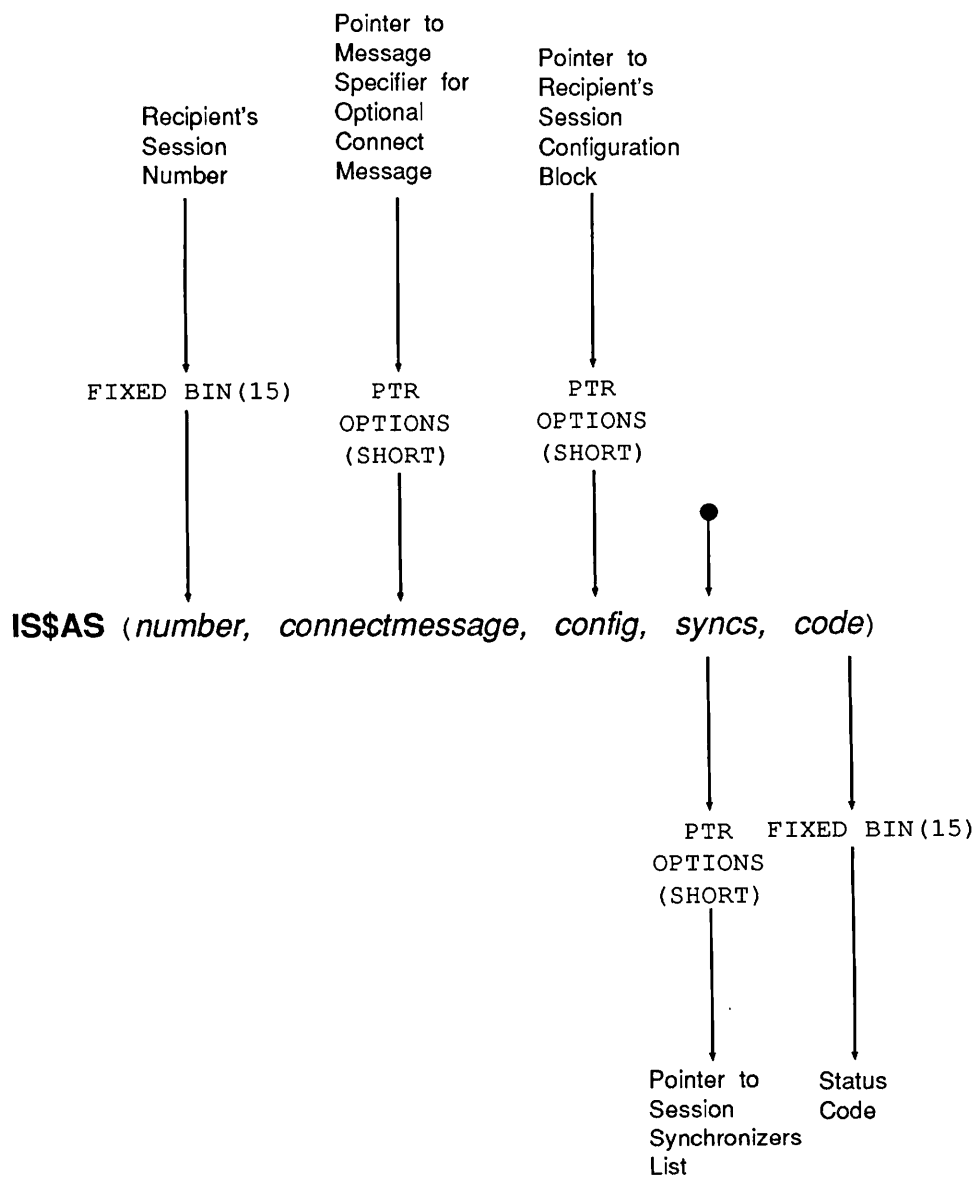
The illustrations show input parameters above the subroutine call and output parameters below the subroutine call. A special graphic, shown as a dot and arrow, indicates a pointer to an area in your program. You supply this pointer as an input parameter and the subroutine uses this pointer to write information into the area it points to. The subroutine returns the pointer as an output parameter. The subroutine call does not modify the pointer itself.

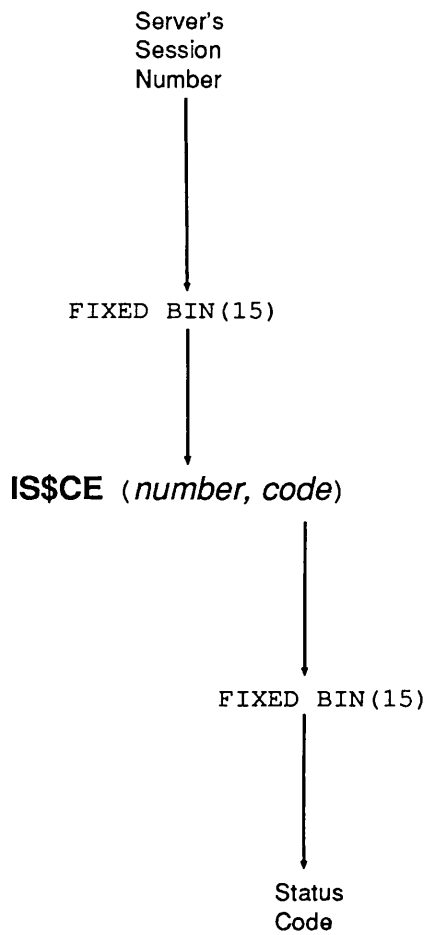
IS\$AB

Allocate an ISC Message Buffer

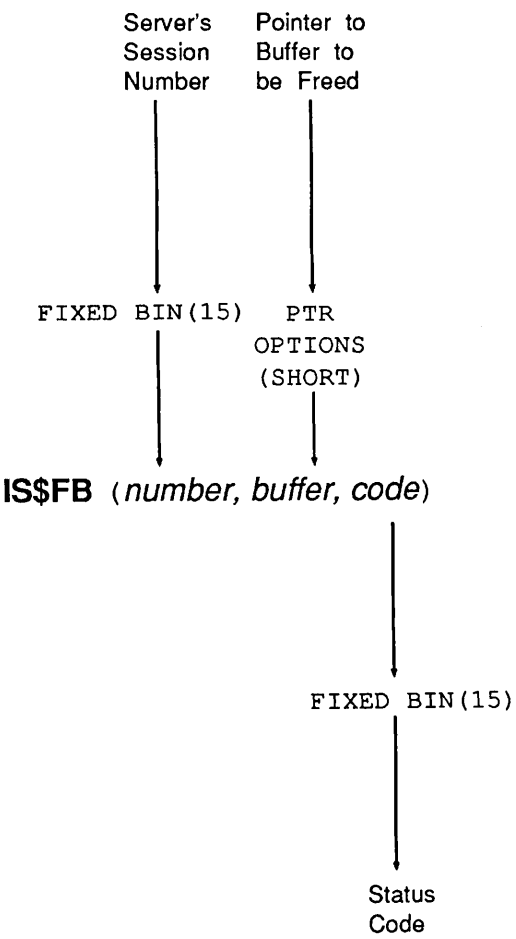


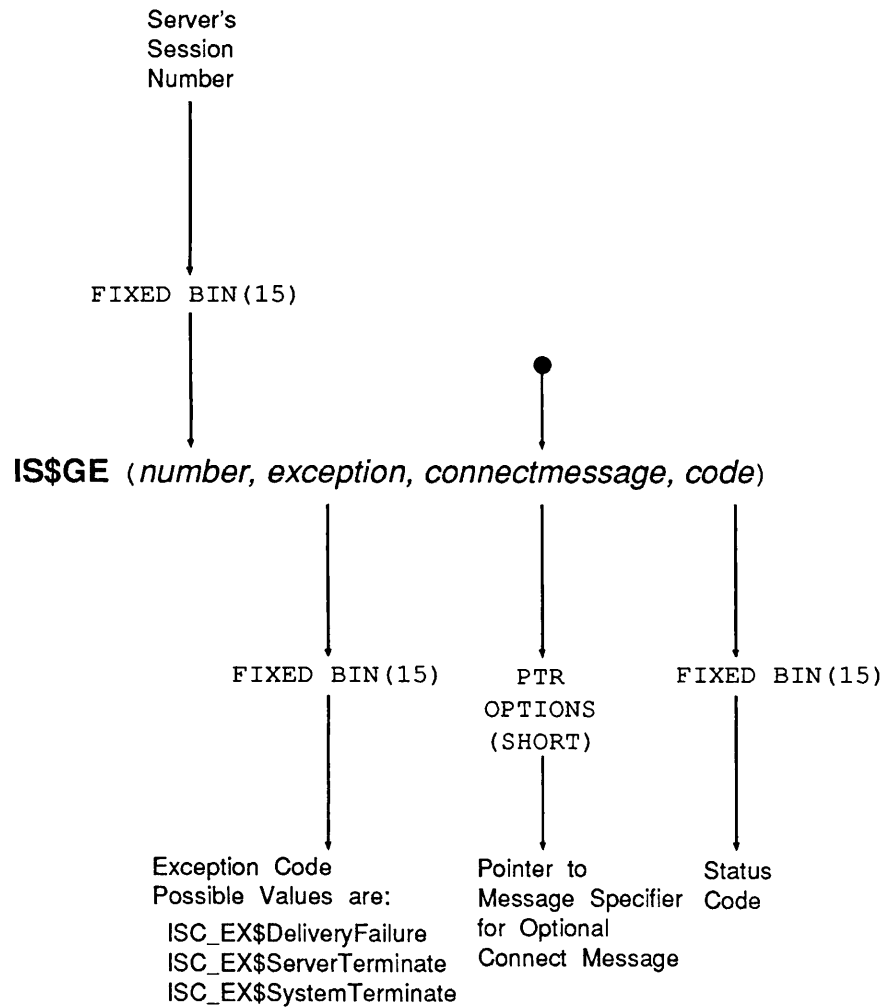
Accept an ISC Session



IS\$CE**Clear an ISC Session Exception**

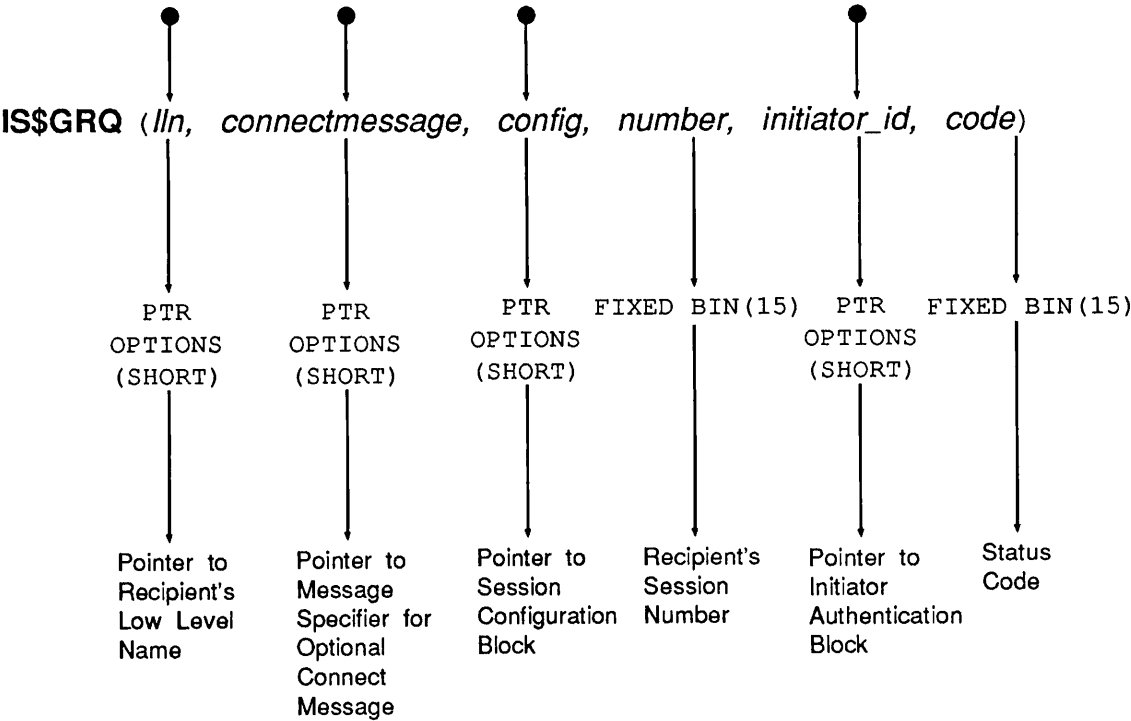
IS\$FB
Free an ISC Message Buffer

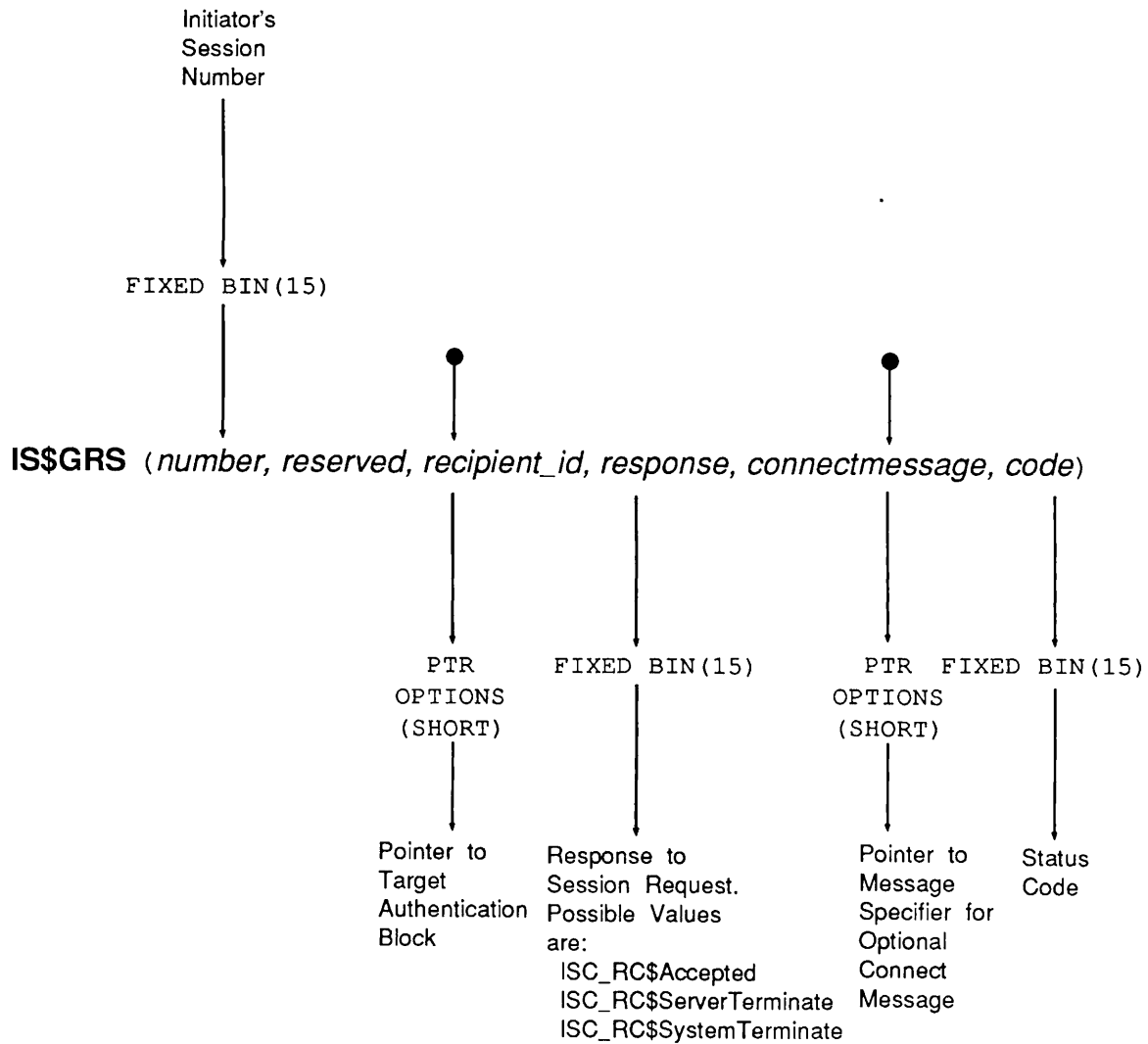


IS\$GE**Get an ISC Session Exception**

IS\$GRQ

IS\$GRQ
Get an ISC Session Request

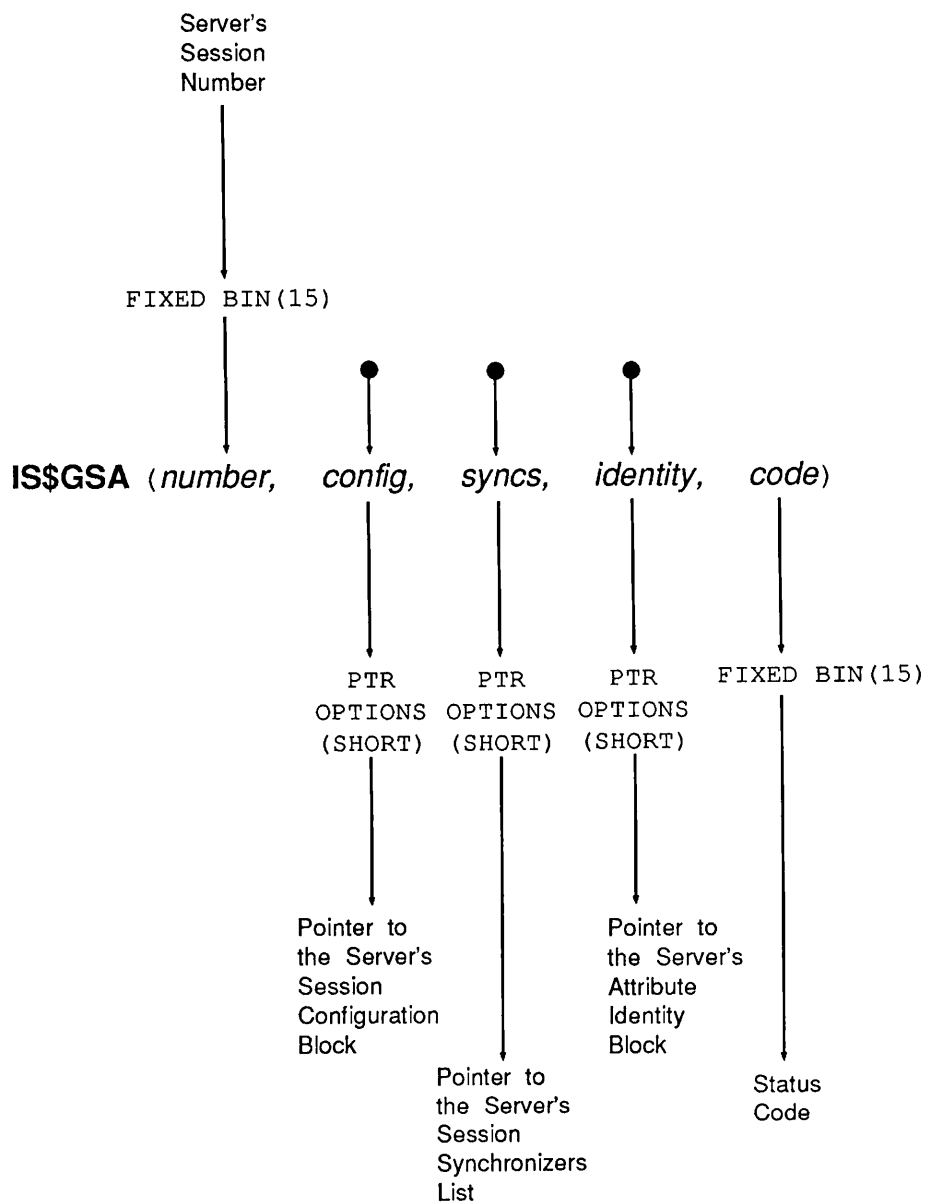


IS\$GRS**Get an ISC Session Request Response**

IS\$GSA

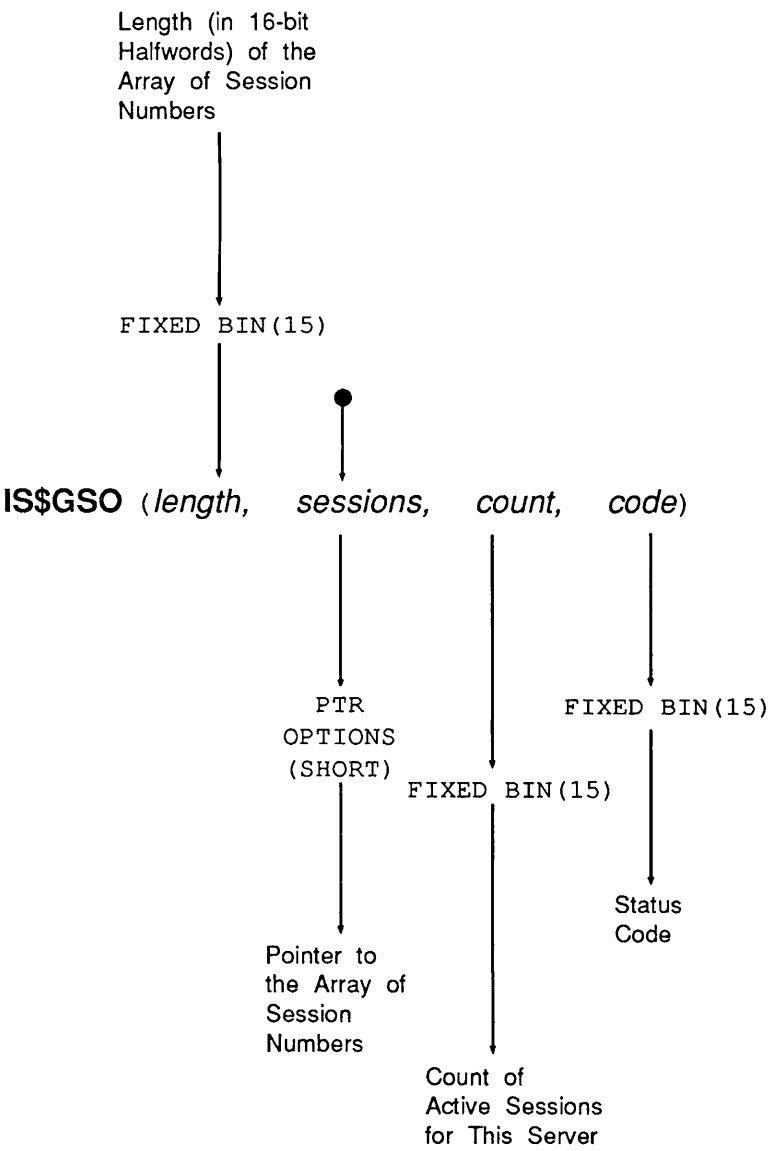
IS\$GSA

Get ISC Session Attributes



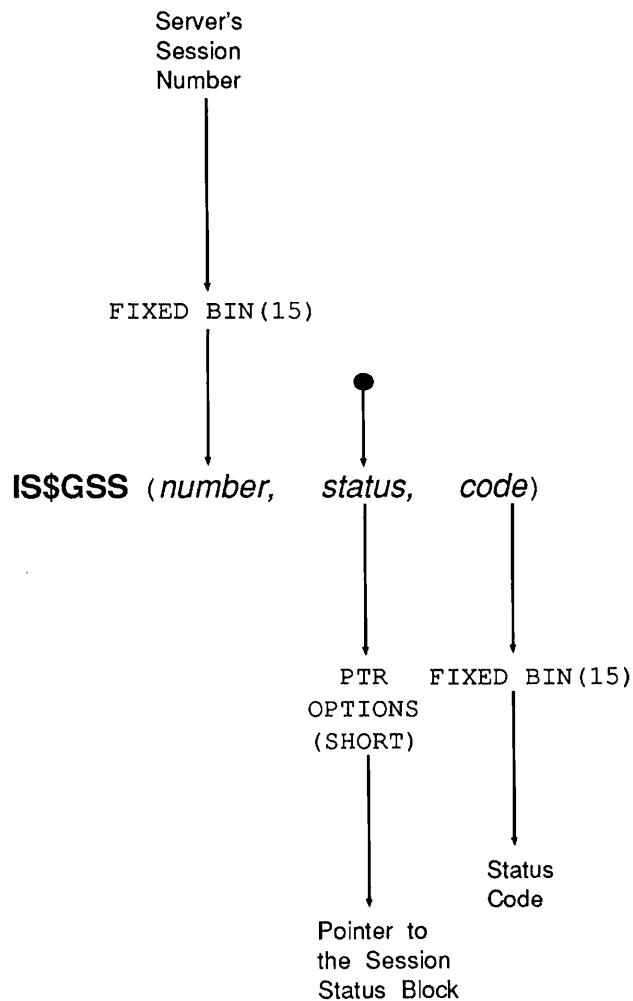
IS\$GSO

Get a List of ISC Sessions Owned by This Server

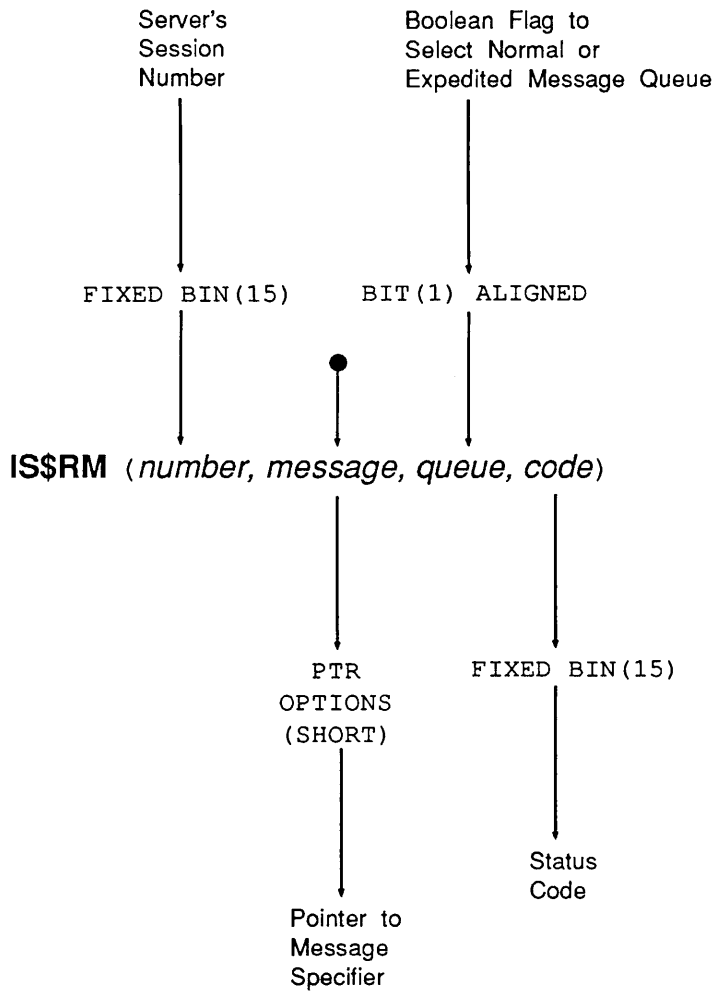


IS\$GSS

Get ISC Session Status Information



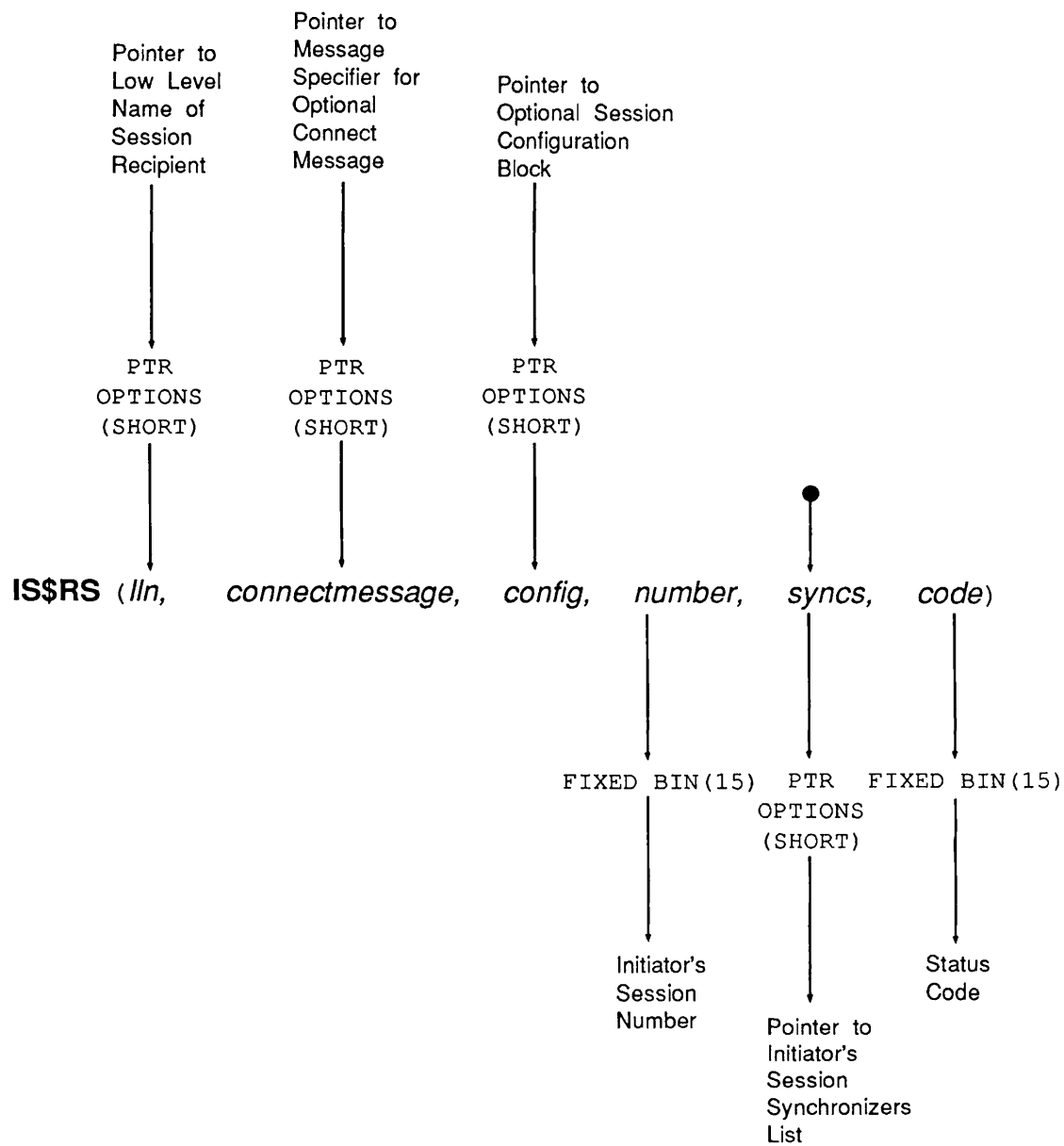
IS\$RM
Receive an ISC Message

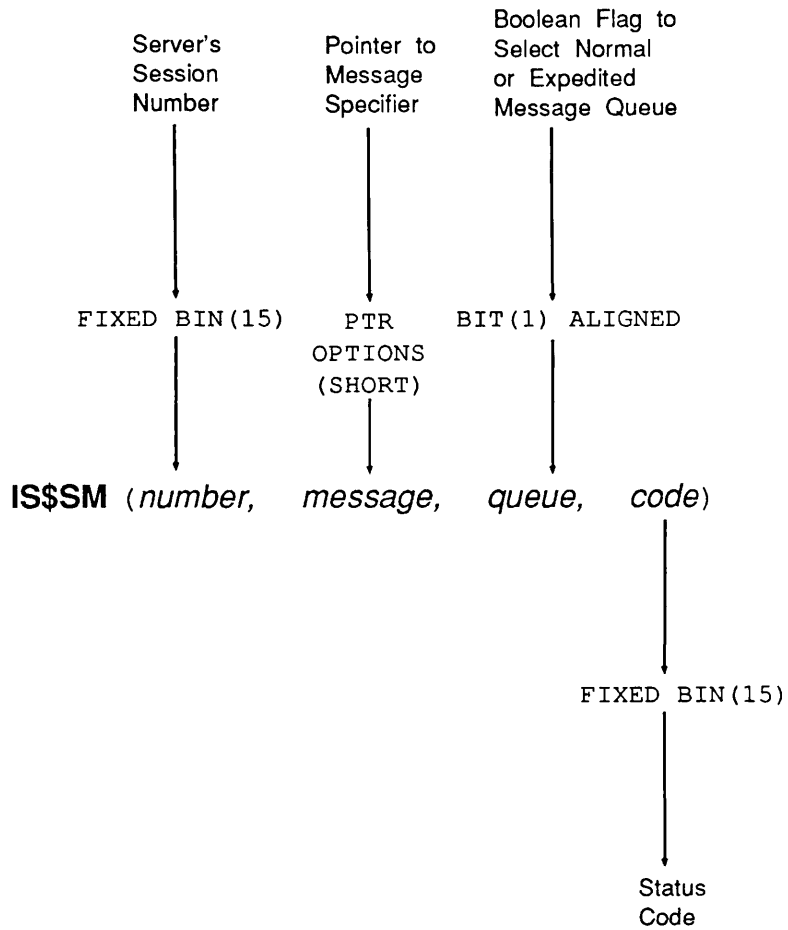


IS\$RS

IS\$RS

Request an ISC Session

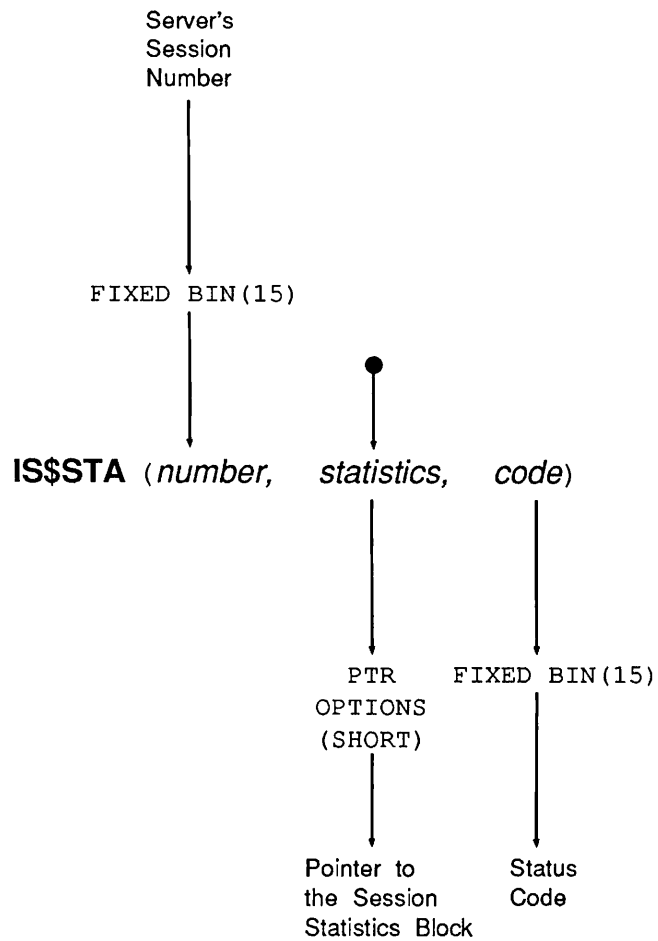


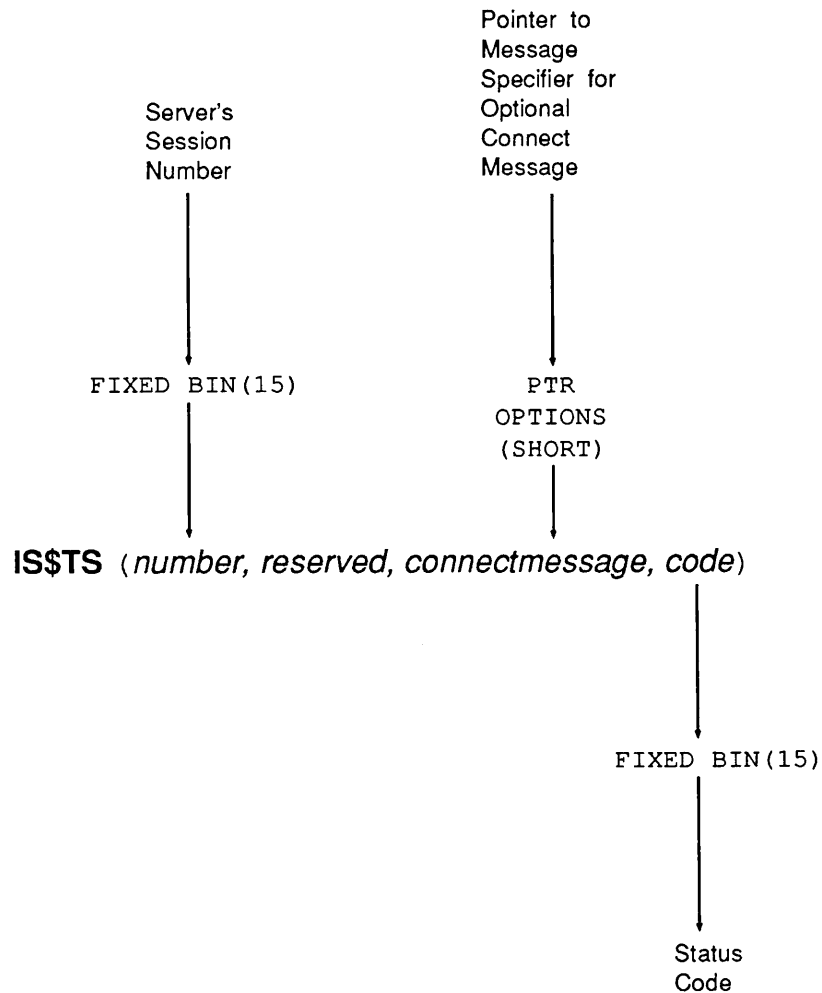
IS\$SM**Send an ISC Message**

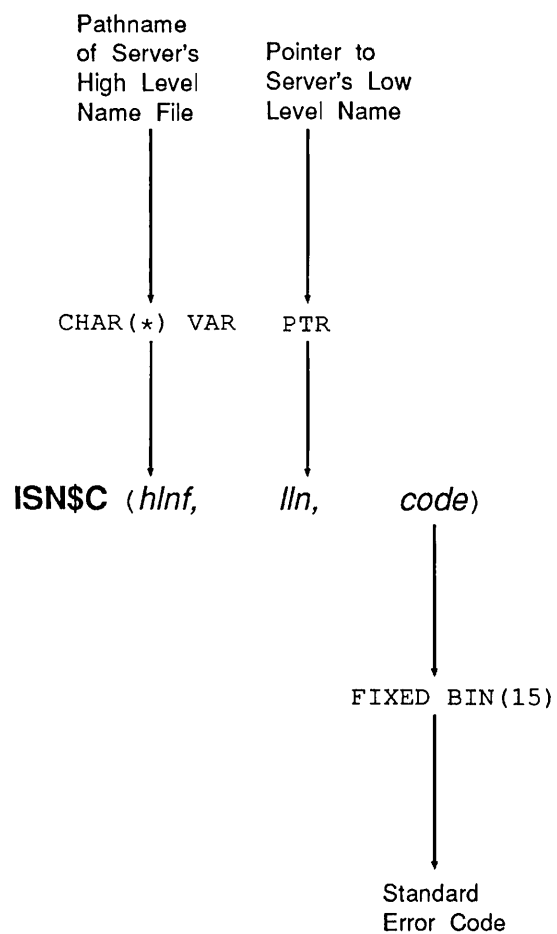
IS\$STA

IS\$STA

Get ISC Current Session Statistics

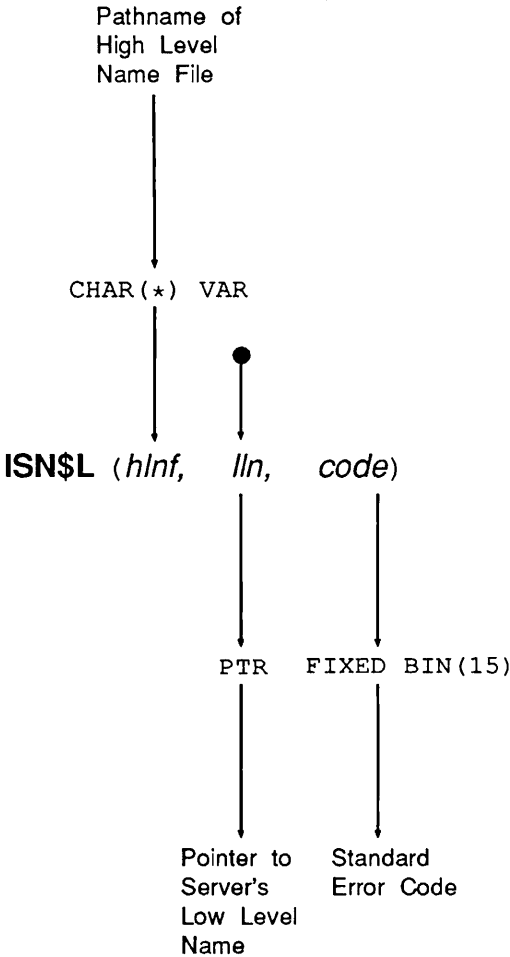


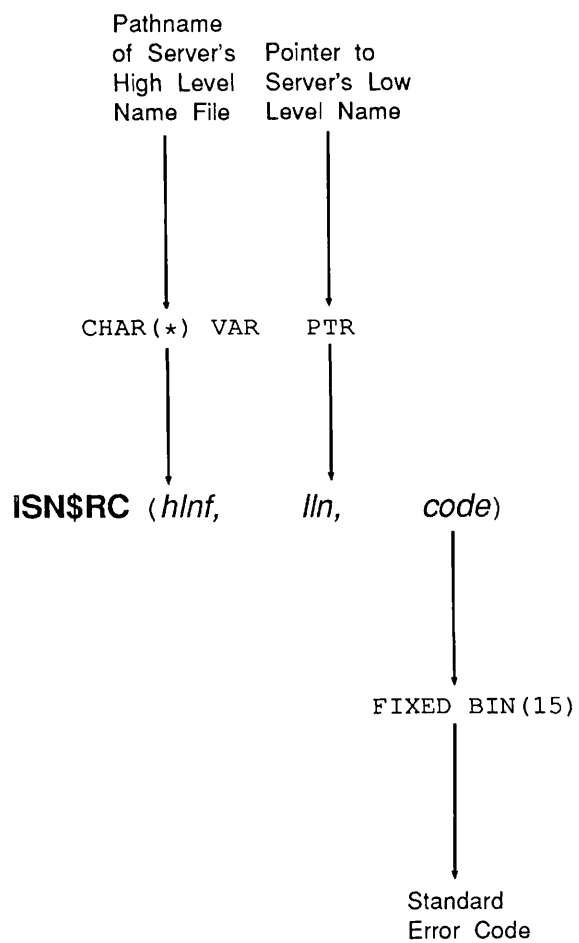
IS\$TS**Terminate an ISC Session**

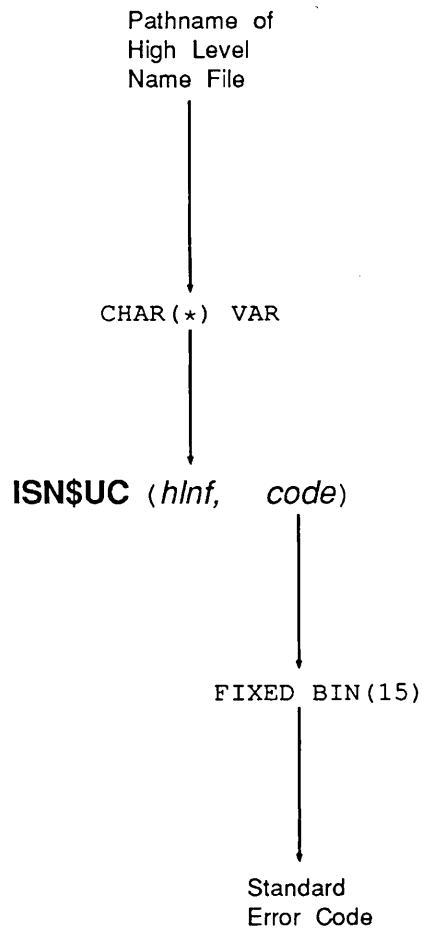
ISN\$C**ISN\$C****Catalog Server's Low Level Name**

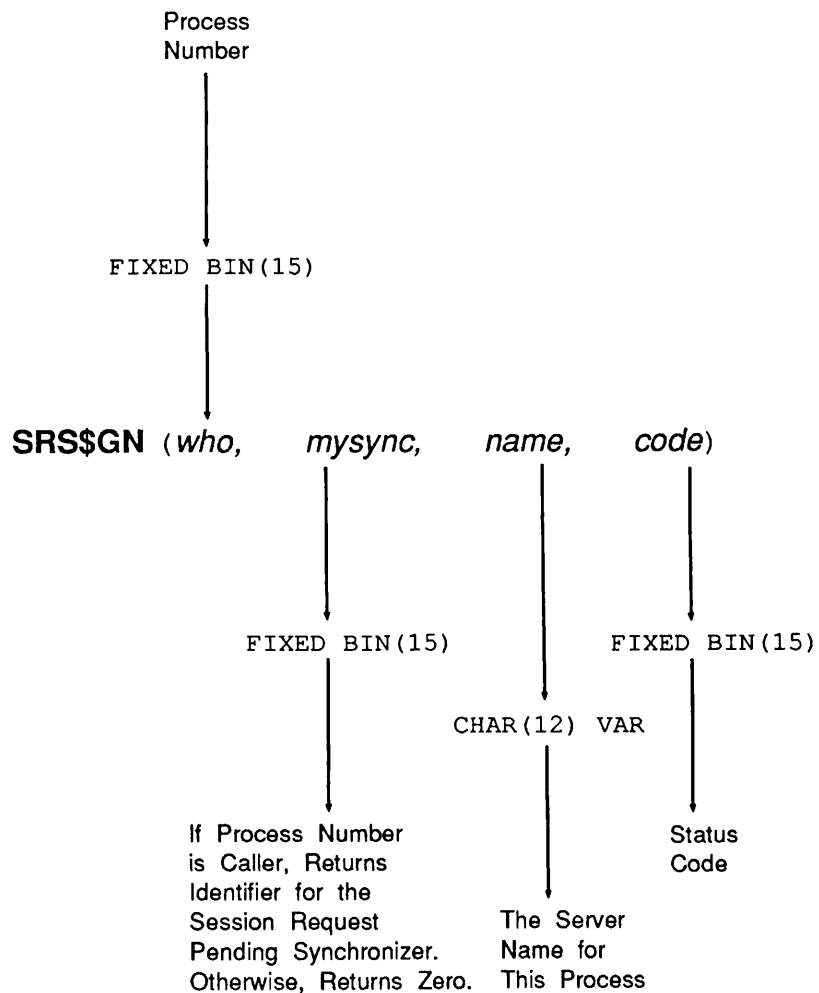
ISN\$L

Look Up Server's Low Level Name in a High Level Name File



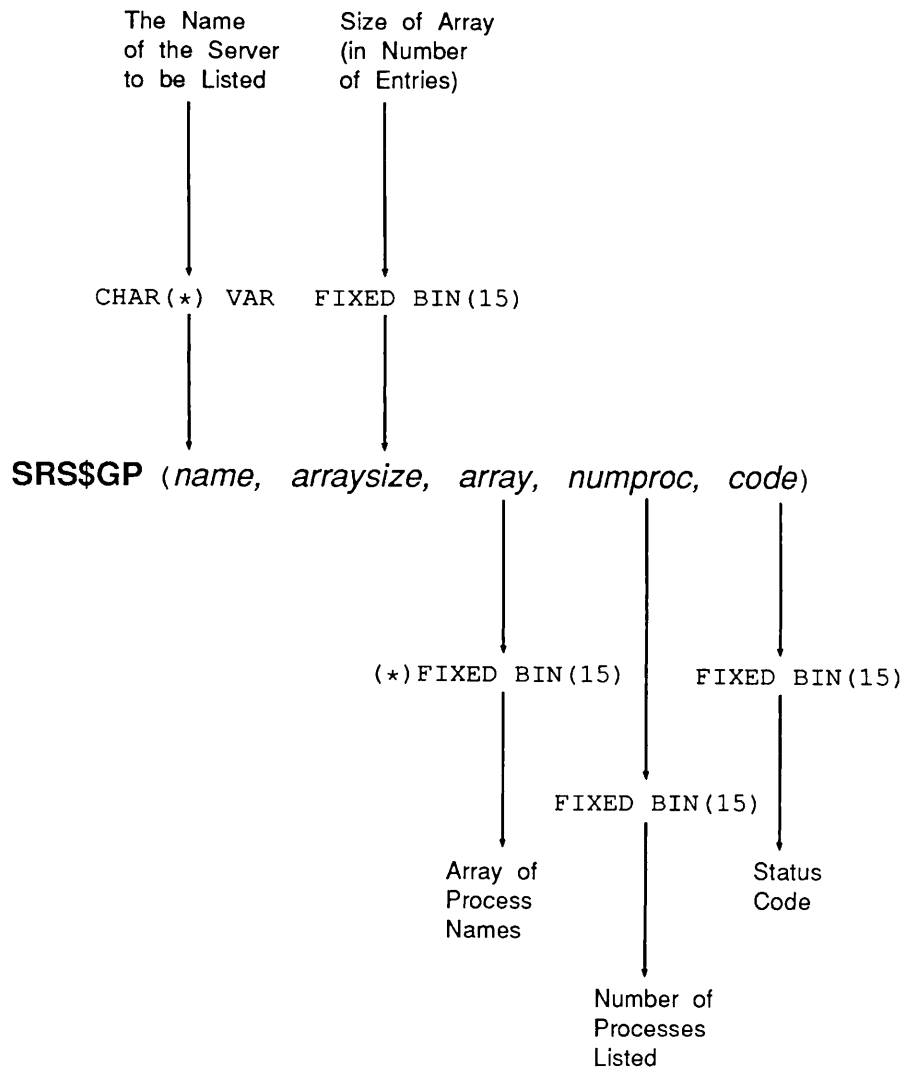
ISN\$RC**ISN\$RC****Recatalog Server's Low Level Name**

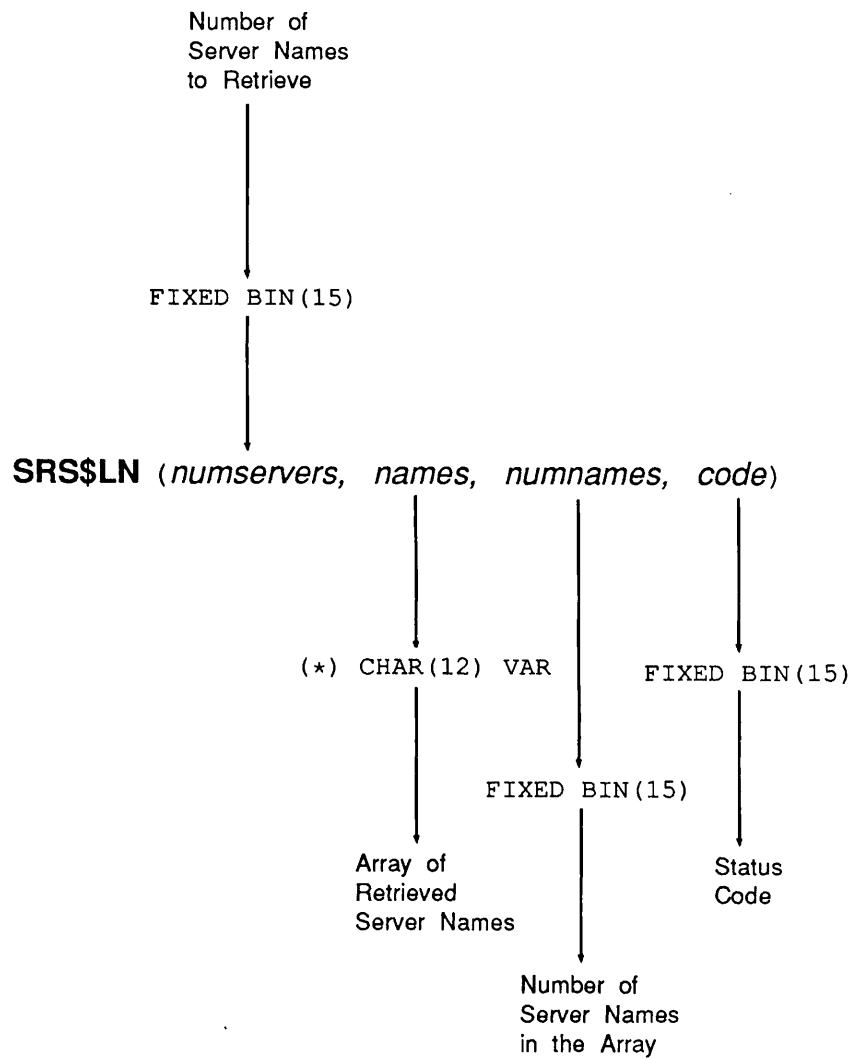
ISN\$UC**Uncatalog (Delete) Server's Low Level Name**

SRS\$GN**SRS\$GN****Get the Server Name of a Process**

SRS\$GP

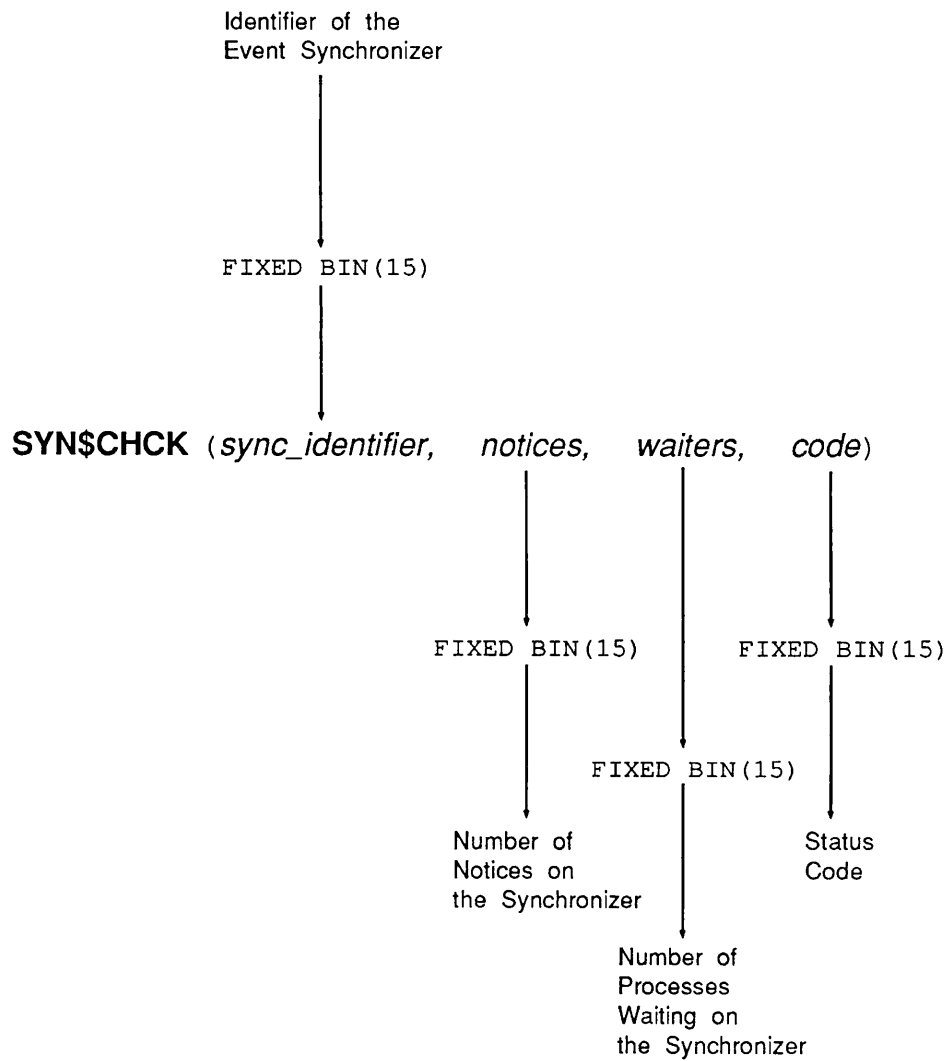
Get the Process Numbers of all Processes That Share a Specified Server Name



SRS\$LN**SRS\$LN****List All Active Server Names**

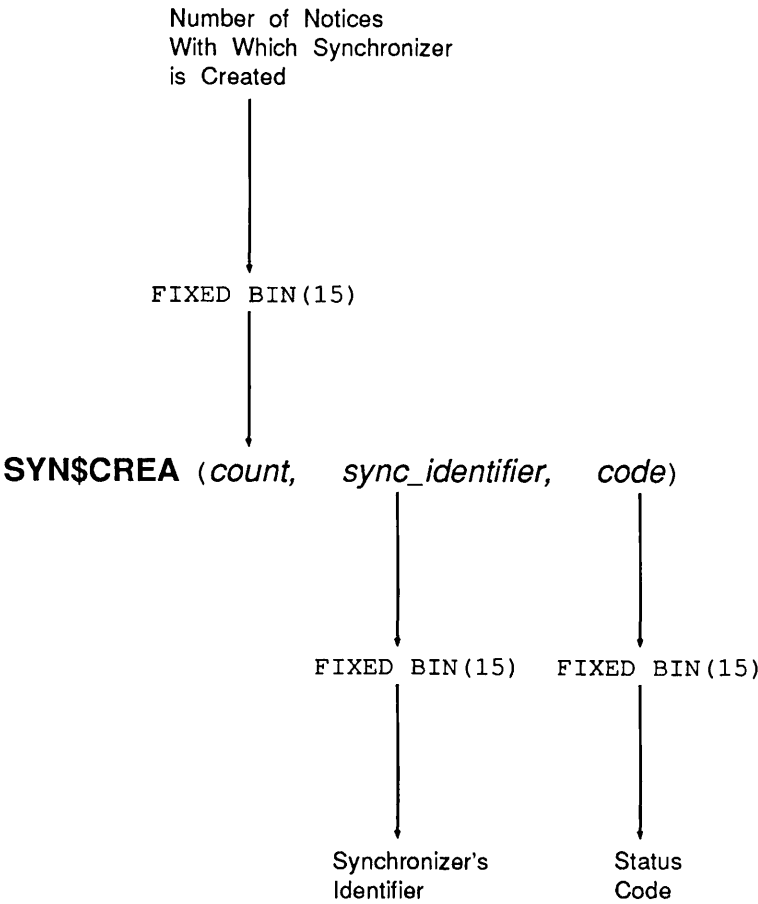
SYN\$CHCK

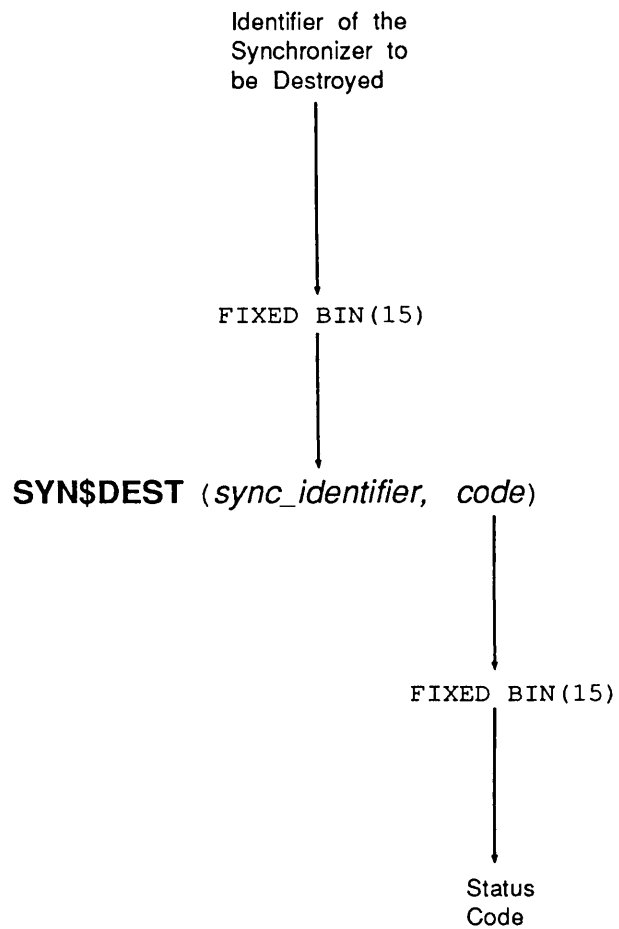
Return Information About an Event Synchronizer

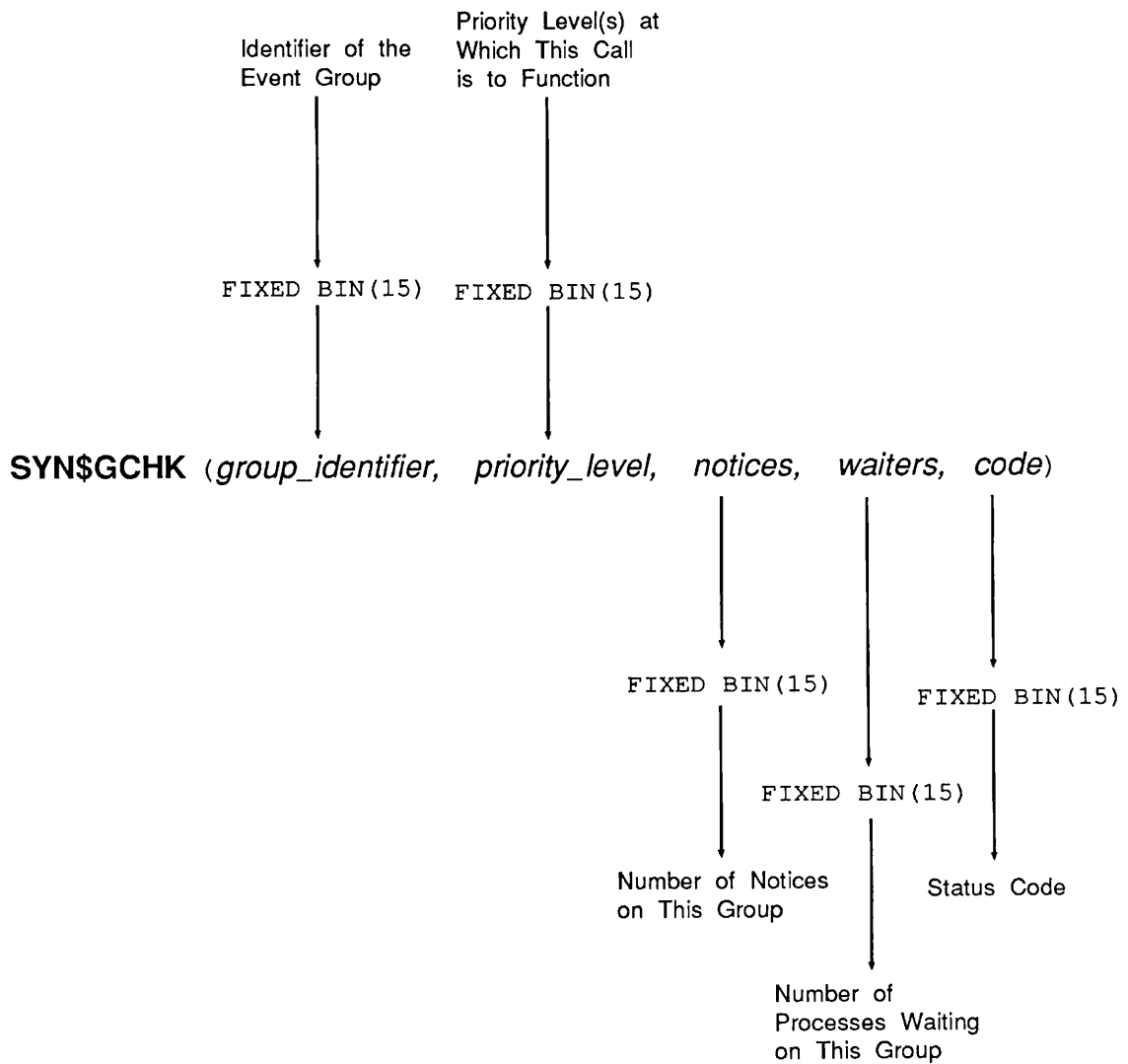


SYN\$CREA

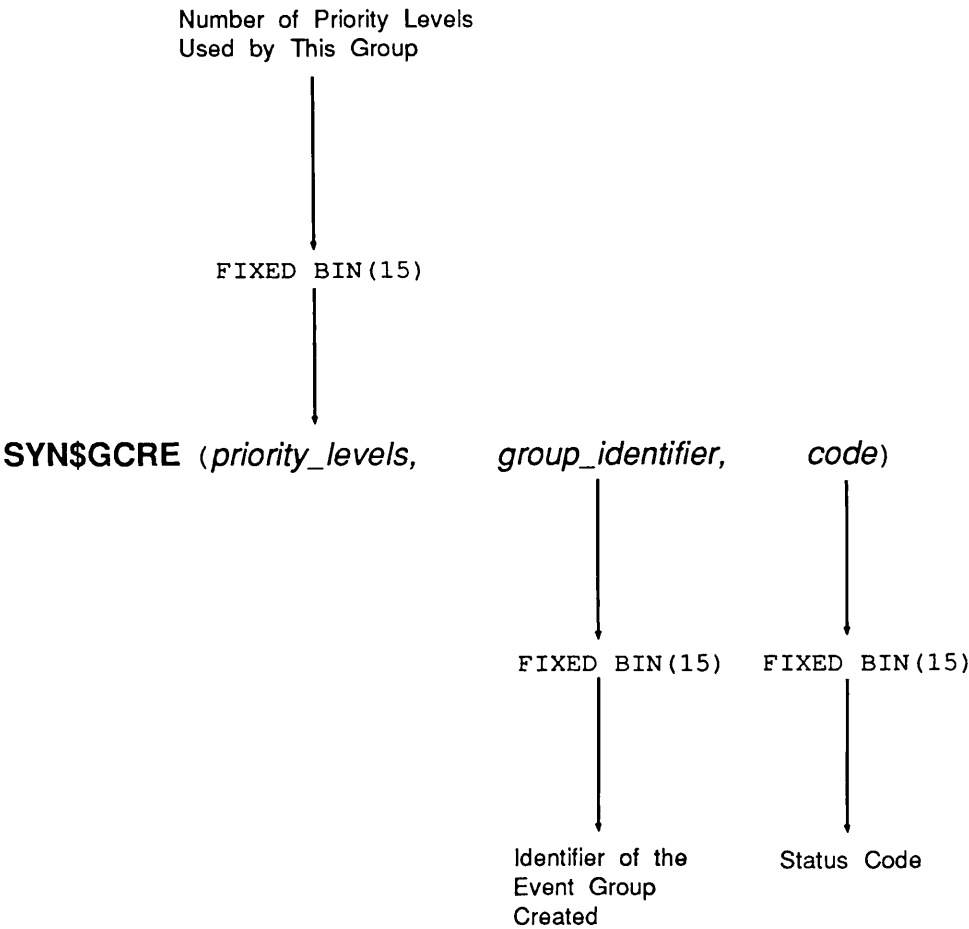
SYN\$CREA
Create an Event Synchronizer



SYN\$DEST**Destroy an Event Synchronizer**

SYN\$GCHK**Return Count of Notices or Waiters on an Event Group**

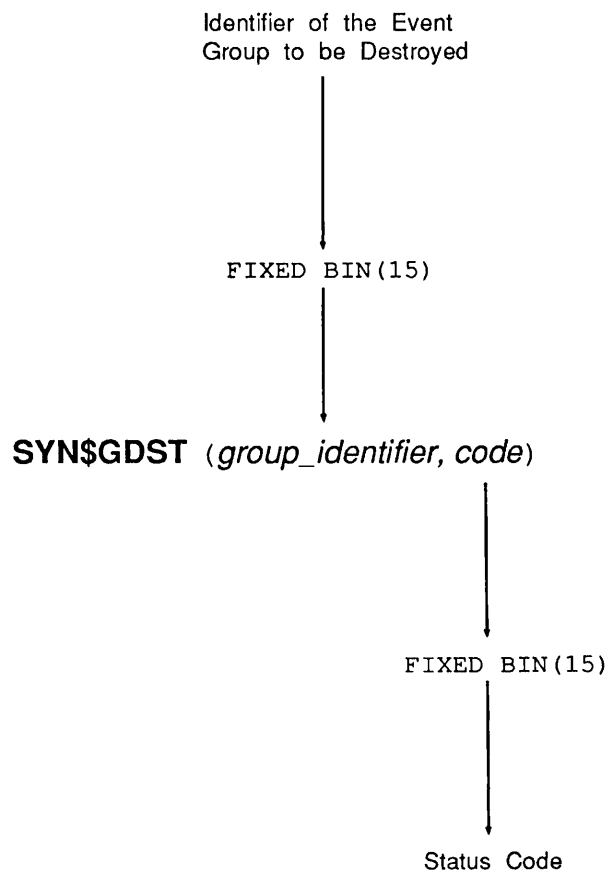
SYN\$GCRE
Create an Event Group



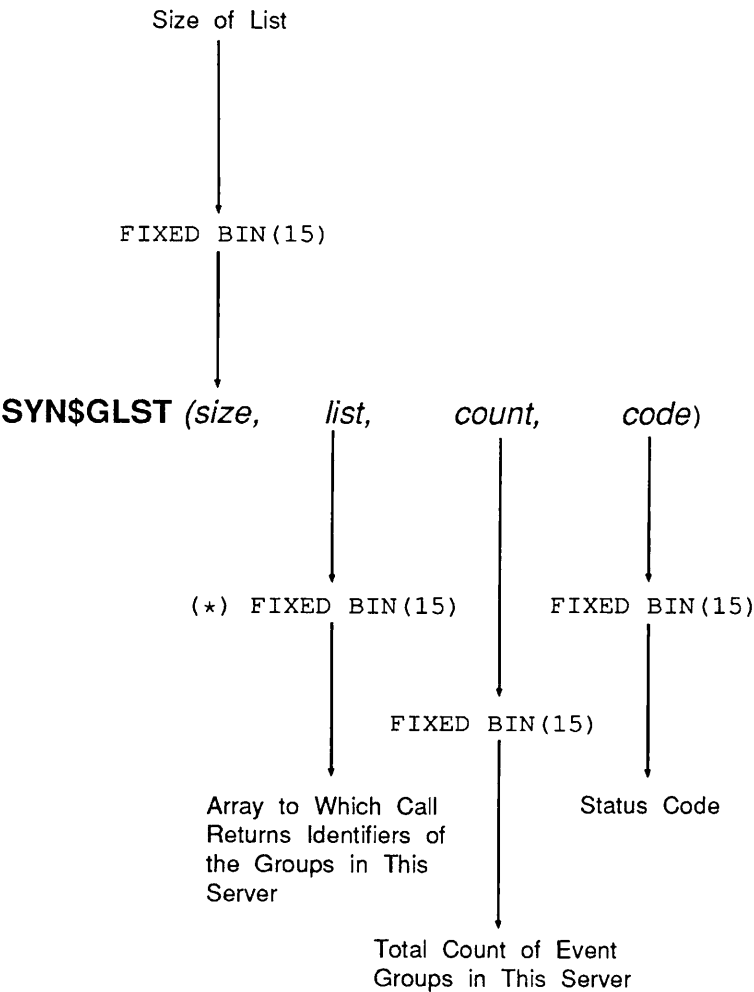
SYN\$GDST

SYN\$GDST

Destroy an Event Group



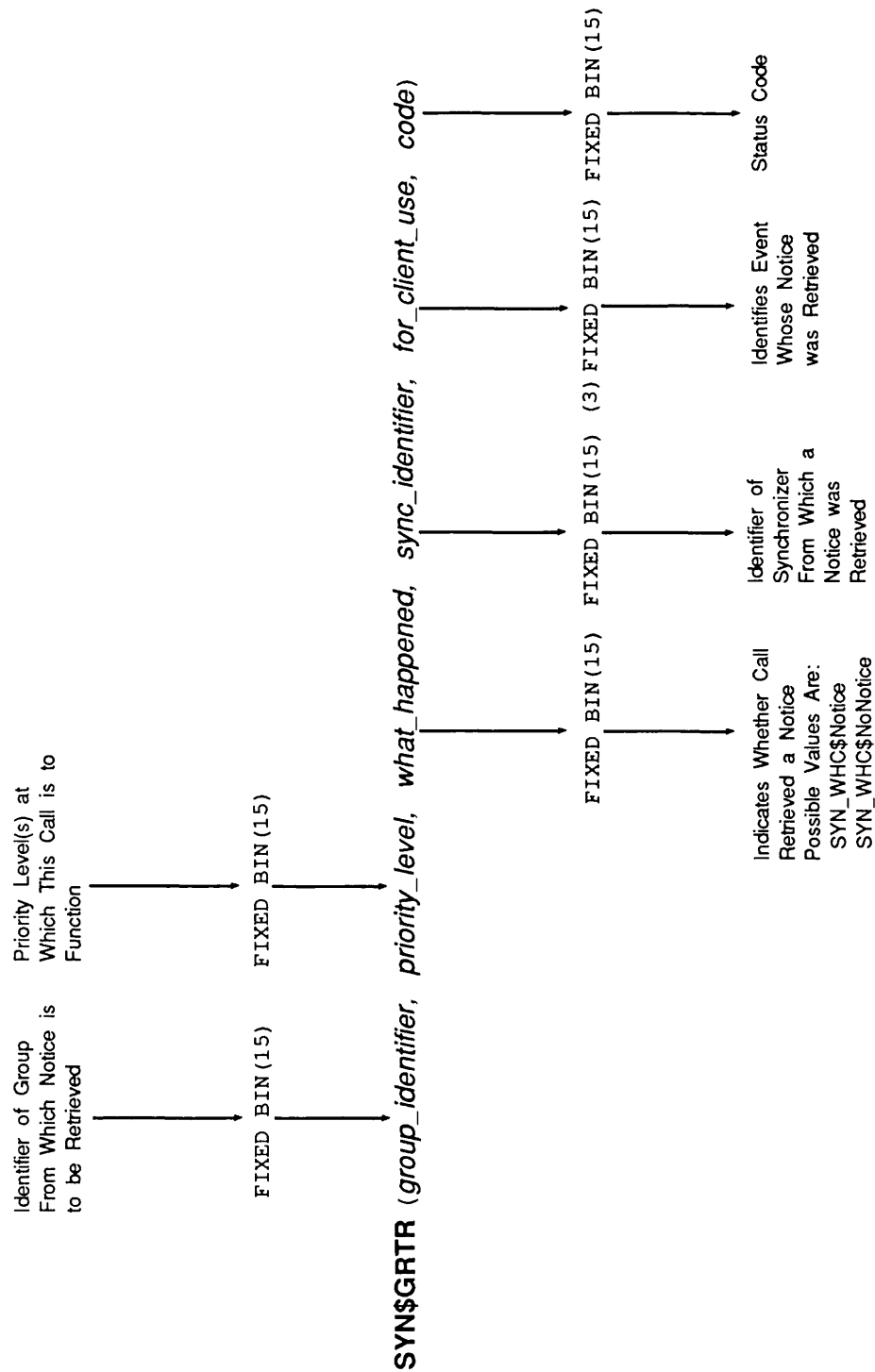
SYN\$GLST
List Groups in Server



SYN\$GRTR

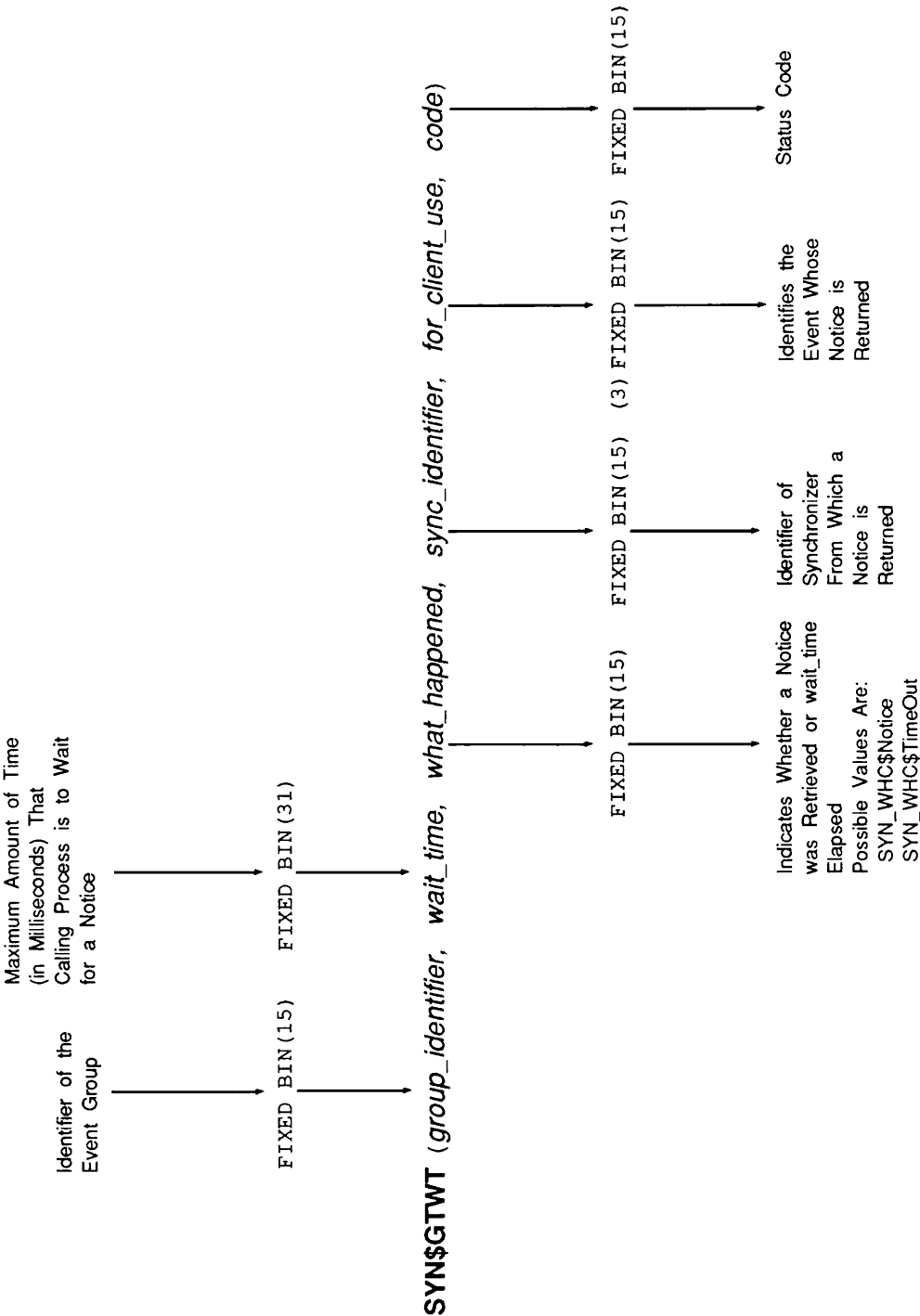
SYN\$GRTR

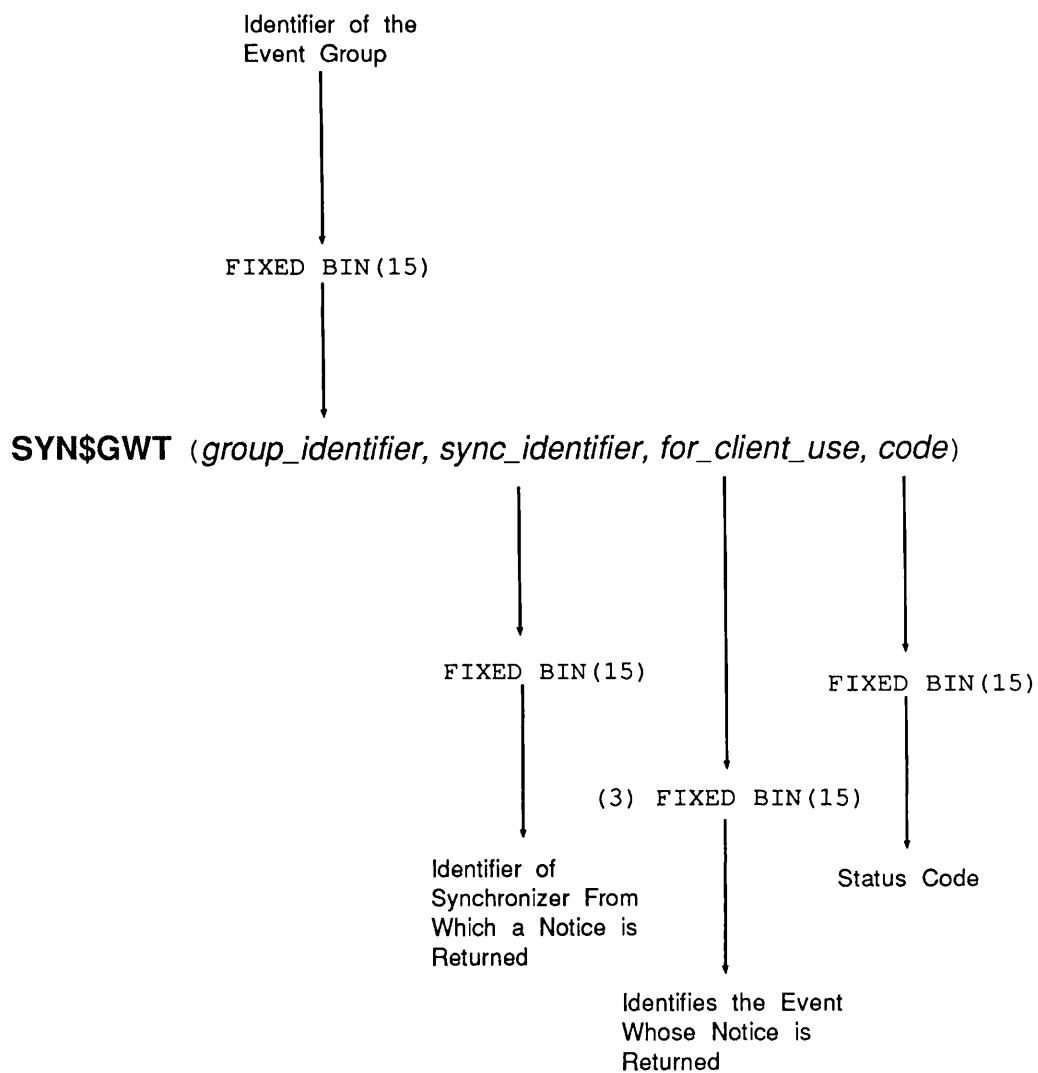
Retrieve a Notice From an Event Group

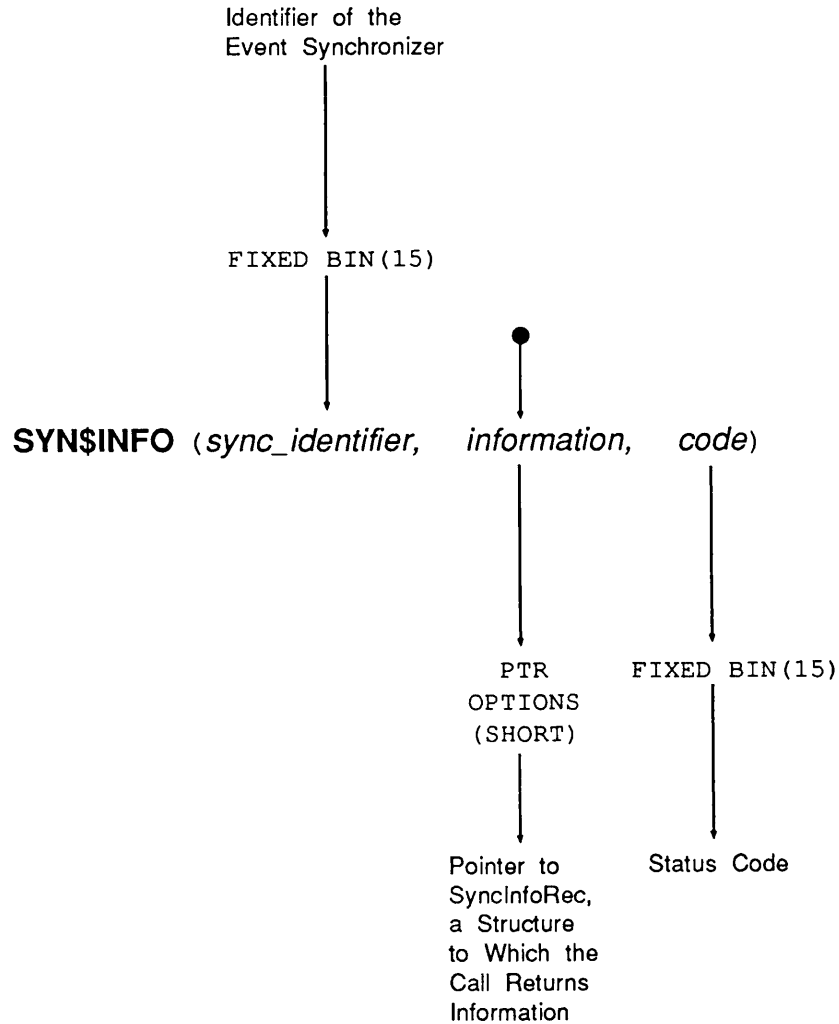


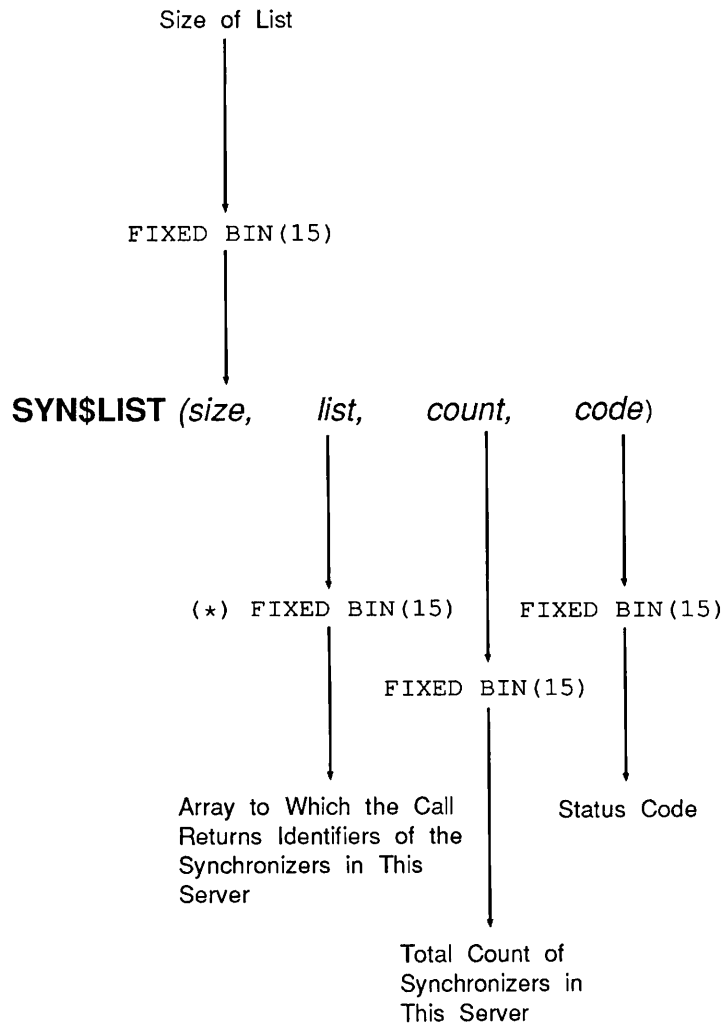
SYN\$GTWT

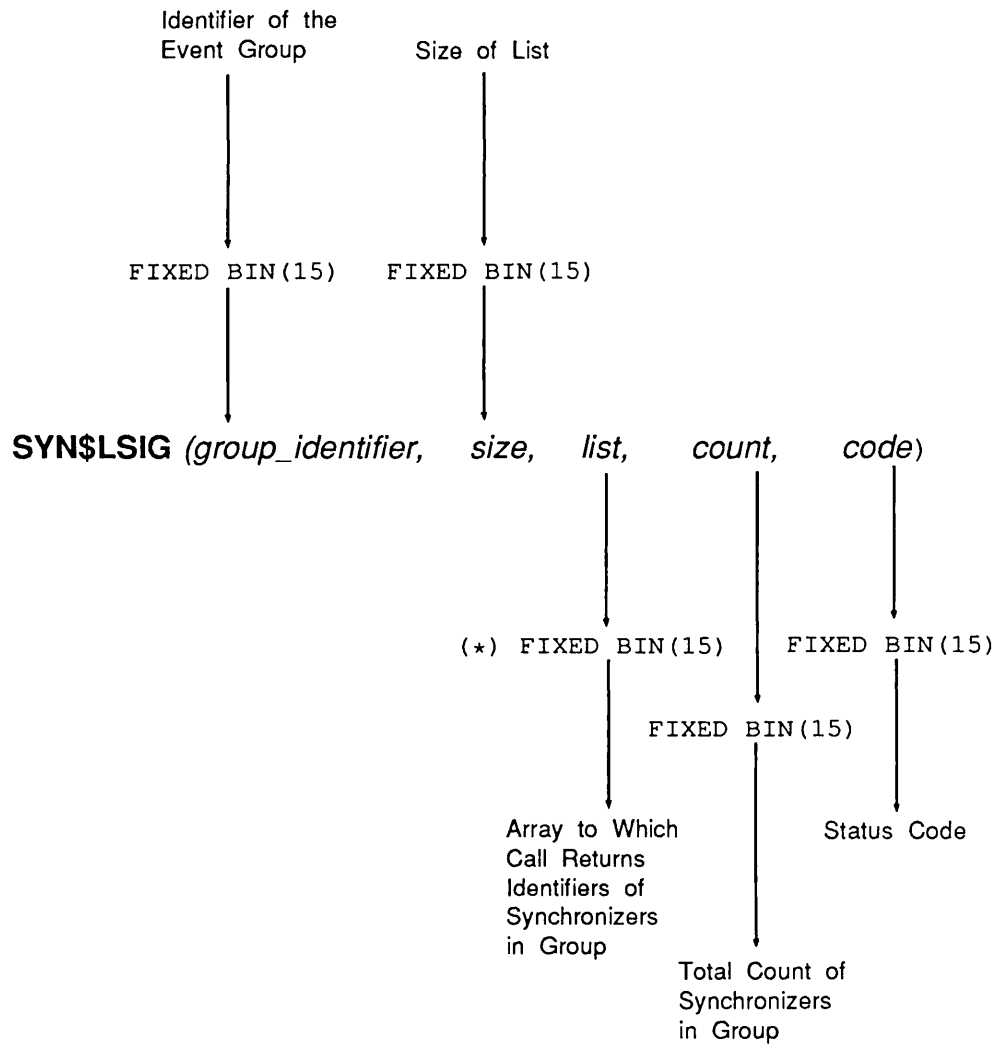
Perform a Timed Wait on an Event Group



SYN\$GWT**SYN\$GWT****Wait on an Event Group**

SYN\$INFO**Return Information About an Event Synchronizer**

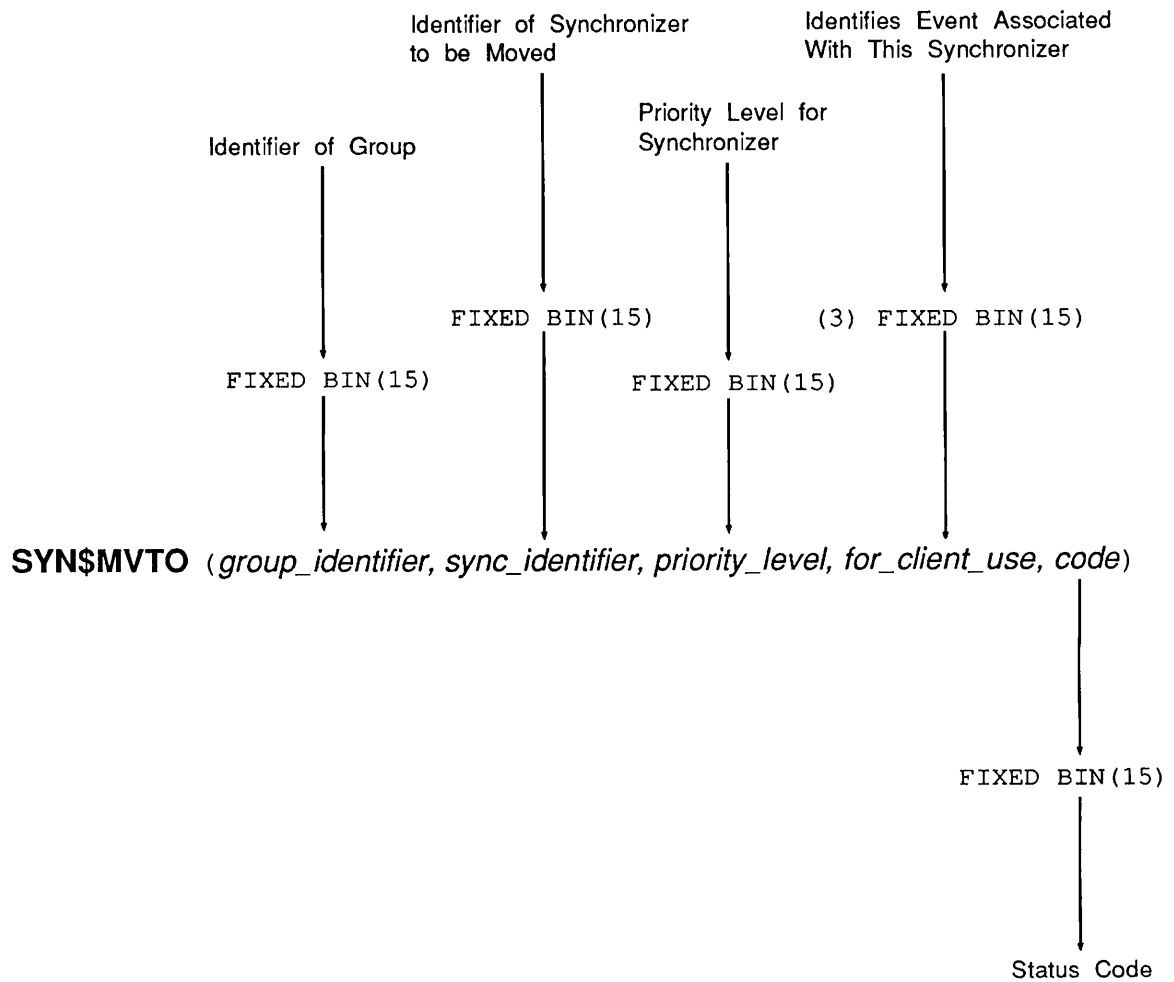
SYN\$LIST**SYN\$LIST****List Synchronizers Owned by This Server**

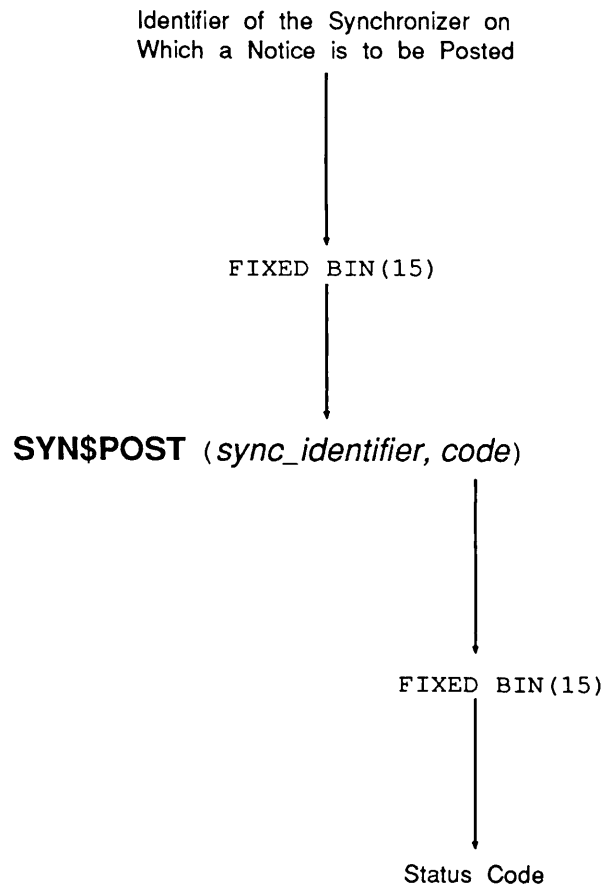
SYN\$LSIG**List Synchronizers Belonging to This Event Group**

SYN\$MVTO

SYN\$MVTO

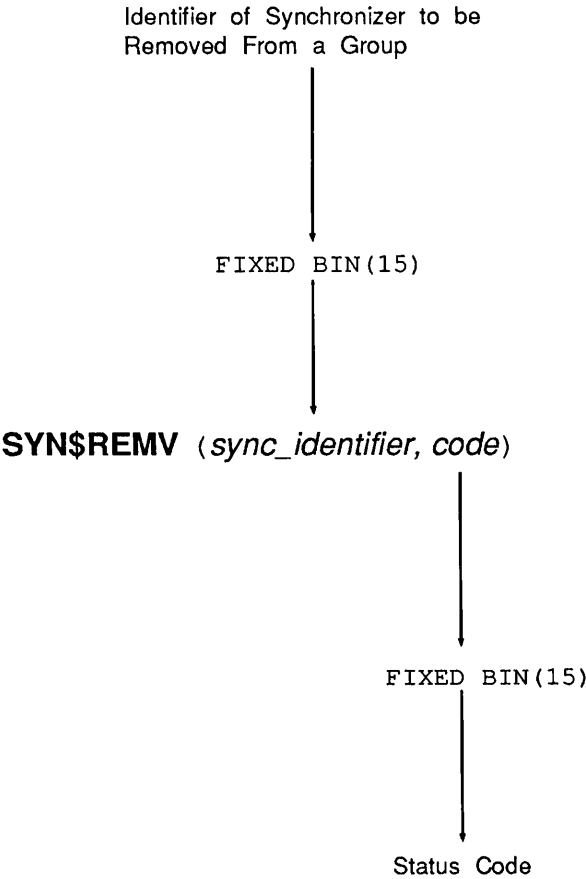
Move an Event Synchronizer Into an Event Group

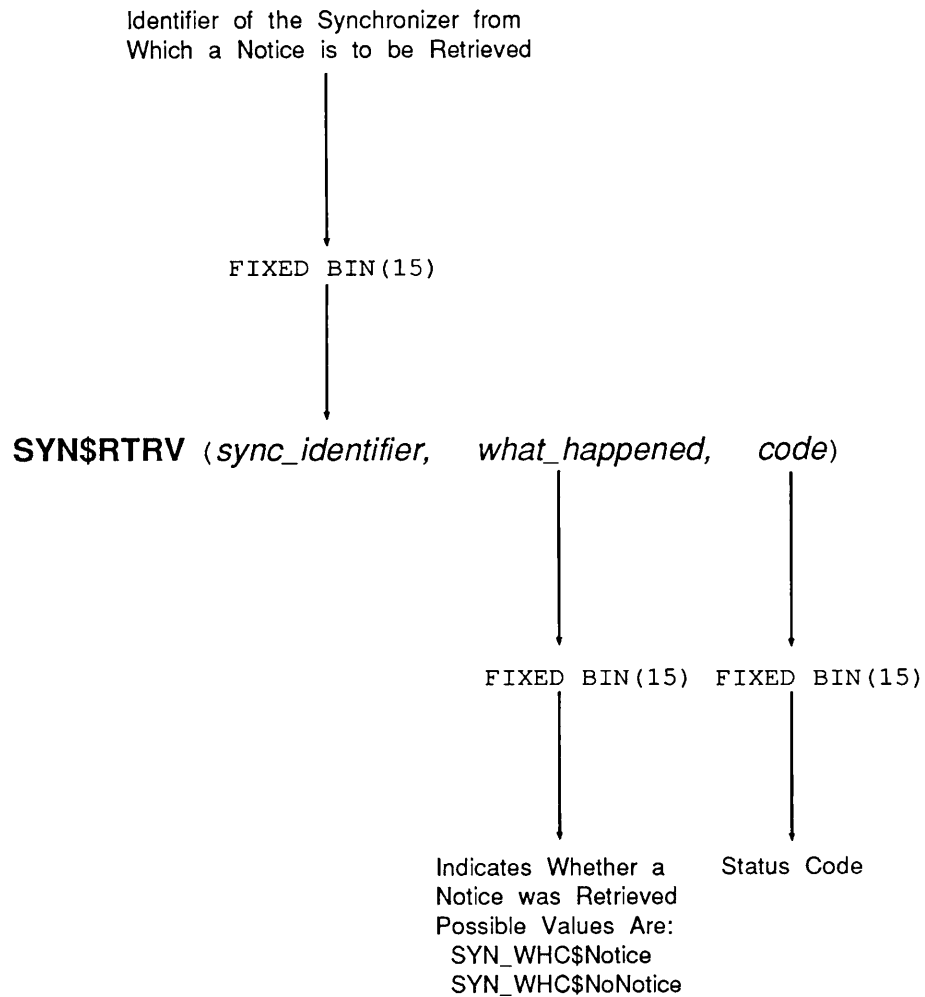


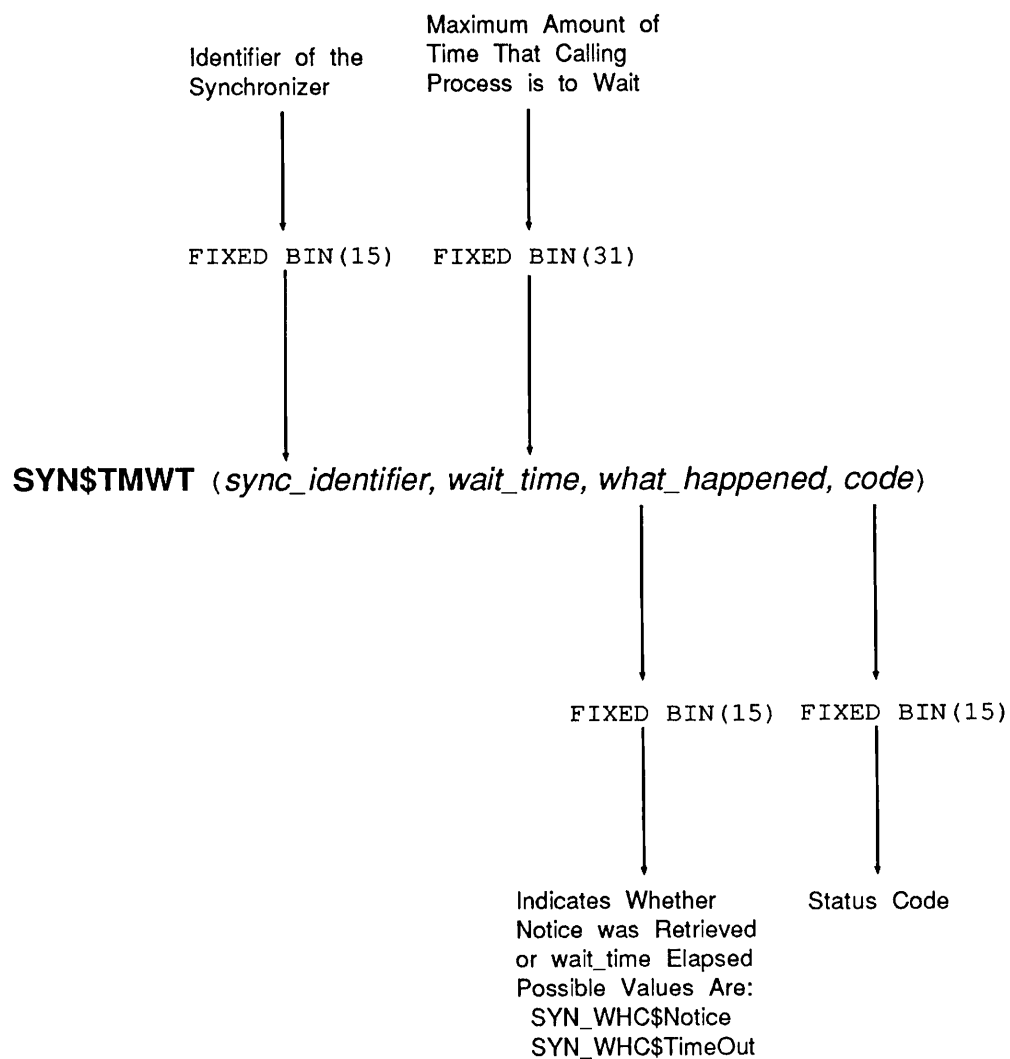
SYN\$POST**Post a Notice on an Event Synchronizer**

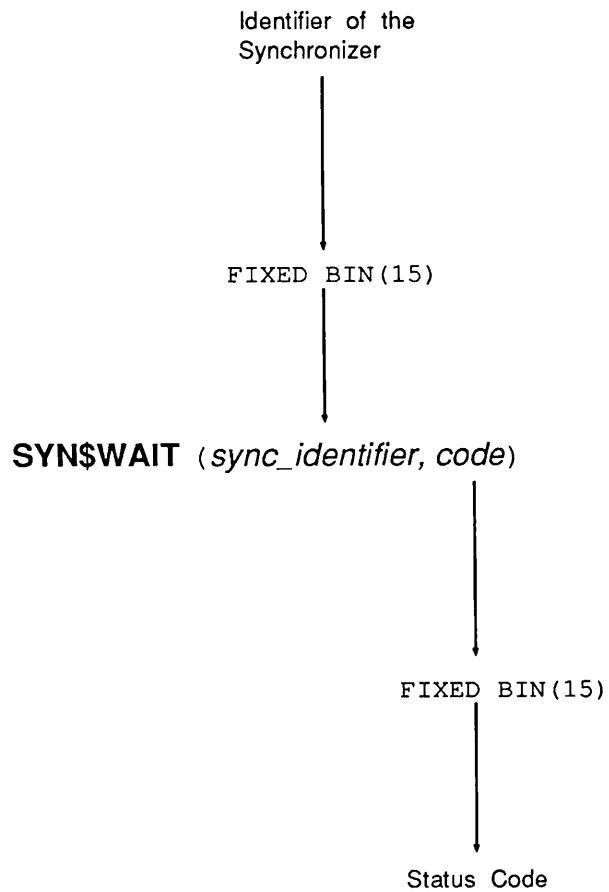
SYN\$REMV

SYN\$REMV
Remove an Event Synchronizer From an Event Group



SYN\$RTRV**Retrieve a Notice From an Event Synchronizer**

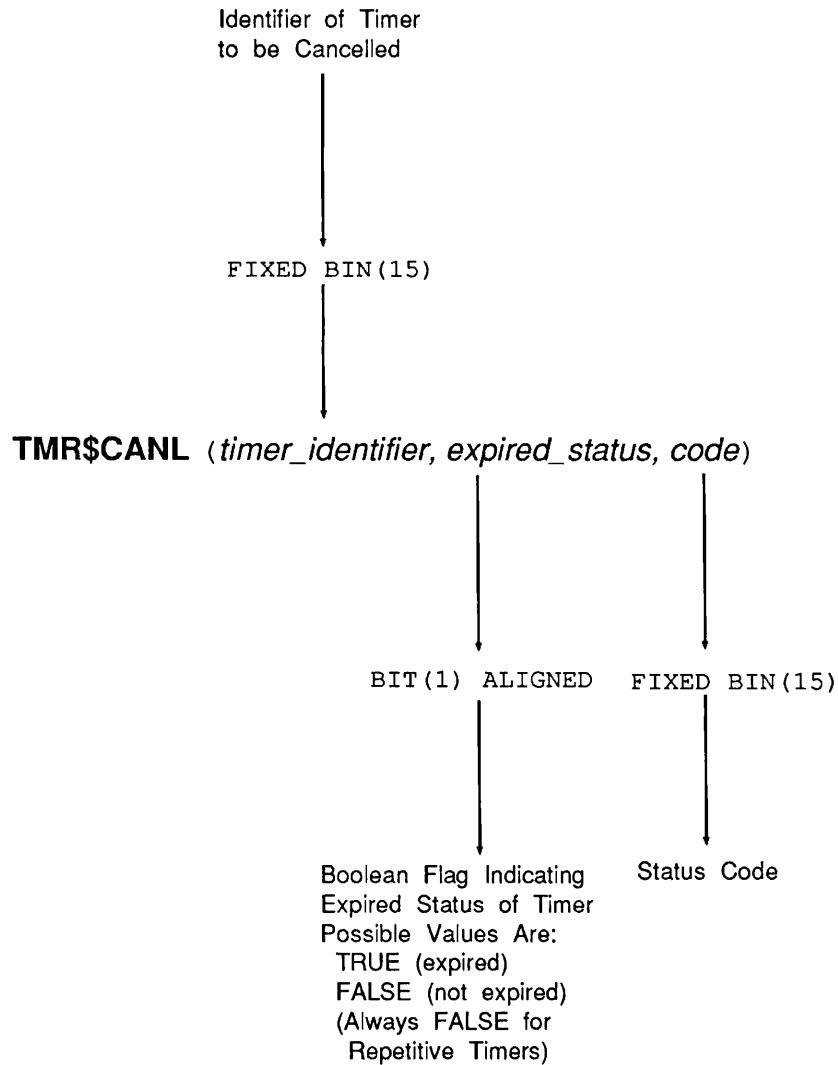
SYN\$TMWT**SYN\$TMWT****Perform a Timed Wait on a Synchronizer**

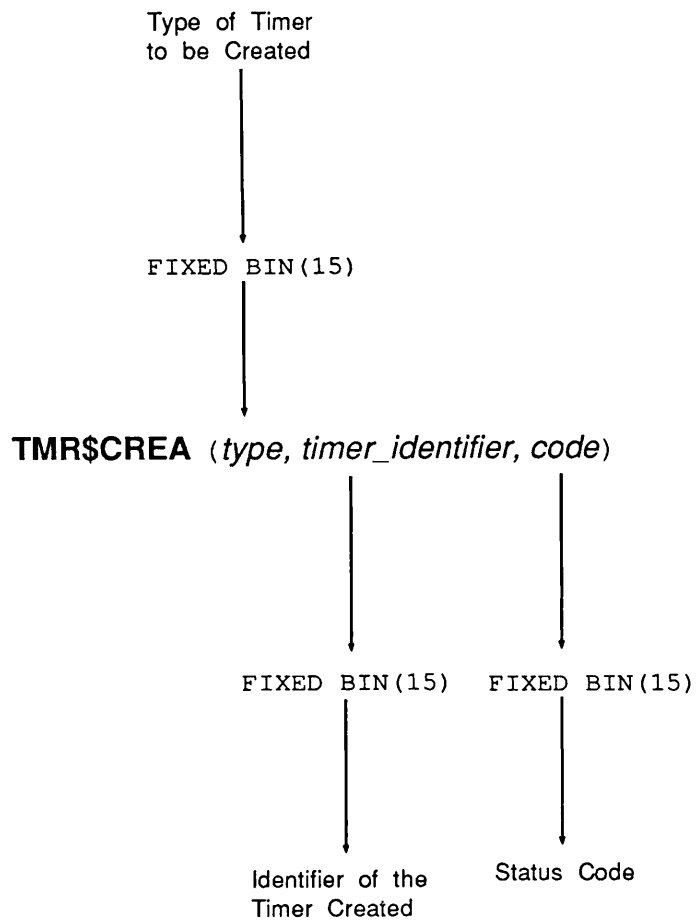
SYN\$WAIT**Wait on an Event Synchronizer**

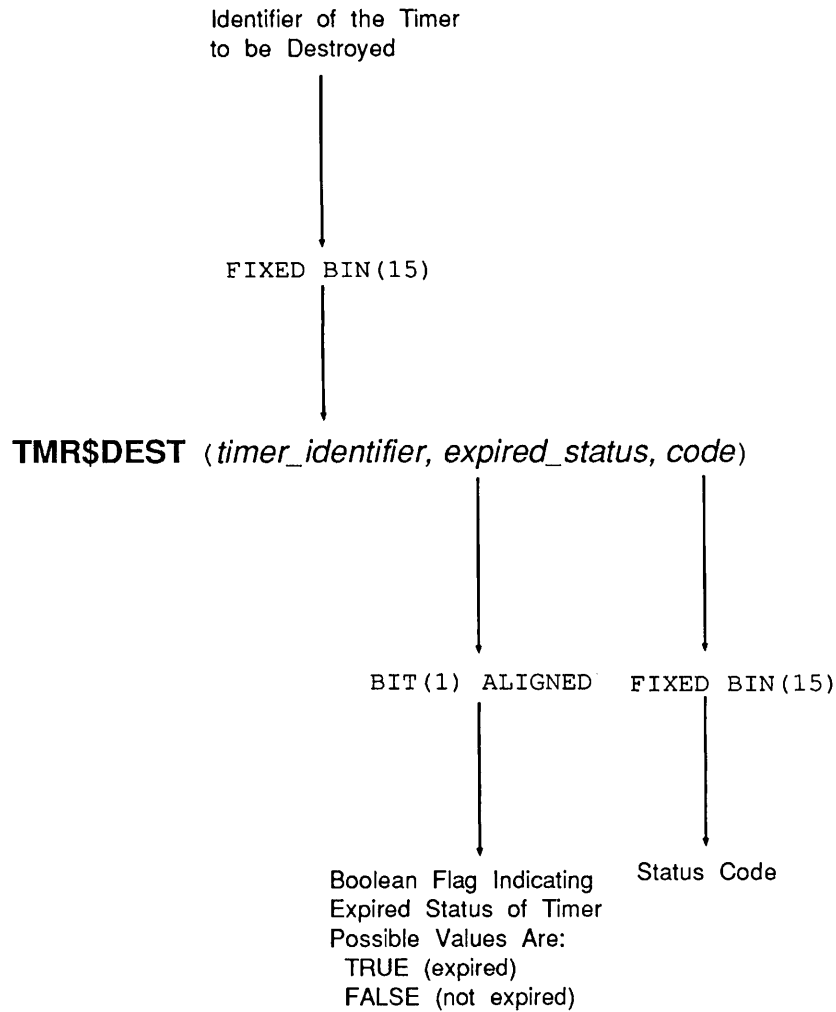
TMR\$CANL

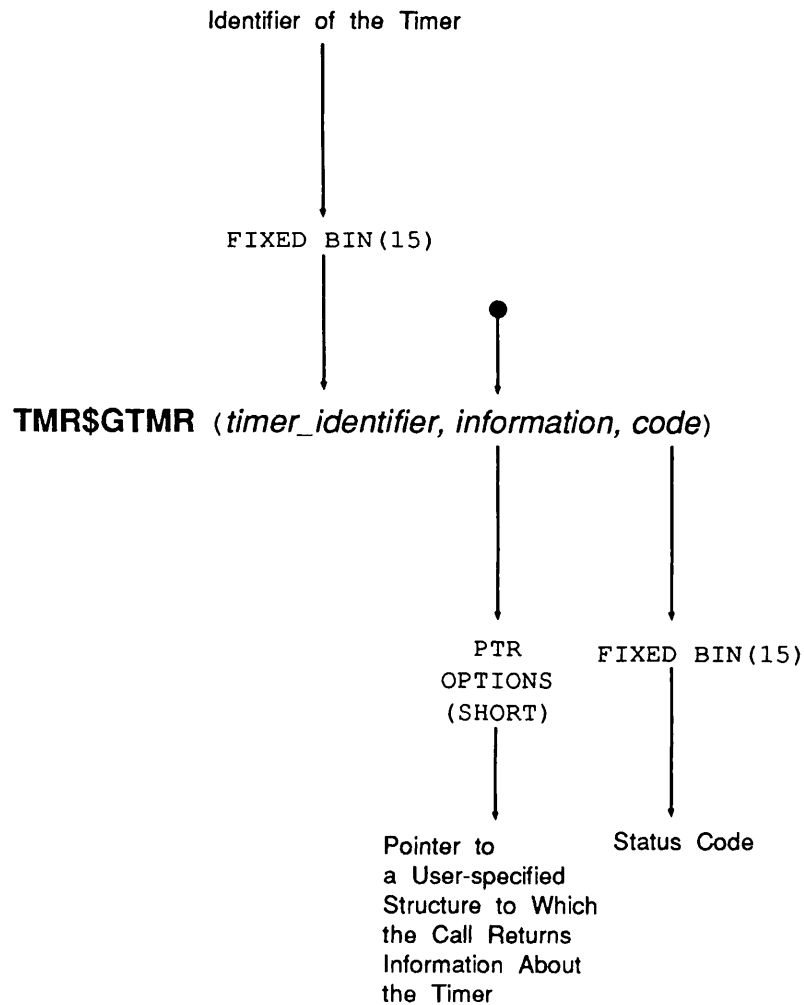
TMR\$CANL

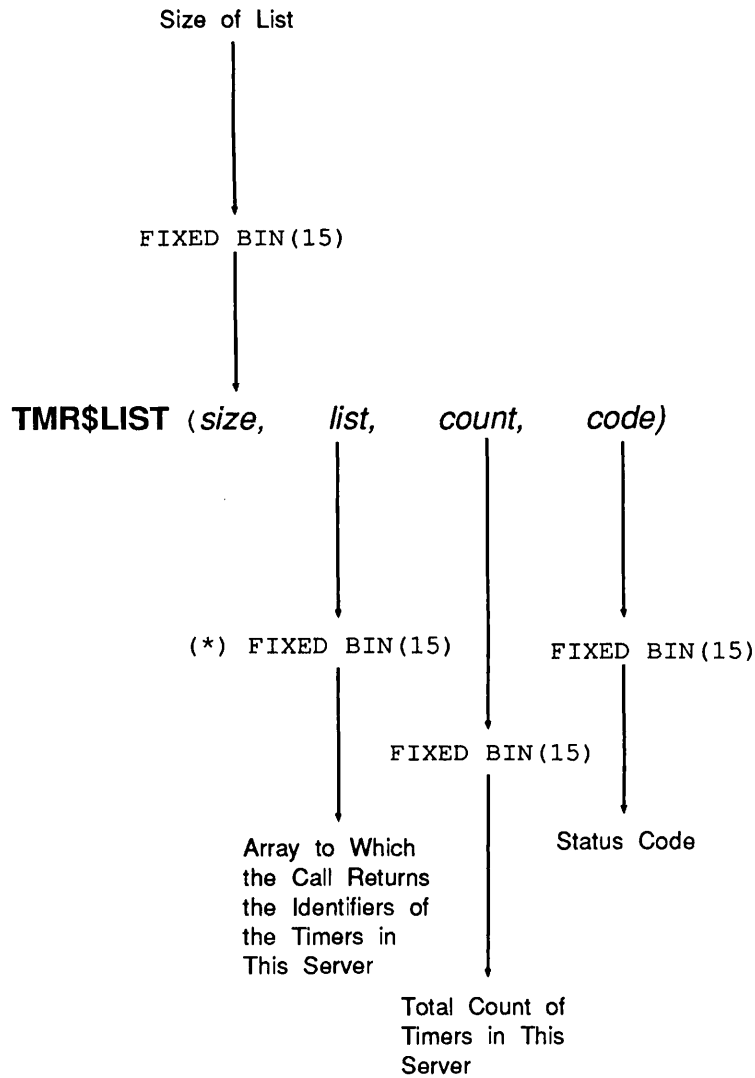
Cancel a Timer

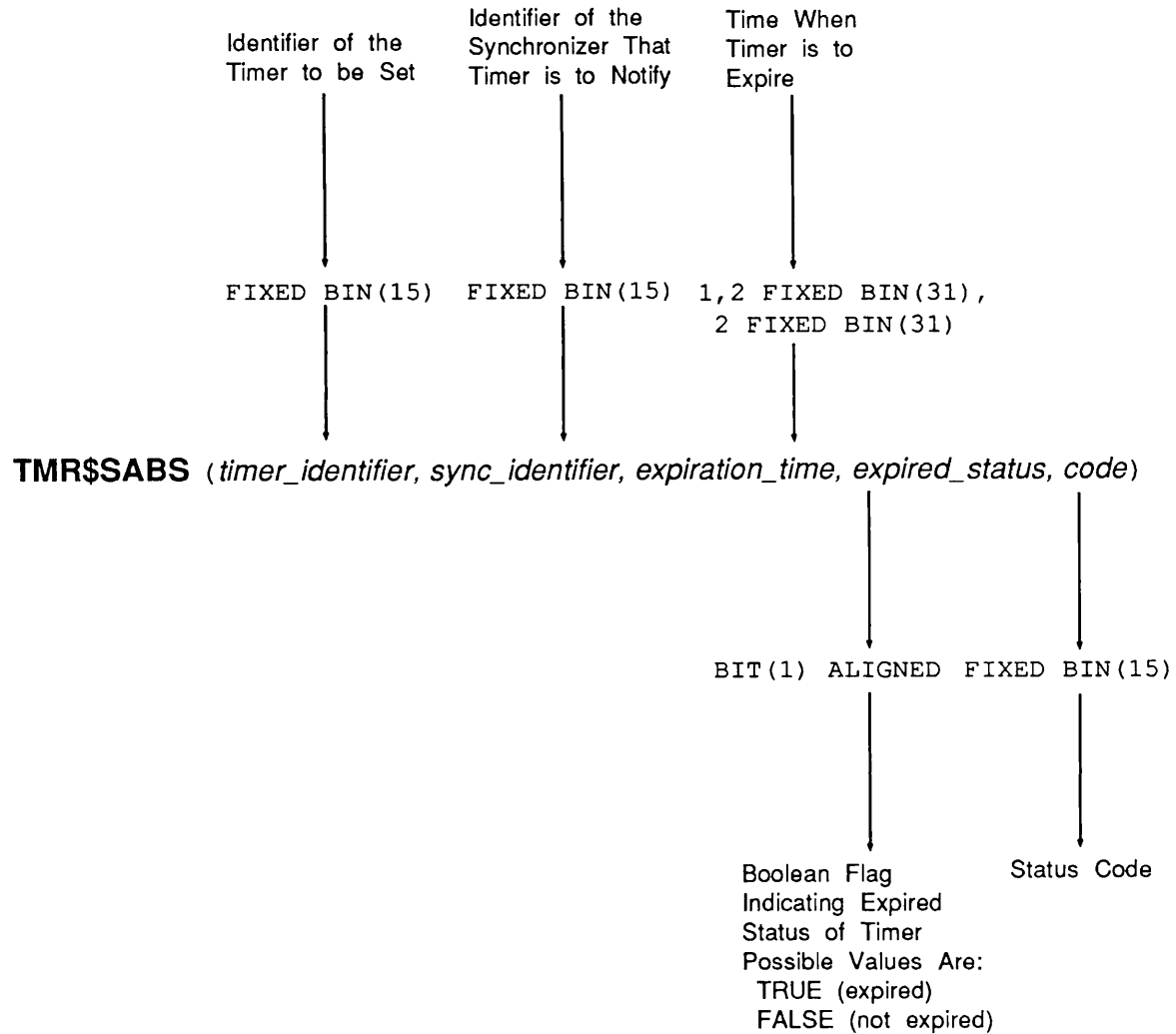


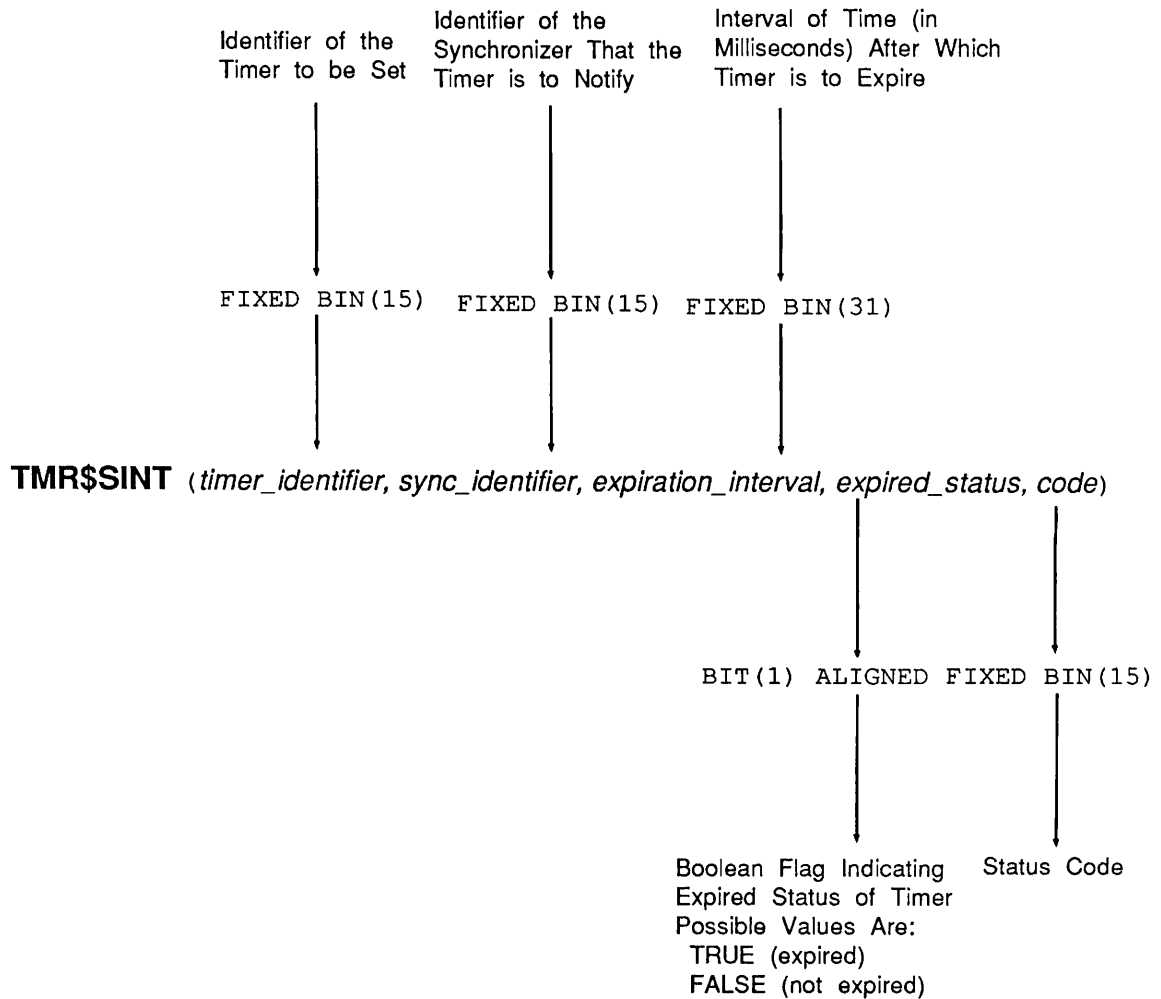
TMR\$CREA**Create a Timer**

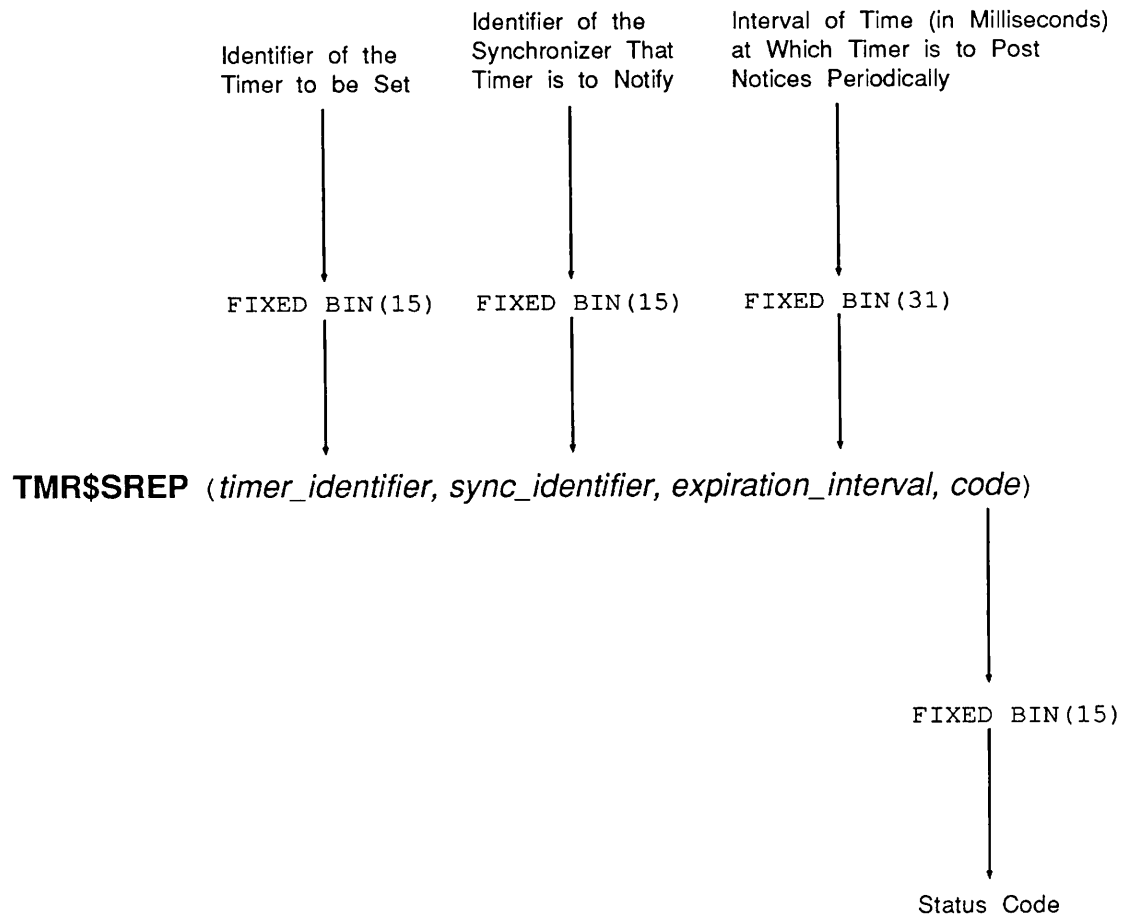
TMR\$DEST**TMR\$DEST****Destroy a Timer**

TMR\$GTMR**Return Information About a Timer**

TMR\$LIST**TMR\$LIST****List Timers Within This Server**

TMR\$SABS**Set an Absolute Timer**

TMR\$SINT**TMR\$SINT****Set an Interval Timer**

TMR\$SREP**Set a Repetitive Timer**

Sample Programs

Programs Accessing Synchronizers and Groups

The two sample programs that follow perform the same operations. The first version is written in PL/I and the second in F77. Both versions:

- Create an event synchronizer with an initial notice count of 1
- Create an event group with 2 priority levels
- Move the event synchronizer just created into priority level 1 of the event group just created
- Retrieve information about this synchronizer
- List the event groups that belong to this server

PL/I Example

```
/* PL/I program invoking SYN$CREA, SYN$GCRE, SYN$MVTO, */
/* SYN$DEST, SYN$GDST, SYN$GLST, and SYN$INFO          */

SyncExample: PROCEDURE OPTIONS(MAIN);

%include 'syscom>sync_codes.ins.pl1';
%include 'syscom>keys.ins.pl1';
%include 'syscom>errormsghdlr.ins.pl1';

dcl syn$crea external entry (fixed bin(15), fixed bin(15), fixed bin(15));
dcl syn$gcre external entry (fixed bin(15), fixed bin(15), fixed bin(15));
dcl syn$mvto external entry (fixed bin(15), fixed bin(15), fixed bin(15),
                             (3)fixed bin, fixed bin(15));
dcl syn$info external entry (fixed bin(15), pointer, fixed bin(15));
dcl syn$glst external entry (fixed bin(15), (*)fixed bin,
                             fixed bin(15), fixed bin(15));
dcl syn$dest external entry (fixed bin(15), fixed bin(15));
dcl syn$gdst external entry (fixed bin(15), fixed bin(15));
dcl er$print external entry (fixed bin(15), char(*)var, fixed bin(15),
                             char(*)var, char(*)var);
```

```
dcl notice, SyncNum, status fixed bin(15);
dcl PriorityLevels, GroupNum, level fixed bin(15);
dcl fcu(1:3) fixed bin(15);
dcl GroupListSize, GroupCount fixed bin(15);
dcl GroupList(1:5) fixed bin(15);
dcl info like SyncInfoRec;
dcl InfoPtr pointer;

/* Create a synchronizer with one notice count. */

notice = 1;
call syn$crea (notice, SyncNum, status);

/* Output message; if there is an error, do not continue. */

call er$print (k$nrtn, ssc$sync, status, 'Cannot create synchronizer',
              'SyncExample');

/* Create a group with 2 priority levels. */

PriorityLevels = 2;
call syn$gcre (PriorityLevels, GroupNum, status);
if (status ^= SYN_SC$OK)

/* Output message; if there is an error, clean it up. */

    then do;
        call er$print (k$irtn, ssc$sync, status, 'Cannot create group',
                      'SyncExample');
        call syn$dest (SyncNum, status);
        return;
    end;

/* Move the synchronizer into level 1 of the group. */

level = 1;
fcu(1) = 0;
fcu(2) = 0;
fcu(3) = 1;
call syn$mvto (GroupNum, SyncNum, level, fcu, status);
if (status ^= SYN_SC$OK)
    then do;
        call er$print (k$irtn, ssc$sync, status,
                      'Cannot move sync to group', 'SyncExample');
        call syn$gdst (GroupNum, status);
        call syn$dest (SyncNum, status);
        return;
    end;
```

```
/* Get information about synchronizer. */

InfoPtr = ADDR(info);
call syn$info (SyncNum, InfoPtr, status);
if (status ^= SYN_SC$OK)
    then do;
        call er$print (k$irtn, ssc$sync, status,
            'Cannot get info on sync', 'SyncExample');
        call syn$gdst(GroupNum, status);
        call syn$dest(SyncNum, status);
        return;
    end;

/* Get a list of groups which belong to this server. */

GroupListSize = 5;
call syn$glst (GroupListSize, GroupList, GroupCount, status);
if (status ^= SYN_SC$OK)
    then do;
        call er$print (k$irtn, ssc$sync, status,
            'Cannot get list of groups', 'SyncExample');
        call syn$gdst(GroupNum, status);
        call syn$dest(SyncNum, status);
        return;
    end;

end SyncExample;
```

F77 Example

```
C F77 program invoking SYN$CREA, SYN$GCRE, SYN$MVTO,
C SYN$GLST, and SYN$INFO
C
$INSERT SYSCOM>SYNC_CODES.INS.FTN
$INSERT SYSCOM>KEYS.INS.FTN
C
C Declarations
      INTEGER*2    NOTICE, SYNCNUM, STATUS, PRIORITYLEVELS, GROUPNUM
      INTEGER*2    FCU(3), GROUPLISTSIZE, GROUPCOUNT, GROUPLIST(5)
      INTEGER*2    SUBSYSTEM(3), LEVEL
      INTEGER*4    INFOPTR
      INTEGER*2    SYNCINFOREC(6), INGROUP, GRPNUM, LEV
      INTEGER*2    FORCLIENT(3), NAME(5)
      CHARACTER*8   ROUTINE
      CHARACTER*5   SSNAME

C Equivalences
      EQUIVALENCE (INGROUP, SYNCINFOREC(1))
      EQUIVALENCE (GRPNUM, SYNCINFOREC(2))
      EQUIVALENCE (LEV, SYNCINFOREC(3))
      EQUIVALENCE (FORCLIENT, SYNCINFOREC(4))
      EQUIVALENCE (NAME(2), ROUTINE)
      EQUIVALENCE (SUBSYSTEM(2), SSNAME)

C Create a synchronizer with one notice count.
      NAME(1) = 8
      SUBSYSTEM(1) = 5
      SSNAME = 'SYNC$'
      NOTICE = 1
      CALL SYN$CREA (NOTICE, SYNCNUM, STATUS)
      IF (STATUS .EQ. SYE$OK) GO TO 10
      ROUTINE = 'SYN$CREA'
      CALL ER$PRINT (K$NRTN, SUBSYSTEM, STATUS, 0, NAME)

C Create a group with 2 priority levels.
10    PRIORITYLEVELS = 2
      CALL SYN$GCRE (PRIORITYLEVELS, GROUPNUM, STATUS)
      IF (STATUS .EQ. SYE$OK) GO TO 20
      ROUTINE = 'SYN$GCRE'
      CALL ER$PRINT(K$IRTN, SUBSYSTEM, STATUS, 0, NAME)
      CALL SYN$DEST(SYNCNUM, STATUS)
      CALL EXIT
```


C Move the synchronizer into level 1 of the group.

```
20   FCU(1) = 0
      FCU(2) = 0
      FCU(3) = 1
      LEVEL = 1
      CALL SYN$MVTO (GROUPNUM, SYNCNUM, LEVEL, FCU, STATUS)
      IF (STATUS .EQ. SYE$OK) GO TO 30
      ROUTINE = 'SYN$MVTO'
      CALL ER$PRINT(K$IRTN, SUBSYSTEM, STATUS, 0, NAME)
      CALL SYN$GDST(GROUPNUM, STATUS)
      CALL SYN$DEST(SYNCNUM, STATUS)
      CALL EXIT
```

C Get information about synchronizer.

```
30   INFOPTR = LOC(SYNCINFOREC)
      CALL SYN$INFO (SYNCNUM, INFOPTR, STATUS)
      IF (STATUS .EQ. SYE$OK) GO TO 40
      ROUTINE = 'SYN$INFO'
      CALL ER$PRINT(K$IRTN, SUBSYSTEM, STATUS, 0, NAME)
      CALL SYN$GDST(GROUPNUM, STATUS)
      CALL SYN$DEST(SYNCNUM, STATUS)
      CALL EXIT
```

C Get a list of groups which belong to this server.

```
40   GROUPLISTSIZE = 5
      CALL SYN$GLST (GROUPLISTSIZE, GROUPLIST, GROUPCOUNT, STATUS)
      IF (STATUS .EQ. SYE$OK) GO TO 50
      ROUTINE = 'SYN$GLST'
      CALL ER$PRINT(K$IRTN, SUBSYSTEM, STATUS, 0, NAME)
      CALL SYN$GDST(GROUPNUM, STATUS)
      CALL SYN$DEST(SYNCNUM, STATUS)
```

```
50   CALL EXIT
      END
```

Programs Accessing Timers

The following two program examples perform the same operations: they create a repetitive timer and get information about the timer. The first example is written in PL/I and the second in F77.

PL/I Example

```
/* PL/I program creating a repetitive timer and getting */
/* information about it. */

TmrExample: PROCEDURE OPTIONS(MAIN);

%include 'syscom>timermik.ins.pll';
%include 'syscom>keys.ins.pll';
%include 'syscom>errormsghdlr.ins.pll';

dcl tmr$dest external entry (fixed bin(15), bit(1) aligned, fixed bin(15));
dcl tmr$gtmr external entry (fixed bin(15), pointer, fixed bin(15));
dcl tmr$crea external entry (fixed bin(15), fixed bin(15), fixed bin(15));
dcl er$print external entry (fixed bin(15), char(*)var, fixed bin(15),
                             char(*)var, char(*)var);

dcl kind, TmrNum, status fixed bin(15);
dcl info like RepTimerInfo;
dcl InfoPtr pointer;

/* Create a repetitive timer. */

kind = Repetitive;
call tmr$crea (kind, TmrNum, status);
call er$print (k$nrtn, ssc$timer, status,
              'Cannot create timer', 'SyncExample');

/* Get information about timer. */

InfoPtr = ADDR(info);
call tmr$gtmr (TmrNum, InfoPtr, status);

call er$print (k$irtn, ssc$timer, status,
              'Cannot get info on timer', 'SyncExample');
call tmr$dest (TmrNum, Expired, Status);
end TmrExample;
```

F77 Example

C F77 program creating a repetitive timer and
C getting information about it.

\$INSERT SYSCOM>TIMERMIK.INS.FTN

\$INSERT SYSCOM>KEYS.INS.FTN

C Declarations

```
INTEGER*2 KIND, TMRNUM, STATUS, SUBSYSTEM(4), NAME(5), EXPIRED
INTEGER*4 INFOPTR, REMAININGTIME, SUCCEEDINGINTERVALS
INTEGER*2 REPTIMERINFO(8), TIMER, SYNC, STATE, KINDINFO
CHARACTER*6 SSNAME
CHARACTER*8 ROUTINE
```

C Equivalences

```
EQUIVALENCE (TIMER, REPTIMERINFO(1))
EQUIVALENCE (SYNC, REPTIMERINFO(2))
EQUIVALENCE (STATE, REPTIMERINFO(3))
EQUIVALENCE (KINDINFO, REPTIMERINFO(4))
EQUIVALENCE (REMAININGTIME, REPTIMERINFO(5))
EQUIVALENCE (SUCCEEDINGINTERVALS, REPTIMERINFO(7))
EQUIVALENCE (NAME(2), ROUTINE)
EQUIVALENCE (SUBSYSTEM(2), SSNAME)
```

C Create a repetitive timer.

```
NAME(1) = 8
SUBSYSTEM(1) = 6
SSNAME = 'TIMER$'
KIND = REP
CALL TMR$CREA (KIND, TMRNUM, STATUS)
IF (STATUS .EQ. TME$OK) GO TO 10
ROUTINE = 'TMR$CREA'
CALL ER$PRINT(K$NRTN, SUBSYSTEM, STATUS, 0, NAME)
```

C Get information about timer.

```
10  INFOPTR = LOC(REPTIMERINFO)
    CALL TMR$GTMR (TMRNUM, INFOPTR, STATUS)
    ROUTINE = 'TMR$GTMR'
    CALL ER$PRINT(K$IRTN, SUBSYSTEM, STATUS, 0, NAME)
    CALL TMR$DEST(TMRNUM, EXPIRED, STATUS)

20  CALL EXIT
    END
```

Programs Using InterServer Communications

The two sample programs that follow demonstrate the two sides of an ISC session. The first program shows the session initiator side of the session, the second program shows the session recipient side of the session. Both programs are written in PL/I. Both programs use a directory named HLNFS, which you must create on your system. The session initiator program:

- Looks up the Low Level Name of the session recipient.
- Requests an ISC session with a default session configuration.
- Sends a Normal message that contains a data part.

The session recipient program:

- Catalogs the server's Low Level Name.
- Creates an event group and moves synchronizers into it.
- Sets local configuration parameters.
- Sends and receives Normal messages that contain a data part.
- Handles exceptions of different types.

PL/I Example: ISC Session Initiator Program

```
/* PL/I program for ISC session initiator */

SESSION_INITIATOR: procedure options (main);

/* Include Key Files and Structure Files: */

%include 'syscom>isc_keys.ins.pll';
%include 'syscom>isc_structures.ins.pll';
%include 'syscom>sync_codes.ins.pll';
%include 'syscom>errormsghdlr.ins.pll';
%include 'syscom>keys.ins.pll';
%include 'syscom>errd.ins.pll';

/* Establish Constants */

%replace TRUE      by '1'b;
%replace FALSE    by '0'b;
%replace NORMAL    by '0'b; /* normal message flag for IS$SM and IS$RM */
%replace MESSAGE_SIZE_IN_BYTES by 1024;
```

```

/* Declare Procedures & Functions: */

declare is$ab external entry    (fixed bin, fixed bin, fixed bin)
                                returns(ptr options(short));
declare is$rs external entry    (ptr options(short), ptr options(short),
                                ptr options(short), fixed bin, ptr options(short), fixed bin);
declare is$grs external entry   (fixed bin, ptr options(short),
                                ptr options(short), fixed bin, ptr options(short), fixed bin);
declare is$sm external entry     (fixed bin, ptr options(short),
                                bit(1) aligned, fixed bin);
declare is$ts external entry     (fixed bin, fixed bin, ptr
                                options(short), fixed bin);
declare isn$l external entry     (char(128) var, ptr, fixed bin);
declare syn$wait external entry  (fixed bin, fixed bin);
declare gsnam$ external entry    (char(32) var);
declare er$print external entry  (fixed bin, char(*) var, fixed bin,
                                char(32) var, char(32) var);
declare ioa$ external entry      (char(*), fixed bin);
declare null builtin;
declare addr builtin;

/* LOCAL VARIABLES */

/* Associate structures in your program with the ISC templates */

declare 1 Auth      like InitiatorAuthBlock,
        1 MS        like MessageSpecifier,
        1 LLN       like LowLevelName,
        1 Syncs     like SessionSyncList;

/* Declare other ISC variables */

declare SessionId bin;
declare ISC_code bin;
declare sync_code bin;
declare code bin;
declare ResponseCode bin;
declare HLNf char(128) var;
declare bptr ptr options(short);

/* Declare other non-ISC variables */

declare i fixed bin;
declare session_allocated bit(1) aligned;
declare connect_message char(128) ;
declare buffer (MESSAGE_SIZE_IN_BYTES) fixed bin based;

/* ----- Execution begins here ----- */

/* Initialize variables */

HLNf = 'hlnfs>glenn.hlnf'; /* Recipient's High Level Name File */
session_allocated = FALSE;
bptr = NULL;

```

```

/* Look up the Low Level Name of the server with whom      */
/* you wish to establish a session.                        */

call isn$l (HLNF, addr(LLN), code);

if (code ^= 0)
then
do;
call er$print (k$irtn, 'ERRD', code,
               'No HLNF for server', 'session_initiator');
goto ERROR_CLEANUP;
end;

/* Set the version numbers for all ISC structures used. */

Auth.Version = ISC_VERSION_NUMBER;
Syncs.Version = ISC_VERSION_NUMBER;
LLN.Version = ISC_VERSION_NUMBER;
MS.Version = ISC_VERSION_NUMBER;

/* Set the values of the Message Specifier parameters */
/* for sending a Connect message.                      */

MS.Control.SuppliedLength = 30;
MS.Control.ReturnedLength = 0;
MS.Control.BufferLocation = addr(connect_message);
MS.Data.BufferLength = 0;
MS.Data.BufferLocation = null();

/* Create a Connect message */

connect_message = 'Message from session initiator';

/* Request a session. This session request specifies the */
/* default session configuration and includes a connect message */

call is$rs(addr(LLN), addr(MS), null(), SessionId, addr(Syncs), ISC_code);

if (ISC_code ^= ISC_SC$OK)
then
do;
call er$print (k$irtn, 'ISC', ISC_code,
               'IS$RS to recipient failed', 'session_initiator');
goto ERROR_CLEANUP;
end;

session_allocated = TRUE;
/* Wait for a response to your session request. A response */
/* to the SessionResponsePending synchronizer automatically */
/* wakes up your server.                                   */

call syn$wait (Syncs.SessionResponsePending, sync_code);

if (sync_code ^= SYN_SC$OK)
then
do;
call er$print (k$irtn, 'SYNC', sync_code,
               'Can''t wait on SRpP sync', 'session_initiator');
goto ERROR_CLEANUP;
end;

```

```

/* Get the response to your session request. */

MS.Control.SuppliedLength = 128;
MS.Control.ReturnedLength = 0;
MS.Control.BufferLocation = addr(connect_message);
MS.Data.BufferLength = 0;
MS.Data.BufferLocation = null();

call is$grs(SessionId, null(), addr(Auth),
           ResponseCode, addr(MS), ISC_code);

if (ISC_code ^= ISC_SC$OK)
then
do;
call er$print (k$sirtn, 'ISC', ISC_code,
              'IS$GRS from recipient failed', 'session_initiator');
goto ERROR_CLEANUP;
end;

/* Determine if the session request has been accepted or rejected */

select (ResponseCode);

when (ISC_RC$Accepted)

/* Write your own code here that examines the connect message */
/* returned from IS$GRS in Message Specifier. For example: */
/* if((MS.Control.ReturnedLength >= 0) & (connect_message(1)='A')) */
/* then print out a message indicating service A is being provided.*/
;

when (ISC_RC$ServerTerminate)
do;

/* Write your own code here that examines the connect message */
/* returned from IS$GRS in Message Specifier. */

call er$print (k$sirtn, 'ISC', ResponseCode,
              'Server Terminate on IS$GRS', 'session_initiator');

session_allocated = FALSE;
goto ERROR_CLEANUP;
end;

when (ISC_RC$SystemTerminate)
do;
call er$print (k$sirtn, 'ISC', ResponseCode,
              'System Terminate on IS$GRS', 'session_initiator');
session_allocated = FALSE;
goto ERROR_CLEANUP;
end;

otherwise
goto ERROR_CLEANUP;

end;

```

```

/* Session request was accepted. Prepare to exchange messages. */
/* Initialize the parameters of the Message Specifier.          */

MS.Control.SuppliedLength = 0;
MS.Control.ReturnedLength = 0;
MS.Control.BufferLocation = null();
MS.Data.BufferLength = 0;
MS.Data.BufferLocation = null();

/* Allocate a buffer for the data part of a Normal message.    */
/* This buffer is allocated from the Message Area established   */
/* by the default configuration parameters.                      */

bptr = is$ab (SessionId, MESSAGE_SIZE_IN_BYTES, ISC_code);

if (ISC_code ^= ISC_SC$OK)
then
do;
    call er$print (k$irtn, 'ISC', ISC_code, 'IS$AB failed', 'session_initiator');
    goto ERROR_CLEANUP;
end;

/* Write your message into the data part buffer. The example   */
/* shown here fills the buffer with ascending numbers          */

do i = 1 to MESSAGE_SIZE_IN_BYTES;
    bptr -> buffer (i) = i;
end;

/* Set the parameters of the Message Specifier to describe     */
/* this data part buffer.                                       */

MS.Data.BufferLength = MESSAGE_SIZE_IN_BYTES;
MS.Data.BufferLocation = bptr;

/* Wait for an available slot on the send queue. If a slot is  */
/* available, no wait occurs and the program proceeds immediately. */

call syn$wait (Syncs.ReadyToSend, sync_code);

/* Send the message. The message sent is a Normal message     */
/* consisting of a data part only.                              */

call is$sm (SessionId, addr(MS), NORMAL, ISC_code);

if ISC_code ^= ISC_SC$OK
then
do;
    call er$print (k$irtn, 'ISC', ISC_code, 'IS$SM failed', 'session_initiator');
    goto ERROR_CLEANUP;
end; /* otherwise */
call ioa$('Message successfully sent to session recipient. %.', 100);

/* Send and receive other messages, as needed. When finished,  */
/* terminate your side of the session.                          */

call is$ts (SessionID, 0, null(), ISC_code);
return;

ERROR_CLEANUP:

```



```
/* This error-handling routine terminates your side of the session, */  
/* freeing all allocated resources. */  
  
if (session_allocated)  
    then call is$ts (SessionID, 0, null(), ISC_code);  
return;  
  
end;
```

PL/I Example: ISC Session Recipient Program

```

/* PL/I program for ISC session recipient. This program accepts a */
/* session request with the default parameters and then places */
/* its synchronizers in an event group. It then responds to */
/* exceptions of various types (including receiving all pending */
/* messages). If no exception is pending, it sends and receives */
/* Normal messages. All of these operations are governed by the */
/* posting of notices on synchronizers in the user's event group. */

SESSION_RECIPIENT: procedure options (main);

/* Include Key Files and Structure Files: */

#include 'syscom>isc_keys.ins.pll';
#include 'syscom>isc_structures.ins.pll';
#include 'syscom>sync_codes.ins.pll';
#include 'syscom>srs_codes.ins.pll';
#include 'syscom>errormsghdlr.ins.pll';
#include 'syscom>keys.ins.pll';
#include 'syscom>errd.ins.pll';

/* Establish Constants */

%replace TRUE      by '1'b;
%replace FALSE     by '0'b;

%replace NORMAL     by '0'b; /* Normal message flag for IS$SM, IS$RM */
%replace BUFFERSIZE by 1024; /* Size of Data message buffer */
%replace SESSIONBLOCKSIZE by 657; /* Size of Session Information block */

/* Declare Procedures & Functions */

declare is$ab external entry    (fixed bin, fixed bin, fixed bin)
    returns (ptr options(short));
declare is$fb external entry    (fixed bin, ptr options(short), fixed bin);
declare is$grq external entry   (ptr options(short), ptr options(short),
    ptr options(short), fixed bin, ptr options(short), fixed bin);
declare is$as external entry    (fixed bin, ptr options(short),
    ptr options(short), ptr options(short), fixed bin);
declare is$rm external entry    (fixed bin, ptr options(short),
    bit(1) aligned, fixed bin);

declare is$sm external entry    (fixed bin, ptr options(short),
    bit(1) aligned, fixed bin);
declare is$ts external entry    (fixed bin, fixed bin, ptr
    options(short), fixed bin);
declare is$ce external entry    (fixed bin, fixed bin);
declare is$ge external entry    (fixed bin, fixed bin, ptr
    options(short), fixed bin);
declare isn$c external entry    (char(128) var, ptr, fixed bin);
declare isn$rc external entry   (char(128) var, ptr, fixed bin);
declare syn$gwt external entry   (fixed bin, fixed bin, ptr, fixed bin);

```

```

declare syn$gcre external entry (fixed bin, fixed bin, fixed bin);
declare syn$mvto external entry (fixed bin, fixed bin, fixed bin,
    ptr, fixed bin);

declare srs$gn external entry (fixed bin, fixed bin, char(12) var,
    fixed bin);
declare gsnam$ external entry (char(32) var);
declare er$print external entry (fixed bin, char(*) var, fixed bin,
    char(32) var, char(32) var);
declare str$au external entry (fixed bin(31)) returns(ptr options(short));
declare str$fu external entry (ptr options(short));
declare ioa$ external entry (char(*), fixed bin);
declare null builtin;
declare addr builtin;

/* LOCAL VARIABLES */

/* Declare ISC structures */

declare 1 SessionBlock based,
    2 SessionId fixed bin,
    2 Auth like InitiatorAuthBlock,
    2 Config like SessionConfigurationBlock,
    2 MS like MessageSpecifier,
    2 LLN like LowLevelName,
    2 Syncs like SessionSyncList;

/* Declare Connect message array */

declare my_connect_message char(128);

/* Declare other ISC local variables */

declare SyncGroupId bin;
declare ISC_code bin;
declare sync_code bin;
declare srs_code bin;
declare code bin;
declare ResponseCode bin;
declare HLNf char(128) var;
declare myLLN like LowLevelName;
declare exception fixed bin;

/* Declare variables for the rest of the program */

declare MyNode char(32) var;
declare SessionRequestPending fixed bin;
declare SyncNotified fixed bin;
declare SessionBlockPtr ptr options(short);
declare SendBufferPtr ptr options(short);
declare BufferLength fixed bin;
declare i fixed bin;
declare received_all bit(1) aligned;
declare buffer (BUFFERSIZE) fixed bin based;

```

```
/* ----- Execution begins here ----- */

/* Specify the High Level Name File location */

HLNF = 'hlnfs>glenn.hlnf';

/* Determine your server name and node name, and construct */
/* a Low Level Name (LLN). */

call gsnam$ (MyNode);
myLLN.NodeName = MyNode;
myLLN.ForClientUse = 5; /* perhaps a code for a subservice provided by
                        this server */

call srs$gn (0, SessionRequestPending, myLLN.ServerName, srs_code);
if (srs_code ^= SRS_SC$OK)
then
do;
call er$print (k$irtn, 'SRS', srs_code,
              'Can''t get my server name', 'session_recipient');
return;
end;

/* Catalog (or recatalog) your Low Level Name (LLN) */

call isn$c (HLNF, addr(myLLN), code);

if (code = e$exst)
then call isn$rc (HLNF, addr(myLLN), code);

if (code ^= 0)
then
do;
call er$print (k$irtn, 'ERRD', code,
              'Can''t (re)catalog my HLNF', 'session_recipient');
return;
end;

/* Create an event group with four priority levels. */

call syn$gcre (4, SyncGroupId, sync_code);

if Sync_code ^= SYN_SC$OK
then
do;
call er$print (k$irtn, ssc$sync, sync_code, 'Cannot create sync group',
              'session_recipient');
return;
end;
```

```

/* Move your SessionRequestPending synchronizer into the event group. */

call syn$mvto (SyncGroupId, SessionRequestPending, 1, null(), sync_code);

if sync_code ^= SYN_SC$OK
then
do;
call er$print (k$irtn, ssc$sync, sync_code, 'Cannot add SRqP to group',
'session recipient');
return;
end;

/* The following code is a loop that includes all of the */
/* processing for this ISC session. The loop is only exited */
/* when an error occurs during processing. */

do while (1 = 1);

/* Wait on the event group containing your */
/* SessionRequestPending synchronizer. */

call syn$gwt (SyncGroupId, SyncNotified, SessionBlockPtr, sync_code);

if sync_code ^= SYN_SC$OK
then
do;
call er$print(k$irtn, ssc$sync, sync_code, 'Cannot wait on sync group',
'session recipient');
return;
end;

/* Wait was ended by a notice posted to one of the synchronizers in */
/* your event group. Determine which synchronizer in the event group */
/* was notified. */

select (SyncNotified);

when (SessionRequestPending)
do;

/* Allocate a session block. This session block will */
/* contain all of the structures needed during this */
/* session. Session block is pointed to by SessionBlockPtr. */

SessionBlockPtr = str$au (SESSIONBLOCKSIZE);

/* Set the version numbers for the ISC structures used. */

SessionBlockPtr -> SessionBlock.LLN.Version = ISC_VERSION_NUMBER;
SessionBlockPtr -> SessionBlock.MS.Version = ISC_VERSION_NUMBER;
SessionBlockPtr -> SessionBlock.Config.Version = ISC_VERSION_NUMBER;
SessionBlockPtr -> SessionBlock.Auth.Version = ISC_VERSION_NUMBER;

```

```

/* Set the Message Specifier parameters to receive a      */
/* Connect message as part of the get session request.    */

SessionBlockPtr -> SessionBlock.MS.Control.SuppliedLength = 128;
SessionBlockPtr -> SessionBlock.MS.Control.ReturnedLength = 0;
SessionBlockPtr -> SessionBlock.MS.Control.BufferLocation =
    addr(my_connect_message);
SessionBlockPtr -> SessionBlock.MS.Data.BufferLength = 0;
SessionBlockPtr -> SessionBlock.MS.Data.BufferLocation = null();

/* Get the session request.                                */

call is$grq(addr(SessionBlockPtr -> SessionBlock.LLN),
            addr(SessionBlockPtr -> SessionBlock.MS),
            addr(SessionBlockPtr -> SessionBlock.Config),
            SessionBlockPtr -> SessionBlock.SessionId,
            addr(SessionBlockPtr -> SessionBlock.Auth), ISC_code);

if (ISC_code ^= ISC_SC$OK)
    then
        do;
            call er$print (k$irtn, ssc$isc, ISC_code,
                'GRQ from client failed', 'session_recipient');
            call str$fu (SessionBlockPtr);
            return;
        end;

/* Write some code here that examines the initiator authentication */
/* information provided by IS$GRQ to the SessionBlock.Auth structure. */
/* Determine whether or not to accept a session with this server.    */
/* Read the connect message (if any) supplied by the session initiator. */
/* This program assumes that the session request is acceptable.      */

/* Set Syncs version number. */

SessionBlockPtr -> SessionBlock.Syncs.Version = ISC_VERSION_NUMBER;

/* Create a Connect message. */

my_connect_message = 'Connected to ISC Server Glenn';

/* Initialize Message Specifier parameters for */
/* sending the connect message.                */

SessionBlockPtr -> SessionBlock.MS.Version = ISC_VERSION_NUMBER;
SessionBlockPtr -> SessionBlock.MS.Control.SuppliedLength = 29;
SessionBlockPtr -> SessionBlock.MS.Control.ReturnedLength = 0;
SessionBlockPtr -> SessionBlock.MS.Control.BufferLocation =
    addr(my_connect_message);
SessionBlockPtr -> SessionBlock.MS.Data.BufferLength = 0;
SessionBlockPtr -> SessionBlock.MS.Data.BufferLocation = null();

```

```
/* Accept the session with the default configuration. */
/* This call to IS$AS also sends a Connect message to */
/* the session initiator. */

call is$as(SessionBlockPtr -> SessionBlock.SessionId,
          addr(SessionBlockPtr -> SessionBlock.MS), null(),
          addr(SessionBlockPtr -> SessionBlock.Syncs), ISC_code);

if (ISC_code ^= ISC_SC$OK)
then
do;
call er$print (k$sirtn, ssc$isc, ISC_code,
              'IS$AS failed', 'session_recipient');
call str$fu (SessionBlockPtr);
return;
end; /* Session is established. */

/* Move other synchronizers into your event group. */

call syn$mvto (SyncGroupId, SessionBlockPtr -> SessionBlock.Syncs.
              ExceptionPending, 2, SessionBlockPtr, sync_code);

if sync_code ^= SYN_SC$OK
then
do;
call er$print (k$sirtn, ssc$sync, sync_code,
              'Cannot add ExceptionPending sync to group', 'session_recipient');
call str$fu (SessionBlockPtr);
return;
end;

call syn$mvto (SyncGroupId, SessionBlockPtr -> SessionBlock.Syncs.
              ReadyToReceive, 3, SessionBlockPtr, sync_code);

if sync_code ^= SYN_SC$OK
then
do;
call er$print (k$sirtn, ssc$sync, sync_code,
              'Cannot add ReadyToReceive sync to group', 'session_recipient');
call str$fu (SessionBlockPtr);
return;
end;

call syn$mvto (SyncGroupId, SessionBlockPtr -> SessionBlock.Syncs.
              ReadyToSend, 4, SessionBlockPtr, sync_code);
```

```
        if sync_code ^= SYN_SC$OK
        then
            do;
                call er$print (k$irtn, ssc$sync, sync_code,
                    'Cannot add ReadyToSend sync to group', 'session recipient');
                call str$fu (SessionBlockPtr);
                return;
            end;

    end;

/* The following series of 'when' clauses respond to notices being */
/* posted on various synchronizers in the event group. The order */
/* in which the program checks the synchronizers is as follows: */
/* ExceptionPending, ReadyToSend and ReadyToReceive */

/* Exception processing upon notification of ExceptionPending. */

    when (SessionBlockPtr -> SessionBlock.Syncs.ExceptionPending)
    do;
        /* Get the exception. */

        call is$ge (SessionBlockPtr -> SessionBlock.SessionId, Exception,
            addr(SessionBlockPtr -> SessionBlock.MS), ISC_code);

        if ISC_code ^= ISC_SC$OK
        then
            do;
                call er$print (k$irtn, ssc$isc, ISC_code,
                    'Cannot get exception', 'session recipient');
                call str$fu (SessionBlockPtr);
                return;
            end;

        /* Clear the exception. */

        call is$ce (SessionBlockPtr -> SessionBlock.SessionId, ISC_code);

        if ISC_code ^= ISC_SC$OK
        then
            do;
                call er$print (k$irtn, ssc$isc, ISC_code,
                    'Cannot clear Exception', 'session recipient');
                call str$fu (SessionBlockPtr);
                return;
            end;

        select (Exception);
```



```

/* Handling an exception caused by a server terminate or */
/* system terminate.                                     */

when (ISC_EX$ServerTerminate, ISC_EX$SystemTerminate)
do;

/* If the exception was a ServerTerminate, the initiating */
/* server may have sent a Connect message. Check the     */
/* Message Specifier Control area. After clearing the     */
/* exception, you can optionally receive all remaining   */
/* messages on the receive queue. Receive all remaining  */
/* Normal messages until IS$RM returns an ISC_code of    */
/* Terminated.                                          */

received_all = FALSE;

do while (^received_all);

/* Set the Message Specifier to receive a message. */

SessionBlockPtr -> SessionBlock.MS.Control.SuppliedLength = 128;
SessionBlockPtr -> SessionBlock.MS.Control.ReturnedLength = 0;
SessionBlockPtr -> SessionBlock.MS.Control.BufferLocation =
    addr(my_connect_message);
SessionBlockPtr -> SessionBlock.MS.Data.BufferLength = 0;
SessionBlockPtr -> SessionBlock.MS.Data.BufferLocation = null();

/* Receive a message. */

call is$rm (SessionBlockPtr -> SessionBlock.SessionId,
    addr(SessionBlockPtr -> SessionBlock.MS), NORMAL, ISC_code);

select (ISC_code);

when (ISC_SC$Terminated)
    received_all = TRUE;

when (ISC_SC$OK)
do;
    call ioa$ ('Message successfully received from
        session initiator. %.', 100);

/* If the message contains a data part, you should */
/* insert your own code here to read the message */
/* from the data buffer.                          */
/* own address space using MS.Data.BufferLocation */
/* as the pointer to the data and MS.Data.BufferLength */
/* as a count of the bytes in the message.        */

```

```
/* Free the buffer that contained the data part      */
/* message.                                          */

    call is$fb (SessionBlockPtr -> SessionBlock.SessionId,
               SessionBlockPtr ->
               SessionBlock.MS.Data.BufferLocation, ISC_code);

    if (ISC_code ^= ISC_SC$OK)
    then
        do;
            call er$print (k$irtn, ssc$isc, ISC_code,
                          'Free Buffer failed', 'session_recipient');
            call str$fu (SessionBlockPtr);
            return;
        end;
    end;                                /* of okay status */

otherwise
    do;
        call er$print (k$irtn, ssc$isc, ISC_code,
                      'Receive Message failed', 'session_recipient');
    end;
end;                                /* of select */
end;                                /* of do while not all received loop */

/* Terminate your side of the session.      */

call is$ts (SessionBlockPtr -> SessionBlock.SessionId, 0,
           null(), ISC_code);

call str$fu (SessionBlockPtr);

if ISC_code ^= ISC_SC$OK
then
    do;
        call er$print (k$irtn, ssc$isc, ISC_code,
                      'Cannot terminate session', 'session_recipient');
        return;
    end;
end;

/* Handling an Exception caused by a delivery failure. */

when (ISC_EX$DeliveryFailure)
do;
```

```

        /* Write your own code here that resynchronizes message */
        /* exchange with the other server, instructing it to      */
        /* retransmit any possibly lost messages.                  */

    end;

otherwise
do;
    call er$print (k$irtn, ssc$isc, Exception,
        'Unknown Exception returned from is$ge', 'session recipient');
    call str$fu (SessionBlockPtr);
    return;
end;

end;      /* of select on Exception type */
end;      /* of Exception Pending sync notified */

/* Sending a message upon notification of ReadyToSend. */

when (SessionBlockPtr -> SessionBlock.Syncs.ReadyToSend)
do;

    /* Allocate a buffer to use in sending the message. */

    BufferLength = BUFFERSIZE;

    SendBufferPtr = is$ab (SessionBlockPtr -> SessionBlock.SessionId,
        BufferLength, ISC_code);

    if ISC_code ^= ISC_SC$OK
    then
    do;
        call er$print (k$irtn, ssc$isc, ISC_code,
            'Cannot allocate send message buffer', 'session recipient');
        call str$fu (SessionBlockPtr);
        return;
    end;

    /* Set the Message Specifier for sending a message. */

    my_connect_message = 'Message from the Recipient ISC Server';

    SessionBlockPtr -> SessionBlock.MS.Control.SuppliedLength = 37;
    SessionBlockPtr -> SessionBlock.MS.Control.ReturnedLength = 0;
    SessionBlockPtr -> SessionBlock.MS.Control.BufferLocation =
        addr(my_connect_message);
    SessionBlockPtr -> SessionBlock.MS.Data.BufferLength = BUFFERSIZE;
    SessionBlockPtr -> SessionBlock.MS.Data.BufferLocation = SendBufferPtr;

    /* Write your own code here to copy the data part of your */
    /* message into the buffer. For example:                    */

    do i = 1 to BUFFERSIZE;
        SendBufferPtr -> buffer(i) = i;
    end;

```

```

/* Send the message. */

call is$sm (SessionBlockPtr -> SessionBlock.SessionId,
  addr(SessionBlockPtr -> SessionBlock.MS), NORMAL, ISC_code);

if ISC_code ^= ISC_SC$OK
then
  do;
    call er$print (k$irtn, ssc$isc, ISC_code,
      'Cannot send message', 'session recipient');
    call str$fu (SessionBlockPtr);
    return;
  end;
  call ioa$('Message successfully sent to session initiator. %.', 100);
end; /* of ReadyToSend sync */

/* Receiving a Normal message upon notification of ReadyToReceive. */

when (SessionBlockPtr -> SessionBlock.Syncs.ReadyToReceive)
do;

  /* Set the Message Specifier for receiving a message that */
  /* consists of a data part and a control part. */

  SessionBlockPtr -> SessionBlock.MS.Control.SuppliedLength = 128;
  SessionBlockPtr -> SessionBlock.MS.Control.ReturnedLength = 0;
  SessionBlockPtr -> SessionBlock.MS.Control.BufferLocation =
    addr(my_connect_message);
  SessionBlockPtr -> SessionBlock.MS.Data.BufferLength = 0;
  SessionBlockPtr -> SessionBlock.MS.Data.BufferLocation = null();

  /* Receive the message. */

  call is$rm (SessionBlockPtr -> SessionBlock.SessionId,
    addr(SessionBlockPtr -> SessionBlock.MS), NORMAL, ISC_code);

  select (ISC_code);

    when (ISC_SC$OK)
    do;
      call ioa$('Message successfully received from
        session initiator. %.', 100);

      /* If the message contains a data part, you should */
      /* insert your own code here to read the message */
      /* from the data buffer. For example, you could */
      /* copy the message to your own address space using */
      /* MS.Data.BufferLocation as the pointer to the */
      /* data and MS.Data.BufferLength as the count of */
      /* the bytes in the message. You should also write */
      /* code that examines the control part of the */
      /* message. */

```

```

/* Free the buffer that contained the data part message */

call is$fb (SessionBlockPtr -> SessionBlock.SessionId,
  SessionBlockPtr ->
  SessionBlock.MS.Data.BufferLocation, ISC_code);

if (ISC_code ^= ISC_SC$OK)
then
  do;
    call er$print (k$irtn, ssc$isc, ISC_code,
      'Free Buffer failed', 'session_recipient');
    call str$fu (SessionBlockPtr);
    return;
  end;
end; /* of okay status */

when (ISC_SC$NothingYet) /* IS$RM status code */
; /* Ignore it. */

when (ISC_SC$Exception) /* IS$RM status code */
do;

  /* Get the exception. */

  call is$ge (SessionBlockPtr -> SessionBlock.SessionId,
    Exception, addr(SessionBlockPtr -> SessionBlock.MS),
    ISC_code);

  if ISC_code ^= ISC_SC$OK
  then
    do;
      call er$print (k$irtn, ssc$isc, ISC_code,
        'Cannot get exception', 'session_recipient');
      call str$fu (SessionBlockPtr);
      return;
    end;

  /* Clear the exception. */

  call is$ce (SessionBlockPtr -> SessionBlock.SessionId,
    ISC_code);

  if ISC_code ^= ISC_SC$OK
  then
    do;
      call er$print (k$irtn, ssc$isc, ISC_code,
        'Cannot clear Exception', 'session_recipient');
      call str$fu (SessionBlockPtr);
      return;
    end;
end;

```

```
/* Process the different types of exceptions. */

select (Exception);

when (ISC_EX$ServerTerminate, ISC_EX$SystemTerminate)
do;

/* You can include code here that receives messages */
/* pending on your receive queues. */

    call is$ts (SessionBlockPtr -> SessionBlock.SessionId, 0,
               null(), ISC_code);
    call str$fu (SessionBlockPtr);

    if ISC_code ^= ISC_SC$OK
    then
    do;
        call er$print (k$irtn, ssc$isc, ISC_code,
                      'Cannot terminate session', 'session recipient');
        return;
    end;
end;

when (ISC_EX$DeliveryFailure)
do;
    /* Write your own code here to resynchronize */
    /* message exchange with the other server. */
end;

otherwise
do;
    call er$print (k$irtn, ssc$isc, Exception,
                  'Unknown Exception returned from is$ge', 'session
recipient');
    call str$fu (SessionBlockPtr);
    return;
end;

end; /* of select on exception type. */
end; /* of when IS$RM status code = exception. */

otherwise
do;
    call er$print (k$irtn, ssc$isc, ISC_code,
                  'Receive Message failed', 'session_recipient');
    call str$fu (SessionBlockPtr);
    return;
end;

end; /* of select on status code from IS$RM */

end; /* of ReadyToReceive sync. */
```

```
    end; /* of select on synchronizer notified. */  
end; /* of do while (1 = 1) loop. */  
end; /* of session_recipient program. */
```

Status Codes

This appendix lists the status codes returned by synchronizer, timer, and ISC subroutines. It also lists the codes returned by individual subroutines to indicate the nature of an event or status.

Synchronizer Status Codes

The following status codes are defined in synchronizer include files for programs written in PL/I, FTN, and Pascal. The status codes defined in the PMA and C synchronizer include files are identical to the status codes defined in the PL/I synchronizer include file.

	<i>PL/I</i>	<i>FTN</i>	<i>Pascal</i>	<i>Meaning</i>
0	SYN_SC\$OK	SYE\$OK	SYN_OK	Success.
1	SYN_SC\$NoResources	SYE\$NR	SYN_NoResources	No resources available.
2	SYN_SC\$InvNoticeCount	SYE\$IN	InvNoticeCount	Notice count is less than 0.
3	SYN_SC\$InvSyncNum	SYE\$IS	InvSyncNum	Sync identifier is invalid.
4	SYN_SC\$SyncInGroup	SYE\$IG	SyncInGroup	Operation not allowed on sync in group.
5	SYN_SC\$SyncHasWaiter	SYE\$HW	SyncHasWaiter	Operation not allowed on sync with waiters.
6	SYN_SC\$InvTimeInt	SYE\$IT	InvTimeInt	Timer interval cannot be <= 0.
7	SYN_SC\$InvGroupNum	SYE\$GN	InvGroupNum	Group identifier is invalid.

	<i>PL/I</i>	<i>FTN</i>	<i>Pascal</i>	<i>Meaning</i>
8	SYN_SC\$SyncNotInGroup	SYE\$NG	SyncNotInGroup	Sync must be in group for this operation.
9	SYN_SC\$InvPriority	SYE\$IP	InvPriority	Priority must be greater than 0.
10	SYN_SC\$ListTooSmall	SYE\$LS	SYN_ListTooSmall	List provided by caller is too short.
11	SYN_SC\$MaxSyncsAlloc	SYE\$MS	MaxSyncsAlloc	Maximum number of syncs has been allocated.
12	SYN_SC\$WaitHasAborted	SYE\$WA	WaitHasAborted	Wait operation was aborted.
13	SYN_SC\$InternalError	SYE\$IE	SYN_InternalError	Internal system error occurred.

Synchronizer What_Happened Codes

	<i>PL/I</i>	<i>FTN</i>	<i>Pascal</i>	<i>Meaning</i>
0	SYN_WHC\$Notice	SYW\$N	Notice	Sync returned a notice.
1	SYN_WHC\$NoNotice	SYW\$NN	NoNotice	No notice posted on sync.
2	SYN_WHC\$TimeOut	SYW\$TO	TimeOut	Wait time elapsed.

Timer Status Codes

The following status codes are defined in timer include files for programs written in PL/I, FTN, and Pascal. The status codes defined in the PMA and C timer include files are identical to the status codes defined in the PL/I timer include file.

	<i>PL/I</i>	<i>FTN</i>	<i>Pascal</i>	<i>Meaning</i>
0	TMR_SC\$OK	TME\$OK	TMR_OK	Success.
1	TMR_SC\$NoUpdateRights	TME\$NR	NoRights	No privilege.

	<i>PL/I</i>	<i>FTN</i>	<i>Pascal</i>	<i>Meaning</i>
2	TMR_SC\$InvTimeZone	TME\$IZ	InvTimeZone	Time zone is out of range.
3	TMR_SC\$NoResources	TME\$NS	TMR_NoResources	No resources.
4	TMR_SC\$InvTimeOffset	TME\$IO	InvTimeOffset	Time offset is invalid.
5	TMR_SC\$InvTimer	TME\$IR	InvTimer	Invalid timer identifier.
6	TMR_SC\$InvTimeParameter	TME\$IP	InvTimeParameter	Invalid time input value.
7	TMR_SC\$MaxTimerAlloc	TME\$MT	MaxTimerAlloc	Max. number of timers has been allocated.
8	TMR_SC\$ListTooSmall	TME\$LS	TMR_ListTooSmall	List provided by caller is too short.
9	TMR_SC\$InternalError	TME\$IE	TMR_InternalError	System error has occurred.

ISC Status Codes

The following status codes are defined in ISC include files for programs written in PL/I and FTN. The status codes defined in ISC include files for Pascal, C, and PMA are identical to the status codes defined in the PL/I include files.

	<i>PL/I</i>	<i>FTN</i>	<i>Meaning</i>
0	ISC_SC\$OK	ISE\$OK	Operation was successful.
1	ISC_SC\$AlreadyServer	ISE\$AS	Caller is already a server.
2	ISC_SC\$BadAddress	ISE\$BA	Server name is invalid.
3	ISC_SC\$NoRoom	ISE\$NR	Some system resource required to perform this operation is unavailable.
4	ISC_SC\$NoServer	ISE\$NV	Caller is not a server.
5	ISC_SC\$SoftwareError	ISE\$SE	A serious software error has occurred.
6	ISC_SC\$TooManySessions	ISE\$TM	Caller already owns too many sessions.
9	ISC_SC\$InvalidConfig	ISE\$IC	Configuration is invalid.

	<i>PL/I</i>	<i>FTN</i>	<i>Meaning</i>
11	ISC_SC\$NothingYet	ISE\$NY	Requested item is not available.
12	ISC_SC\$LongMessage	ISE\$LM	Control part of message was too long for the buffer provided and has been truncated.
13	ISC_SC\$Terminated	ISE\$T	Session has been terminated.
14	ISC_SC\$NoSession	ISE\$NS	Caller does not own session.
16	ISC_SC\$BadMessage	ISE\$BM	Message specifier is invalid.
17	ISC_SC\$NotEstablished	ISE\$NE	Session is not established.
18	ISC_SC\$Exception	ISE\$EX	An exception has occurred on the session.
19	ISC_SC\$NoRights	ISE\$NA	Insufficient rights to perform operation.
20	ISC_SC\$BadException	ISE\$BE	Exception is invalid.
21	ISC_SC\$NoQueue	ISE\$NQ	Specified queue does not exist.
22	ISC_SC\$Established	ISE\$ES	Session is already established.
23	ISC_SC\$NoSuchServer	ISE\$SS	Specified server does not exist.
24	ISC_SC\$TooShort	ISE\$TS	Array is too short.
25	ISC_SC\$NoMessageArea	ISE\$NM	No message area exists for this session.
26	ISC_SC\$BadSize	ISE\$BS	Buffer size is invalid.
27	ISC_SC\$BadBuffer	ISE\$BB	Buffer being returned is invalid.
29	ISC_SC\$BadVersion	ISE\$BV	Version number is invalid.

ISC Response Codes Returned by IS\$GRS

	<i>PL/I</i>	<i>FTN</i>	<i>Meaning</i>
105	ISC_RC\$Accepted	ISR\$A	Session request was accepted.
210	ISC_RC\$ServerTerminate	ISR\$T	Session request was rejected by recipient.
211	ISC_RC\$SystemTerminate	ISR\$ST	Session request was rejected by system.

ISC Exception Codes Returned by IS\$GE

	<i>PL/I</i>	<i>FTN</i>	<i>Meaning</i>
208	ISC_EX\$DeliveryFailure	ISX\$DF	Delivery failure, some data may be lost.
210	ISC_EX\$ServerTerminate	ISX\$T	Session has been terminated by other server.
211	ISC_EX\$SystemTerminate	ISX\$ST	Session has been terminated by system.

ISC Phase Codes Returned by IS\$GSS

	<i>PL/I</i>	<i>FTN</i>	<i>Meaning</i>
1	ISC_PC\$Establishing	ISP\$E	Session is being established.
2	ISC_PC\$DataTransfer	ISP\$DT	Session has been established.
3	ISC_PC\$Terminating	ISP\$T	Session is being terminated.

SRS Status Codes

	<i>PL/I</i>	<i>FTN</i>	<i>Meaning</i>
0	SRS_SC\$OK	SSE\$OK	Operation was successful.
1	SRS_SC\$NoResources	SSE\$NR	No resources available.
2	SRS_SC\$BadName	SSE\$BN	Specified server name is invalid.
3	SRS_SC\$NoSuchServer	SSE\$SS	Specified server does not exist.
4	SRS_SC\$ListTooSmall	SSE\$TS	The list provided by the caller is too short.
5	SRS_SC\$AlreadyRegistered	SSE\$AR	Already registered as a server.
6	SRS_SC\$NotRegistered	SSE\$NG	Not registered as a server.
7	SRS_SC\$AlreadyExists	SSE\$AE	Server name in use by another server.

Limits

The following limitations apply to PRIMOS Revision 22.0, and are subject to change for future PRIMOS revisions.

Synchronizer and Timer Limits

- A server can have a maximum combined total of 2048 event groups and event synchronizers. This total includes both individual event synchronizers and event synchronizers that are members of event groups.
- A server can have a maximum of 32 timers.
- No more than 32,767 notices can be pending on an event synchronizer. Attempting to post a notice on an event synchronizer that already has 32,767 notices pending will permanently disable the event synchronizer. It is possible to retrieve notices from the disabled event synchronizer, but it is not possible to post additional notices on it.
- Slave processes cannot use synchronizers or timers.

ISC Limits

- A server can participate in as many as 255 concurrent sessions. The maximum number of concurrent sessions on the system is 511. Each of these sessions involves two servers.
- A server can configure as many message areas as it has available dynamic segment space. When a session initiator configures a message area, a space of the specified size is reserved in that user's dynamic segments for the duration of the session. If a session request fails for lack of message area space, the session initiator can do any one of the following operations. It can terminate one of its existing sessions to free message area space (the server must have been the initiator of that session). It can share a message area with an existing session by setting the **ExistingSessionID** local session parameter. It can request that the System Administrator assign it more dynamic segments.

- A system can support a maximum of 255 virtual circuits. A remote session requires a virtual circuit for Normal message exchange and a second virtual circuit for the exchange of Expedited messages. Therefore, a system can support a maximum of from 127 to 255 concurrent remote sessions, depending on how those sessions are configured. These remote sessions may be owned by a single server or several different servers. Note that virtual circuits are also used by other PRIMOS operations. Refer to Chapter 13 for further details on remote sessions.
- No server may have more than 63 requests for session establishment that are awaiting response. ISC rejects any request for a session with a server that has reached this limit.
- A server can establish a session with itself. ISC considers a server participating in this type of session to be actually participating in two sessions, one as session initiator, one as session recipient.
- Slave processes cannot participate in ISC sessions.
- A server can have as many as 2048 synchronizers. This includes all synchronizers (ISC and non-ISC) for all concurrent sessions.
- An ISC synchronizer can have as many as 32,767 pending notices. Exceeding this number of notices permanently disables ISC use of the synchronizer.

Data Structures

Synchronizer Information Record

Described in Chapter 4.

```
dcl 1 SyncInfoRec based,  
    2 InGroup bit(1) aligned,  
    2 GroupNum fixed bin(15),  
    2 Priority fixed bin(15),  
    2 ForClient (3) fixed bin (15);
```

Absolute Timer Information Record

Described in Chapter 5.

```
dcl 1 AbsTimerInfo based,  
    2 Timer fixed bin (15),  
    2 Sync fixed bin (15),  
    2 State fixed bin (15),  
    2 Kind fixed bin (15),  
    2 ExpirationTime like AbsoluteTime;
```

Interval Timer Information Record

Described in Chapter 5.

```
dcl 1 IntTimerInfo based,  
    2 Timer fixed bin (15),  
    2 Sync fixed bin (15),  
    2 State fixed bin (15),  
    2 Kind fixed bin (15),  
    2 RemainingTime fixed bin (31);
```

Repetitive Timer Information Record

Described in Chapter 5.

```
dcl 1 RepTimerInfo based,  
  2 Timer fixed bin (15),  
  2 Sync fixed bin (15),  
  2 State fixed bin (15),  
  2 Kind fixed bin (15),  
  2 RepState,  
    3 RemainingTime fixed bin (31),  
    3 SucceedingIntervals fixed bin (31);
```

Attribute Identity Block

Described in Chapter 14.

```
dcl 1 AttributeIdentityBlock,  
  2 Version fixed bin(15),  
  2 NodeName char(16) var,  
  2 UserID char(32) var,  
  2 IAMInitiator bit(1) aligned,  
  2 TargetServerName char(12) var;
```

Initiator Authentication Block

Described in Chapter 8.

```
dcl 1 InitiatorAuthBlock,  
  2 Version fixed bin(15),  
  2 NodeName char(16) var,  
  2 ProjectID char(32) var,  
  2 FullID,  
    3 UserID char(32) var,  
    3 ACLGroupsCount fixed bin(15),  
    3 ACLGroups (32) char(32) var;
```


Low Level Name

Described in Chapter 7.

```
dcl 1 LowLevelName,  
  2 Version fixed bin(15),  
  2 NodeName char(16) var,  
  2 ServerName char(12) var,  
  2 ForClientUse fixed bin(31),  
  2 Reserved (13) fixed bin(15);
```

Message Specifier

Described in Chapter 10.

```
dcl 1 MessageSpecifier,  
  2 Version fixed bin(15),  
  2 Control,  
    3 SuppliedLength fixed bin(15),  
    3 ReturnedLength fixed bin(15),  
    3 BufferLocation ptr,  
  2 Data,  
    3 BufferLength fixed bin(15),  
    3 BufferLocation ptr;
```

Session Configuration Block

Described in Chapter 9.

```
dcl 1 SessionConfigurationBlock,
    2 Version fixed bin(15),                /* local */
    2 SessionServices,
        3 NormalService bit(1),             /* global */
        3 ExpeditedService bit(1),          /* global */
        3 SyncsToBeUsed,
            4 ReadyToSend bit(1),            /* local */
            4 ReadyToSendExpedited bit(1),   /* local */
            4 ReadyToReceive bit(1),         /* local */
            4 ReadyToReceiveExpedited bit(1), /* local */
            4 BufferAvailable bit(1),         /* local */
            4 SessionResponsePending bit(1), /* local */
            4 ExceptionPending bit(1),       /* local */
        3 Reserved bit(7),                  /* local */
    2 QueueLengths,
        3 NormalSend fixed bin(15),         /* global */
        3 NormalReceive fixed bin(15),      /* global */
        3 ExpeditedSend fixed bin(15),      /* global */
        3 ExpeditedReceive fixed bin(15),   /* global */
    2 QueueThresholds,
        3 NormalSend fixed bin(15),         /* local */
        3 NormalReceive fixed bin(15),      /* local */
        3 ExpeditedSend fixed bin(15),      /* local */
        3 ExpeditedReceive fixed bin(15),   /* local */
    2 MaxControlLength fixed bin(15),       /* global */
    2 MaxDataLength fixed bin(15),          /* global */
    2 MaxExpeditedLength fixed bin(15),     /* global */
    2 ExistingSessionID fixed bin(15),      /* local */
    2 MessageArea,
        3 BlockSize fixed bin(15),          /* global */
        3 NumberOfBlocks fixed bin(15),     /* global */
        3 Reserved fixed bin(15);          /* global */
```

Session Statistics Block

Described in Chapter 14.

```
dcl 1 SessionStatisticsBlock,  
  2 Version fixed bin(15),  
  2 NormalMessages,  
    3 Sent fixed bin(31),  
    3 Received fixed bin(31),  
    3 FailedSends fixed bin(31),  
    3 FailedReceives fixed bin(31),  
  2 ExpeditedMessages,  
    3 Sent fixed bin(31),  
    3 Received fixed bin(31),  
    3 FailedSends fixed bin(31),  
    3 FailedReceives fixed bin(31),  
  2 Exceptions,  
    3 Count fixed bin(31),  
  2 Allocations,  
    3 Failed fixed bin(31),  
    3 AverageBufferSize fixed bin(15),  
  2 MessageAreaInfo,  
    3 CurrentAreaUsage fixed bin(15),  
    3 MaxAreaUsage fixed bin(15);
```

Session Status Block

Described in Chapter 14.

```
dcl 1 SessionStatusBlock,  
  2 Version fixed bin(15),  
  2 Phase fixed bin(15),  
  2 ExceptionsToBeCleared fixed bin(15),  
  2 MessageAreaUsers fixed bin(15),  
  2 CurrentQueueStatus,  
    3 NormalSend fixed bin(15),  
    3 NormalReceive fixed bin(15),  
    3 ExpeditedSend fixed bin(15),  
    3 ExpeditedReceive fixed bin(15);
```

Session Synchronizers List

Described in Chapter 9.

```
dcl 1 SessionSyncList,  
  2 Version fixed bin(15),  
  2 ReadyToSend fixed bin(15),  
  2 ReadyToSendExpedited fixed bin(15),  
  2 ReadyToReceive fixed bin(15),  
  2 ReadyToReceiveExpedited fixed bin(15),  
  2 BufferAvailable fixed bin(15),  
  2 SessionResponsePending fixed bin(15),  
  2 ExceptionPending fixed bin(15);
```

Target Authentication Block

Described in Chapter 8.

```
dcl 1 TargetAuthBlock,  
  2 Version fixed bin(15),  
  2 NodeName char(16) var,  
  2 UserID char(32) var;
```

Data Type Equivalents

To call a subroutine from a program written in any Prime language, you must declare the subroutine and its parameters in the calling program. Therefore, you must translate the PL/I data types expected by the subroutine into the equivalent data types in the language of the calling program.

The table that follows shows the equivalent data types for the Prime languages BASIC/VM, C, COBOL 74, FORTRAN IV, FORTRAN 77, Pascal, and PL/I. The leftmost column lists the generic storage unit, which is measured in bits, bytes, or halfwords for each data type. Each storage unit matches the data types listed to the right on the same row. The table does not include an equivalent data type for each generic unit in all languages. However, with knowledge of the corresponding machine representation, you can often determine a suitable workaround. For instance, to see if you can use a left-aligned bit in COBOL 74, you could write a program to test the sign of the 16-bit field declared as COMP. In addition, if a subroutine parameter consists of a structure with elements declared as BIT(n), it can be declared as an integer in the calling program. Read the appropriate language chapter in the *Subroutines Reference I: Using Subroutines* before using any of the equivalents shown in the table.

Note

The term PL/I refers both to full PL/I and to PL/I Subset G (PL/I-G).

Table F-1
Data Type Equivalents

Generic Unit	BASIC/VM SUB FORTRAN	C	COBOL 74	FORTRAN IV	FORTRAN 77	Pascal	PL/I
16-bit integer	INT	short enum	COMP PIC S9(1)- PIC S9(4)	INTEGER INTEGER*2 LOGICAL	INTEGER*2 LOGICAL*2	INTEGER Enumerated	FIXED BIN FIXED BIN(15)
32-bit integer	INT*4	int long	COMP PIC S9(5)- PIC S9(9)	INTEGER*4	INTEGER INTEGER*4 LOGICAL LOGICAL*4	LONGINTEGER	FIXED BIN(31)
64-bit integer			COMP PIC S9(10)- PIC S9(18)				
32-bit float single precision	REAL	float	COMP-1	REAL REAL*4	REAL REAL*4	REAL	FLOAT BIN FLOAT BIN(23)
64-bit float double precision	REAL*8	double	COMP-2	REAL*8	REAL*8	LONGREAL	FLOAT BIN(47)
128-bit float quad precision					REAL*16		
1 bit		short					BIT BIT(1)
1 left-aligned bit		short				BOOLEAN	BIT(1) ALIGNED

Table F-1
Data Type Equivalents – Continued

Generic Unit	BASIC/VM SUB FORTRAN	C	COBOL 74	FORTRAN IV	FORTRAN 77	Pascal	PL/I
Bit string		unsigned int				SET	BIT(n)
Fixed-length character string	INT	char NAME[n] char NAME	DISPLAY PIC A(n) PIC X(n) FILLER		CHARACTER *n	CHAR PACKED ARRAY[1..n] OF CHAR	CHAR(n)
Fixed-length digit string			DISPLAY PIC 9(n)				PICTURE
Fixed-length digit string, 2 digits per byte			COMP-3				FIXED DECIMAL
Varying-length character string		struct {short LENGTH; char DATA[n]; } CVAR				STRING[n]	CHAR(n) VARYING
32-bit pointer		Pointer (32IX-mode)		LOC()	LOC()		POINTER OPTIONS (SHORT)
48-bit pointer		Pointer (64V-mode)				Pointer	POINTER

Notes

For a discussion of possible workarounds for some of the empty boxes in this table as well as a description of generic units for PMA, refer to the appropriate language chapter in the *Subroutines Reference Guide, Volume I*.

Index of Subroutines by Function

This index lists subroutines grouped by the general functions that they perform. See the Index of Subroutines by Name to find a particular subroutine's volume, chapter, and page number.

Access Category

Add an object's name to an access category.	AC\$CAT
Convert an object from ACL protection to password protection.	AC\$RVT
Delete an access category.	CAT\$DL
Determine whether an object is accessible for a given action.	CALAC\$
Determine whether an object is ACL-protected.	ISACL\$
Make an object's ACL identical to that of another object.	AC\$LIK
Modify an existing ACL on an object.	AC\$CHG
Obtain the contents of an object's ACL.	AC\$LST
Obtain the contents of an object's priority ACL.	PA\$LST
Obtain the passwords of a subdirectory of the current directory.	GPAS\$\$
Obtain the user ID and the groups to which it belongs.	GETID\$
Remove an object's priority access.	PA\$DEL
Set an object's ACL to that of its parent directory.	AC\$DFT
Set the owner and nonowner passwords on an object.	SPAS\$\$
Set priority access on an object.	PA\$SET
Set a specific ACL on an object.	AC\$SET

Arrays

Get a character from an array.

GCHAR

Store a character into an array location.

SCHAR

Attach Points

Set the attach point to a directory specified by pathname.

AT\$

Set the attach point to a specified top-level directory and partition.

AT\$ABS

Set the attach point to a specified top-level directory on any partition.

AT\$ANY

Set the attach point to the home directory.

AT\$HOM

Set the attach point to a specified top-level directory on a partition identified by logical disk number.

AT\$LDEV

Set the attach point to the login directory.

AT\$OR

Set the attach point to a directory subordinate to the current directory.

AT\$REL

Set the attach point to a specified user directory and, optionally, make it the home directory.

ATCH\$\$

Binary Search

Perform binary search in ordered table.

BIN\$SR

Buffer Output

Provide free-format output to a buffer.

IOA\$RS

Command Environment

Invoke a command from a running program.	CP\$
Parse command arguments according to a character string "picture" of the command line.	CL\$PIX
Retrieve the value of a CPL local variable.	LV\$GET
Retrieve the value of a global variable.	GV\$GET
Return caller's maximum command environment breadth.	CE\$BRD
Return caller's maximum command environment depth.	CE\$DPT
Return a list of commands valid at mini-command level.	LIST\$CMD
Return breadth of caller's current command environment.	RD\$CE_DP
Set the value of a CPL local variable.	LV\$SET
Set the value of a global variable.	GV\$SET

Command Level

Call a new command level.	COMLV\$
Call a new command level after an error.	CMLV\$E
Initialize the command environment.	ICE\$
Record command error status.	SETRC\$
Return to PRIMOS.	EXIT
Return serialization data.	KLM\$IF
Signal an error in a subsystem.	SS\$ERR

Condition Mechanism

Continue scan for on-units.	CNSIG\$
Convert FORTRAN statement label to PL/I format.	MKLB\$F
Create an on-unit (for FTN users).	MKON\$F
Create an on-unit (for any language except FTN).	MKON\$P

Create an on-unit (for PMA and PL/I users).	MKONU\$
Perform a nonlocal GOTO.	PL1\$NL
Revert an on-unit (for FTN users).	RVON\$F
Revert an on-unit (for any language except FTN).	RVONU\$
Signal a condition (for FTN users).	SGNL\$F
Signal a condition (for any language except FTN).	SIGNL\$

Controllers, Asynchronous, Multi-line

Assign AMLC line.	ASNLN\$
Communicate with AMLC driver.	T\$AMLC
Communicate with SMLC driver.	T\$SLC0

Data Conversion

Convert ASCII number to binary.	CNVA\$A
Convert binary number to ASCII.	CNVB\$A
Convert the DATMOD field (as returned by RDEN\$\$) in format DAY, MON DD YYYY.	FDAT\$A
Convert the DATMOD field (as returned by RDEN\$\$) in format DAY, DD MON YYYY.	FEDT\$A
Convert a string from lowercase to uppercase or uppercase to lowercase.	CASE\$A
Convert the TIMMOD field (as returned by RDEN\$A).	FTIM\$A
Make a number printable, if possible.	ENCD\$A

Date Formats

Convert ASCII date to binary format.	CV\$DTB
Convert binary date to quadseconds.	CV\$DQS
Convert binary date to ISO format.	CV\$FDA
Convert binary date to "visual" format.	CV\$FDV
Convert quadsecond date to binary format.	CV\$QSD

Devices, Assigning or Attaching

Attach specified devices.	ATTDEV
Free a logical file unit number.	IOCS\$F
Provide or set aside available logical file unit.	IOCS\$G

Disk I/O

Read ASCII from disk.	I\$AD07
Read binary from disk.	I\$BD07
Register disk format with driver.	DKGEO\$
Write ASCII to disk (fixed-length records).	O\$AD08
Write binary to disk.	O\$BD07

Drivers, Device-independent

Open PRIMOS file and perform other nondata transfer functions. (Primarily for IOCS applications.)	CONTRL
Read ASCII data.	RDASC
Read binary data.	RDBIN
Write ASCII data.	WRASC
Write binary data.	WRBIN

Encryption of Login Password

Encrypt login validation passwords.

ENCRYPT\$

EPFs

Allocating and Deallocating Space For EPFs

Allocate space for EPF function return information.

ALC\$RA

Allocate space and set value of EPF function return information.

ALS\$RA

Deallocate space for EPF function return information.

FRE\$RA

Management of EPFs

Combine functions of EPF\$ALLC, EPF\$MAP, EPF\$INIT, and EPF\$INVK.

EPF\$RUN

Deactivate the most recent invocation of a specified EPF.

EPF\$DEL

Initiate the execution of a program EPF.

EPF\$INVK

Map the procedure images of an EPF file into virtual memory.

EPF\$MAP

Modify user's search rules to allow dynamic linking to a library EPF.

LN\$SET

Perform the linkage allocation phase for an EPF.

EPF\$ALLC

Perform the linkage initialization phase for an EPF.

EPF\$INIT

Remove an EPF from a user's address space.

REMEPF\$

Replace one EPF runfile with another.

RPL\$

Return the state of the command processing flags in an EPF.

EPF\$CPF

Information from In-Memory User Profile

Return highest segment number.

TL\$SGS

Return maximum number of dynamic segments.

DY\$SGS

Return maximum number of static segments.

ST\$SGS

Error Handling, I/O

Display I/O error message on user terminal.	PRERR
Obtain contents of ERRVEC.	GETERR
Set ERRVEC and perform a return or display ERRVEC message before returning control to system.	ERRSET

Event Synchronizers and Event Groups

Creating, Using, and Destroying Event Synchronizers

Create an event synchronizer.	SYN\$CREA
Destroy an event synchronizer.	SYN\$DEST
Perform a timed wait on an event synchronizer.	SYN\$TMWT
Post a notice on an event synchronizer.	SYN\$POST
Retrieve a notice from an event synchronizer.	SYN\$RTRV
Wait on an event synchronizer.	SYN\$WAIT

Creating, Using, and Destroying Event Groups

Cause a process to wait on an event group.	SYN\$GWT
Cause a process to perform a timed wait on an event group.	SYN\$GTWT
Create an event group.	SYN\$GCRE
Destroy an event group.	SYN\$GDST
Move an event synchronizer into an event group.	SYN\$MVTO
Remove an event synchronizer from an event group.	SYN\$REMV
Retrieve a notice from an event group.	SYN\$GRTR

Getting Information about Synchronizers and Groups

Indicate whether synchronizer is in group; and if it is, return the group number, priority level, and For Client Use field.	SYN\$INFO
List the synchronizers in group and total number.	SYN\$LSIG
List the synchronizers in server and total number.	SYN\$LIST
List the groups in server and total number.	SYN\$GLST
Return number of notices or waiting processes.	SYN\$CHCK
Return number of notices on a group at one or all priority levels; if all levels, return number of waiting processes.	SYN\$GCHK

Executable Images

Restore an R-mode executable image.	REST\$\$
Restore and resumes an R-mode executable image.	RESU\$\$
Save an R-mode executable image.	SAVE\$\$

EXIT\$ Condition

Disable signalling of EXIT\$ condition.	EX\$CLR
Enable signalling of EXIT\$ condition.	EX\$SET
Return state of EXIT\$ signalling.	EX\$RD

File System Objects

Append a specified suffix to a pathname.	AP\$FX\$
Change the open mode of an open file.	CH\$MOD
Change the name of an object in the current directory.	CNAM\$\$
Check for file existence.	EXST\$A
Check for file open.	UNIT\$A

Close a file.	CLOS\$A
Close a file by name and return a bit string indicating closed units.	CL\$FNR
Close a file system object by pathname.	CLO\$FN
Close a file system object by file unit number.	CLO\$FU
Create a new directory.	DIR\$CR
Create a new password directory.	CREPW\$
Create a new subdirectory in the current directory.	CREA\$\$
Delete a file.	DELE\$A
Delete a file identified by a pathname.	FIL\$DL
Delete a segment directory entry.	SGD\$DL
Determine whether an open file system object is local or remote.	ISREM\$
Determine if a segment directory entry exists.	SGD\$EX
Extend or truncate a CAM file.	CF\$EXT
Force PRIMOS to write modified records to disk.	FORCEW
Generate a filename based on another name.	EQUAL\$
List the disks a given user is using.	LUDSK\$
Open supplied name.	OPEN\$A
Open supplied name with verification and delay.	OPNV\$A
Open a segment directory entry.	SGD\$OP
Open, close, delete, change access, or verify the existence of an object.	SRCH\$\$
Open a scratch file with unique name.	TEMP\$A
Open a file anywhere in the PRIMOS file structure.	TSRC\$\$
Position file.	POSN\$A
Position to end-of-file.	GEND\$A
Position in or read from a directory.	RDEN\$\$
Position in, read an entry in, or modify the size of a segment directory.	SGDR\$\$
Read a line of characters from an ASCII disk file.	RDLIN\$
Read sequentially the entries of a directory open on a file unit.	DIR\$RD
Read name and open.	OPNP\$A
Read name and open with verification and delay.	OPVP\$A

Read, write, position, or truncate a file.	PRWF\$\$
Retrieve a CAM file's extent map from disk.	CF\$REM
Return directory quota and disk record usage information.	Q\$READ
Return entries meeting caller-specified selection criteria in a directory open on a file unit.	DIR\$SE
Return the contents of a named entry in a directory open on a file unit.	ENT\$RD
Return a file system object's entryname and parent directory pathname.	EXTR\$A
Return information about a specified file unit.	FINFO\$
Return information on the system's list of logical disks.	LDISK\$
Return the pathname of a specified unit, attach point, or segment.	GPATH\$
Return a logical value indicating whether a specified partition supports ACL protection and quotas.	PAR\$RV
Return position of file.	RPOS\$A
Return the size of a file system entry.	SIZE\$
Return the minimum and maximum file unit numbers currently in use by this user.	UNIT\$S
Return a logical value indicating whether a wildcard name was matched.	WILD\$
Rewind file.	RWND\$A
Scan the file system structure.	TSNC\$A
Search for specified types of entries in a directory open on a file unit.	DIR\$LS
Search for a file with a list of possible suffixes.	SRSFX\$
Set a CAM file's extent length value.	CF\$SME
Set a quota on a subdirectory in the current directory.	Q\$SET
Set or modify an object's attributes in its directory entry.	SATR\$S
Truncate file.	TRNC\$A
Verify a supplied string as a valid filename.	FNCHK\$
Verify a supplied string as a valid pathname.	TNCHK\$
Write a line of characters to a file in compressed ASCII format.	WTLIN\$

ISC

Establish an ISC Session

Initiator gets the session request response.	IS\$GRS
Initiator requests the session.	IS\$RS
Recipient gets the session request.	IS\$GRQ
Recipient accepts the session.	IS\$AS

ISC Message Exchange

Allocate a buffer for a message data part.	IS\$AB
Free an allocated data part buffer.	IS\$FB
Receive a message.	IS\$RM
Send a message.	IS\$SM

Monitor ISC Message Exchange Session

Get session attributes.	IS\$GSA
Get session status.	IS\$GSS
Get sessions owned by your server.	IS\$GSO
Get statistics about a session.	IS\$STA

Terminate an ISC Session and Respond to an Exception

Clear an exception.	IS\$CE
Get an exception.	IS\$GE
Terminate the caller's side of a session.	IS\$TS

Keyboard or ASR Reader

Input ASCII from terminal or ASR reader.	I\$AA01
Perform same function as I\$AA01 but also allow input from a cominput file.	I\$AA12

Matrix Operations

Generate combinations.	COMB
Generate permutations.	PERM

The following groups contain subroutines for single-precision, double-precision, integer, and complex operations, respectively. (* indicates that a subroutine is not available.)

Calculate adjoint matrix.	MADJ, DMADJ, IMADJ, CMADJ
Calculate determinant.	MDET, DMDET, IMDET, CMDET
Calculate inverted matrix.	MINV, DMINV, *, CMINV
Calculate signed cofactor.	MCOF, DMCOF, IMCOF, CMCOF
Calculate transpose matrix.	MTRN, DMTRN, IMTRN, CMTRN
Multiply matrix by a scalar.	MSCL, DMSCL, IMSCL, CMSCL
Perform matrix addition.	MADD, DMADD, IMADD, CMADD
Perform matrix multiplication.	MMLT, DMMLT, IMMLT, CMMLT
Perform matrix subtraction.	MSUB, DMSUB, IMSUB, CMSUB
Set matrix to constant matrix.	MCON, DMCON, IMCON, CMCON
Set matrix to identity matrix.	MIDN, DMIDN, IMIDN, CMIDN
Solve a system of linear equations.	LINEQ, DLINEQ, *, CLINEQ

Memory

Allocate memory on the current stack.	ALOC\$\$
Allocate process-class dynamic memory.	STR\$AP
Allocate subsystem-class dynamic memory.	STR\$AS
Allocate user-class dynamic memory.	STR\$AU
Allocate user-class dynamic memory.	STR\$AL
Free process-class dynamic memory.	STR\$FP
Free subsystem-class dynamic memory.	STR\$FS
Free user-class dynamic memory.	STR\$FR
Free user-class dynamic memory.	STR\$FU
Make the last page of a segment available.	MM\$MLP
Make the last page of a segment unavailable.	MM\$MLP
Move a block of memory.	MOVEW\$

Message Facility

Receive a deferred message.	RMSGD\$
Return the receiving state of a user.	MSG\$ST
Send an interuser message.	SMSG\$
Set the receiving state for messages.	MGSET\$

Numeric Conversions

Convert string (decimal) to 16-bit integer.	CH\$FX1
Convert string (decimal) to 32-bit integer.	CH\$FX2
Convert string (hexadecimal) to 32-bit integer.	CH\$HX2
Convert string (octal) to 32-bit integer.	CH\$OC2

Paper Tape

Control functions for paper tape.	C\$P02
Input ASCII from the high-speed paper-tape reader.	I\$AP02
Input one character from the high-speed paper-tape reader to Register A.	P1IB
Input one character from paper tape, set high-order bit, ignore line feeds, send a line feed when carriage return is read.	P1IN
Output binary data to the high-speed paper-tape punch.	O\$BP02
Output one character to the high-speed paper-tape punch from Register A.	P1OB
Output one character to the high-speed paper-tape punch.	P1OU

Parsing

Parse character string into tokens.	GT\$PAR
Parse a PRIMOS command line.	CMDL\$A

Peripheral Devices

Line Printers

Access a spooler queue.	SPOOL\$
Centronics LP.	O\$AL04
Move data to LPC line printer.	T\$LMPC
Parallel interface to line printer (MPC).	O\$AL06
Place file in spool queue and perform SPOOLER command functions.	SP\$REQ
Versatec printer.	O\$AL14

Printer/Plotter

Versatec.	O\$AL14
Versatec.	T\$VG

Card Reader/Punch

Input from parallel card reader.	I\$AC03
Input from serial card reader.	I\$AC09
Input from MPC card reader.	T\$CMPC
Parallel interface to card punch.	O\$AC03
Parallel interface to card punch and print on card.	O\$AC15
Raw data mover.	T\$PMPC
Read and print card from parallel interface reader.	I\$AC15

Magnetic Tape

Raw data mover.	T\$MT
Read EBCDIC from 9-track.	I\$AM13
Write EBCDIC to 9-track.	O\$AM13

Phantom Processes

Read logout notification information.	LON\$R
Start a phantom process.	PHNTM\$
Switch logout notification on or off.	LON\$CN

Process Suspension

Suspend a process for a specified interval.	SLEEP\$
Suspend a process (interruptible).	SLEP\$I

Query User

Ask question and obtain a YES or NO answer.	YSNO\$A
Prompt and read a name.	RNAM\$A
Prompt and read a number (binary, decimal, octal, or hexadecimal).	RNUM\$A

Randomizing

Generate random number and update "seed," based upon a 32-bit word size and using the Linear Congruential Method.	RAND\$A
Initialize random number generator "seed."	RNDI\$A

Search Rules

Add a rule to the beginning of a search list or before a specified rule.	SR\$ADDB
Add a rule to the end of a search list or after a specified rule.	SR\$ADDE
Create a search list.	SR\$CREAT
Delete a search list.	SR\$DEL
Determine if a search rule exists.	SR\$EXSTR
Disable an optional search rule. Used to disable rules that have been enabled using SR\$ENABL.	SR\$ABSDS
Disable an optional search rule. Used to disable rules that have been enabled using SR\$ENABL.	SR\$DSABL
Enable an optional search rule. Enabled rules can be disabled using SR\$DSABL or SR\$ABSDS.	SR\$ENABL
Free list structure space allocated by SR\$LIST or SR\$READ.	SR\$FR_LS
Initialize all search lists to system defaults.	SR\$INIT
Locate a file using a search list and open the file. Create a file if the file sought does not exist.	OPSR\$
Locate a file using a search list and a list of suffixes. Open the located file, or create a file if the file sought does not exist.	OPSR\$S
Read the next rule from a search list.	SR\$NEXTR

Read all of the rules in a search list.	SR\$READ
Remove a search rule from a search list.	SR\$REM
Return the names of all defined search lists.	SR\$LIST
Set the locator pointer for a search rule.	SR\$SETL
Set a search list using a user-defined search rules file.	SR\$SSR

Semaphores

Drain a semaphore.	SEM\$DR
Notify a semaphore.	SEM\$NF
Open a set of named semaphores.	SEM\$OP
Open a set of named semaphores.	SEM\$OU
Periodically notify a semaphore.	SEM\$TN
Release (close) a named semaphore.	SEM\$CL
Return number of processes waiting on a semaphore.	SEM\$TS
Wait on a semaphore.	SEM\$WT
Wait on a specified named semaphore, with timeout.	SEM\$TW

Server Names

Catalog a server's Low Level Name.	ISN\$C
Get the process numbers of all processes associated with the server name.	SRS\$GP
Get the server name of a process.	SRS\$GN
List the server names on your system.	SRS\$LN
Look up a server's Low Level Name.	ISN\$L
Recatalog a server's Low Level Name.	ISN\$RC
Uncatalog a server's Low Level Name.	ISN\$UC

Sorting

Binary search or build binary table.	BNSRCH
Bubble sort.	BUBBLE
Close merged input files.	MRG3\$\$
Close all sort units.	CLNU\$\$
Diminishing increment sort.	SHELL
Get input records.	RLSE\$\$
Get sorted records.	RTRN\$\$
Heap sort.	HEAP
Insertion sort.	INSERT
Merge sorted files.	MRG1\$\$
Partition exchange sort.	QUICK
Prepare sort table and buffers.	SETU\$\$
Radix exchange sort.	RADXEX
Return next merged record to sort.	MRG2\$\$
Sort one file on ASCII key(s).	SUBSRT
Sort (multiple key types) or merge sorted files.	ASCS\$\$
Sort one or several input files.	SRTF\$\$
Sort tables prepared by SETU\$\$.	CMBN\$\$

Strings

Compare two strings for equality.	CSTR\$A
Compare two substrings for equality.	CSUB\$A
Compare two character strings.	NAMEQ\$
Convert UID\$BT output into character string.	UID\$CH
Determine the operational length of a string.	NLEN\$A
Determine string type.	TYPE\$A
Fill a string with a character.	FILL\$A
Fill a substring with a given character.	FSUB\$A

Get a character from a packed string.	GCHR\$A
Left-justify, right-justify, or center a string within a field.	JSTR\$A
Locate one string within another.	LSTR\$A
Locate one substring within another.	LSUB\$A
Move a character between packed strings.	MCHR\$A
Move one string to another.	MSTR\$A
Move one substring to another.	MSUB\$A
Return unique bit string.	UID\$BT
Rotate string left or right.	RSTR\$A
Rotate substring left or right.	RSUB\$A
Shift string left or right.	SSTR\$A
Shift substring left or right.	SSUB\$A
Test for pathname.	TREE\$A

System Information

General System Information

Check validity of system name passed to it.	SNCHK\$
Determine access to a segment.	RSEGAC\$
Determine if routine is dynamically accessible.	CKDYN\$
Indicate if Login-over-login permitted.	LOV\$SW
Return cold-start setting of ABBREV switch.	AB\$SW\$
Return current date and time.	DATE\$
Return current PRIMOS system name.	GSNAM\$
Return information on the system's list of logical disks.	LDISK\$
Return model number of Prime computer.	CPUID\$
Return operating system revision number.	PRI\$RV
Return PRIMOS II information.	GINFO
Return text representation of error code.	ERTXT\$
Return text representation of error code for specified PRIMOS subsystem.	ER\$TEXT
Return user number and count of users.	USER\$

System Time Information

Return CPU time since login.	CTIM\$A
Return disk time since login.	DTIM\$A
Return time of day.	TIME\$A
Return today's date, American style.	DATE\$A
Return today's date as day of year (Julian date).	DOFY\$A
Return today's date, European (military) style.	EDAT\$A

System Status and Metering Information

Return data about a disk partition.	DS\$AVL
Return data about file units.	DS\$UNI
Return data about a process's environment.	DS\$ENV
Return a variety of metering information.	G\$METR

Timers

Cancel a timer.	TMR\$CANL
Create a timer.	TMR\$CREA
Destroy a timer.	TMR\$DEST
List the identifiers of the timers within a server.	TMR\$LIST
Return the timer type and information.	TMR\$GTMR
Set an absolute timer.	TMR\$\$ABS
Set an interval timer.	TMR\$\$INT
Set and read various timers.	LIMIT\$
Set a repetitive timer.	TMR\$\$REP

User Information

Change login validation password.	CHG\$PW
Check that a process has a given amount of timeslice left.	ASSUR\$
Convert local time to Universal Time.	TMR\$LOCALCONVERT
Convert Universal Time to local time.	TMR\$UNIVCONVERT
Determine whether a forced logout is in progress.	IN\$LO
Display PRIMOS command prompt.	READY\$
Display standard message showing times used.	TI\$MSG
Expand a line using abbreviations preprocessor.	COM\$AB
Generate a new login validation password.	GEN\$PW
List the disks a given user is using.	LUDSK\$
List users with same name as caller.	UNO\$GT
Log out a user.	LOGO\$S
Return amount of CPU time used since login.	PTIME\$
Return current system time.	TMR\$GTIM
Return a list of devices that a user can access.	LUDEV\$
Return permanent time information.	TMR\$GINF
Return timing information and user identification.	TIMDAT
Return the user's project identifier.	PRJID\$
Return user number of initiating process.	SID\$GT
Return user type of current process.	UTYPE\$
Test whether current user is supervisor.	SUSR\$
Validate a name.	IDCHK\$
Validate syntax of a password.	PWCHK\$
Validate a name against composite identification.	VALID\$

User Terminal

Functions

Control functions for user terminal.	C\$A01
Get next character from terminal or command file.	C1IN
Get next character from command line until carriage return.	C1IN\$
Inhibit or enable CONTROL-P.	BREAK\$
Input decimal number.	TIDEC
Input an octal number.	TIOCT
Input a hexadecimal number.	TIHEX
Move characters from terminal or command file to memory.	CNIN\$
Output ASCII to the user terminal or ASR punch.	O\$AA01
Output count characters to the user terminal followed by a line feed and carriage return.	TNOU
Output count characters to the user terminal.	TNOUA
Output the 16-bit integer num to the terminal.	TOVFD\$
Output char to the user terminal. The data type must be a 16-bit integer in F77.	T1OU
Output a six-character signed decimal number.	TODEC
Output a six-character unsigned octal number.	TOOCT
Output a four-character unsigned hexadecimal number.	TOHEX
Output carriage return and line feed.	TONL
Read a line of text from the terminal or from a command file.	COMANL
Read or set erase and kill characters.	ERKL\$\$
Read one character from the user terminal into Register A.	T1IB
Read one character from the user terminal.	T1IN
Write one character from Register A to the user terminal.	T1OB

Input from User Terminal

Parse a command line.	RDTK\$\$
Read a character.	CIIN
Read a character.	CIIN\$
Read a character, suppressing echo.	CINE\$
Read a character (function).	T1IB
Read a character (procedure).	T1IN
Read a decimal number.	TIDEC
Read a hexadecimal number.	TIHEX
Read a line.	CL\$GET
Read a line into a PRIMOS buffer.	COMANL
Read an octal number.	TIOCT
Read a specified number of characters.	CNIN\$

Output to User Terminal

Print a standard error message from PRIMOS or a PRIMOS subsystem.	ER\$PRINT
Print a standard error message.	ERRPR\$
Provide free-format output.	IOA\$
Provide free-format output for error messages.	IOA\$ER
Write characters to terminal, followed by NEWLINE.	TNOU
Write characters to terminal.	TNOUA
Write a signed decimal number.	TODEC
Write a hexadecimal number.	TOHEX
Write a NEWLINE.	TONL
Write an octal number.	TOOCT
Write a decimal number without spaces.	TOVFD\$
Write one character from Register A.	T1OB
Write one character.	T1OU

Control Output to User Terminal

Check for unread terminal input characters.	TTY\$IN
Clear the terminal input and output buffers.	TTY\$RS
Control the way PRIMOS treats the user terminal.	DUPLX\$
Determine if there are pending quits.	QUIT\$
Inhibit or enable BREAK function.	BREAK\$
Read or set the erase and kill characters.	ERKL\$\$
Return information about command output settings.	CO\$GET
Switch input between the terminal and a file.	COMI\$\$
Switch output between the terminal and a file.	COMO\$\$

Index of Subroutines by Name

A\$xy series	FORTTRAN compiler addition functions.	I	B-7
AB\$SW\$	Return cold-start setting of ABBREV switch.	III	2-3
AC\$CAT	Add an object's name to an access category.	II	2-3
AC\$CHG	Modify an existing ACL on an object.	II	2-5
AC\$DFT	Set an object's ACL to that of its parent directory.	II	2-7
AC\$LIK	Set an object's ACL like that of another object.	II	2-9
AC\$LST	Obtain the contents of an object's ACL.	II	2-11
AC\$RVT	Convert an object from ACL protection to password protection.	II	2-13
AC\$SET	Set a specific ACL on an object.	II	2-15
ALC\$RA	Allocate space for EPF function return information.	III	4-16
ALOC\$\$	Allocate memory on the current stack.	III	4-3
ALS\$RA	Allocate space and set value of EPF function.	III	4-21
AP\$FX\$	Append a specified suffix to a pathname.	II	4-4
ASCS\$\$	Sort or merge sorted files (multiple file types and key types). (V-mode)	IV	17-12
ASCS\$\$	Sort or merge sorted files (multiple file types and key types). (R-mode)	IV	17-42
ASCSRT	Synonym for ASCS\$. See above.		
AS\$LIN	Return asynchronous line number.	IV	8-32
AS\$LST	Retrieve asynchronous line characteristics.	IV	8-26
AS\$NLN\$	Assign AMLC line.	IV	8-21
AS\$SET	Set asynchronous line characteristics.	IV	8-33
ASSUR\$	Check process has given amount of timeslice left.	III	2-22
AT\$	Set the attach point to a directory specified by pathname.	II	3-3
AT\$ABS	Set the attach point to a specified top-level directory and partition.	II	3-6
AT\$ANY	Set the attach point to a specified top-level directory on any partition.	II	3-8
AT\$HOM	Set the attach point to the home directory.	II	3-10

AT\$LDEV	Set the attach point by top-level directory and logical disk number.	II	3-11
AT\$OR	Set the attach point to the login directory.	II	3-13
AT\$REL	Set the attach point relative to the current directory.	II	3-15
ATCH\$\$	Set the attach point to a specified directory.	II	A-2
ATTDEV	Change a device assignment temporarily.	IV	3-6
BIN\$SR	Perform binary search in ordered table.	III	6-21
BNSRCH	Binary search.	IV	17-48
BREAK\$	Inhibit or enable BREAK function.	III	3-50
BUBBLE	Bubble sort.	IV	17-50
C\$xy series	FORTTRAN compiler conversion functions.	I	B-5
C\$A01	Control functions for user terminal.	IV	6-5
C\$M05	Control functions for 9-track tape.	IV	E-5
C\$M10	Control functions for 7-track tape.	IV	E-5
C\$M11	Control functions for 7-track tape (BCD).	IV	E-5
C\$M13	Control functions for 9-track tape (EBCDIC).	IV	E-5
C\$P02	Control functions for paper tape.	IV	6-12
C1IN	Read a character.	III	3-5
C1IN\$	Read a character.	III	3-7
C1NE\$	Read a character, suppressing echo.	III	3-9
CALAC\$	Determine whether an object is accessible for a given action.	II	2-17
CASE\$A	Convert between upper- and lowercase.	IV	14-2
CAT\$DL	Delete an access category.	II	2-19
CE\$BRD	Return caller's maximum command environment breadth.	II	6-3
CE\$DPT	Return caller's maximum command environment depth.	II	6-4
CF\$EXT	Extend or truncate a CAM file.	II	4-130
CF\$REM	Get a CAM file's extent map.	II	4-132
CF\$SME	Set a CAM file's extent length value.	II	4-135
CH\$FX1	Convert string (decimal) to 16-bit integer.	III	6-3
CH\$FX2	Convert string (decimal) to 32-bit integer.	III	6-5
CH\$HX2	Convert string (hexadecimal) to 32-bit integer.	III	6-7
CH\$MOD	Change the open mode of an open file.	II	4-6
CH\$OC2	Convert string (octal) to 32-bit integer.	III	6-9
CHG\$PW	Change login validation password.	III	2-23
CKDYN\$	Determine if routine is dynamically accessible.	III	2-4
CL\$FNR	Close a file by name and return a bit string indicating closed units.	II	4-7
CL\$GET	Read a line.	III	3-10

CL\$PIX	Parse command line according to a command line picture.	II	6-5
CLINEQ	Solve linear equations (complex).	IV	18-7
CLNU\$\$	Close all sort units after SRTF\$.	IV	17-29
CLO\$FN	Close a file system object by pathname.	II	4-9
CLO\$FU	Close a file system object by file unit number.	II	4-10
CLOSSA	Close a file.	IV	15-2
CMADD	Matrix addition (complex).	IV	18-9
CMADJ	Calculate adjoint matrix (complex).	IV	18-11
CMBN\$\$	Sort tables prepared by SETU\$.	IV	17-27
CMCOF	Calculate signed cofactor (complex).	IV	18-13
CMCON	Set constant matrix (complex).	IV	18-16
CMDDET	Calculate matrix determinant (complex).	IV	18-18
CMDL\$A	Parse a command line.	IV	16-2
CMIDN	Set matrix to identity matrix (complex).	IV	18-20
CMINV	Calculate signed cofactor (complex).	IV	18-22
CMLV\$E	Call new command level after an error.	III	5-5
CMMLT	Matrix multiplication (complex).	IV	8-24
CMSCL	Multiply matrix by scalar (complex).	IV	18-26
CMSUB	Matrix subtraction (complex).	IV	18-28
CMTRN	Calculate transpose matrix (complex).	IV	18-30
CNAM\$\$	Change the name of an object in the current directory.	II	4-11
CNIN\$	Read a specified number of characters.	III	3-13
CNSIG\$	Continue scan for on-units.	III	7-19
CNVA\$A	Convert ASCII number to binary.	IV	14-4
CNVB\$A	Convert binary number to ASCII.	IV	14-6
CO\$GET	Return information about command output settings.	III	3-52
COM\$AB	Expand a line using Abbreviations preprocessor.	III	2-25
COMANL	Read a line into a PRIMOS buffer.	III	3-15
COMB	Generate matrix combinations.	IV	18-5
COMI\$\$	Switch input between the terminal and a file.	III	3-53
COMLV\$	Call a new command level.	III	5-6
COMO\$\$	Switch output between the terminal and a file.	III	3-55
CONTRL	Perform device-independent control functions.	IV	4-11
CP\$	Invoke a command from a running program.	II	6-9
CPUID\$	Return model number of Prime computer.	III	2-5
CREA\$\$	Create a new subdirectory in the current directory.	II	A-5
CREPW\$	Create a new password directory.	II	A-7
CSTR\$A	Compare two strings for equality.	IV	10-2
CSUB\$A	Compare two substrings for equality.	IV	10-4
CTIM\$A	Return CPU time since login.	IV	12-2
CV\$DQS	Convert binary date to quadseconds.	III	6-12

CV\$DTB	Convert ASCII date to binary format.	III	6-13
CV\$FDA	Convert binary date to ISO format.	III	6-15
CV\$FDV	Convert binary date to "visual" format.	III	6-17
CV\$QSD	Convert quadsecond date to binary format.	III	6-19
D\$xy series	FORTTRAN compiler division functions.	I	B-7
D\$INIT	Initialize disk.	IV	5-13
DATE\$	Return current date and time.	III	2-8
DATE\$A	Return today's date, American style.	IV	12-3
DELE\$A	Delete a file.	IV	15-3
DIR\$CR	Create a new directory.	II	4-15
DIR\$LS	Search for specified types of entries in a directory open on a file unit.	II	4-17
DIR\$RD	Read sequentially the entries of a directory open on a file unit.	II	4-24
DIR\$SE	Return directory entries meeting caller-specified selection criteria.	II	4-29
DISPLY	Update sense light settings.	III	10-3
DKGEO\$	Register disk format with driver.	IV	5-18
DLINEQ	Solve a system of linear equations (double precision).	IV	18-7
DMADD	Matrix additions (double precision).	IV	18-9
DMADJ	Calculate adjoint matrix (double precision).	IV	18-11
DMCOF	Calculate signed cofactor (double precision).	IV	18-13
DMCON	Set matrix to constant matrix (double precision).	IV	18-16
DMDET	Calculate determinant (double precision).	IV	18-18
DMIDN	Set matrix to identity matrix (double precision).	IV	18-20
DMINV	Calculate inverted matrix (double precision).	IV	18-22
DMMLT	Matrix multiplication (double precision).	IV	18-24
DMSCL	Multiply matrix by a scalar (double precision).	IV	18-26
DMSUB	Matrix subtraction (double precision).	IV	18-28
DMTRN	Calculate transpose matrix (double precision).	IV	18-30
DOFY\$A	Return today's date as day of year (Julian).	IV	12-4
DS\$AVL	Return data about a disk partition.	III	2-51
DS\$ENV	Return data about a process's environment.	III	2-53
DS\$UNI	Return data about file units.	III	2-57
DTIM\$A	Return disk time since login.	IV	12-5
DUPLX\$	Control the way PRIMOS treats the user terminal.	III	3-57
DY\$SGS	Return maximum number of dynamic segments.	III	4-25
E\$xy series	FORTTRAN compiler exponentiation	I	B-8
ECL\$CC	Supervise editing of input from terminal or command file (callable from C).	III	3-28a

ECL\$CL	Interface to ECL\$CC (for non-C programs).	III	3-28d
EDAT\$A	Today's date, European (military) style.	IV	12-6
ENCD\$A	Make a number printable if possible.	IV	14-8
ENCRYPT\$	Encrypt login validation passwords.	III	6-24
ENT\$RD	Return the contents of a named entry in a directory open on a file unit.	II	4-37
EPF\$ALLC	Perform the linkage allocation phase for an EPF.	II	5-3
EPF\$CPF	Return the state of the command processing flags in an EPF.	II	5-5
EPF\$DEL	Deactivate the most recent invocation of a specified EPF.	II	5-7
EPF\$INIT	Perform the linkage initialization phase for an EPF.	II	5-9
EPF\$INVK	Initiate the execution of a program EPF.	II	5-11
EPF\$MAP	Map the procedure images of an EPF file into virtual memory.	II	5-15
EPF\$RUN	Combine functions of EPF\$ALLC, EPF\$MAP, EPF\$INIT, and EPF\$INVK.	II	5-19
EQUAL\$	Generate a filename based on another name.	II	4-39
ERKL\$\$	Read or set the erase and kill characters.	III	3-60
ER\$PRINT	Print error messages on terminal.	III	3-31
ERRPR\$	Print a standard error message.	III	10-3a
ERRSET	Set ERRVEC (a system error vector).	III	10-4
ER\$TEXT	Return error message to a variable.	III	2-8a
ERTXT\$	Return text associated with error code.	III	10-5b
EX\$CLR	Disable signalling of EXIT\$ condition.	III	7-35
EX\$RD	Return state of EXIT\$ signalling.	III	7-36
EX\$SET	Enable signalling of EXIT\$ condition.	III	7-37
EXIT	Return to PRIMOS.	III	5-7
EXST\$A	Check for file existence.	IV	15-4
EXTR\$A	Return an object's entryname and parent directory pathname.	II	4-41
F\$xyyy series	FORTTRAN compiler floating-point functions.	I	B-8
FDAT\$A	Convert the DATMOD field returned by RDEN\$\$ to DAY MON DD YYYY.	IV	14-10
FEDT\$A	Convert the DATMOD field returned by RDEN\$\$ to DAY DD MON YYYY.	IV	14-12
FIL\$DL	Delete a file identified by a pathname.	II	4-43
FILL\$A	Fill a string with a character.	IV	10-6
FINFO\$	Return information about a specified file unit.	II	4-45
FNCHK\$	Verify a supplied string as a valid filename.	II	4-49

FORCEW	Force PRIMOS to write modified records to disk.	II	4-51
FRE\$RA	Deallocate space for EPF function return information.	III	4-23
FSUB\$A	Fill a substring with a given character.	IV	10-7
FTIM\$A	Convert the TIMMOD field returned by REDN\$\$.	IV	14-14
G\$METR	Return system metering information.	III	2-63
GCHAR	Get a character from an array.	III	6-25
GCHR\$A	Get a character from a packed string.	IV	10-9
GEND\$A	Position to end of file.	IV	15-5
GEN\$PW	Generate a login validation password.	III	2-26a
GETERR	Return ERRVEC contents.	III	10-6
GETID\$	Obtain the user ID and the groups to which it belongs.	II	2-21
GINFO	Return PRIMOS II information.	III	2-10
GPAS\$\$	Obtain the passwords of a subdirectory of the current directory.	II	2-23
GPATH\$	Return the pathname of a specified unit, attach point, or segment.	II	4-53
GSNAM\$	Return current PRIMOS system name.	III	2-12
GT\$PAR	Parse character string into tokens.	III	6-27
GTROB\$	Find out whether current attach point is on a robust partition.	II	3-18
GV\$GET	Retrieve the value of a global variable.	II	6-12
GV\$SET	Set the value of a global variable.	II	6-14
H\$xy series	FORTTRAN compiler complex number storage.	I	B-5
HEAP	Heap sort.	IV	17-51
I\$AA01	Read ASCII from terminal.	IV	6-8
I\$AA12	Read ASCII from terminal or input stream by REDN\$\$.	IV	6-10
I\$AC03	Input from parallel card reader.	IV	7-22
I\$AC09	Input from serial card reader.	IV	7-24
I\$AC15	Read and print card from parallel card reader.	IV	7-26
I\$AD07	Read ASCII from disk.	IV	5-4
I\$AM05	Read ASCII from 9-track tape.	IV	E-7
I\$AM10	Read ASCII from 7-track tape.	IV	E-7
I\$AM11	Read BCD from 7-track tape.	IV	E-7
I\$AM13	Read EBCDIC from 9-track tape.	IV	E-7
I\$AP02	Read paper tape (ASCII).	IV	6-13

I\$BD07	Read binary from disk.	IV	5-8
I\$BM05	Read binary from 9-track.	IV	E-7
I\$BM10	Read binary from 7-track.	IV	E-7
ICES\$	Initialize the command environment.	III	5-8
IDCHK\$	Validate a name.	III	2-27
IMADD	Matrix addition (integer).	IV	18-9
IMADJ	Calculate adjoint matrix (integer).	IV	18-11
IMCOF	Calculate signed cofactor (integer).	IV	18-13
IMCON	Set matrix to constant matrix (integer).	IV	18-16
IMDET	Calculate matrix determinant (integer).	IV	18-18
IMIDN	Set matrix to identity matrix (integer).	IV	18-20
IMMLT	Matrix multiplication (integer).	IV	18-24
IMSCL	Multiply matrix by scalar (integer).	IV	18-26
IMSUB	Matrix subtraction (integer).	IV	18-28
IMTRN	Calculate transpose matrix (integer).	IV	18-30
IN\$LO	Determine if a forced logout is in progress.	III	2-28
INSERT	Insertion sort.	IV	17-52
IOA\$	Provide free-format output.	III	3-32
IOA\$ER	Provide free-format output for error messages.	III	3-38
IOA\$RS	Perform free-format output to a buffer.	III	6-32
IOCS\$F	Free logical unit.	IV	3-4
IOCS\$G	Get logical unit.	IV	3-2
ISACL\$	Determine whether an object is ACL-protected.	II	2-25
IS\$AB	Allocate an ISC message buffer.	V	10-5
IS\$AS	Accept an ISC session.	V	8-9
IS\$CE	Clear an ISC session exception.	V	11-7
IS\$FB	Free an ISC message buffer.	V	10-7
IS\$GE	Get an ISC session exception.	V	11-5
IS\$GRQ	Get an ISC session request.	V	8-6
IS\$GRS	Get an ISC session request response.	V	8-12
IS\$GSA	Get ISC session attributes.	V	14-4
IS\$GSO	Get list of ISC sessions owned by this server.	V	14-2
IS\$GSS	Get ISC session status information.	V	14-7
IS\$RM	Receive an ISC message.	V	10-12
IS\$RS	Request an ISC session.	V	8-3
IS\$SM	Send an ISC message.	V	10-9
IS\$STA	Get ISC current session statistics.	V	14-10
IS\$TS	Terminate an ISC session.	V	11-3
ISN\$C	Catalog server's Low Level Name.	V	7-5
ISN\$L	Look up server's Low Level Name.	V	7-7
ISN\$RC	Recatalog server's Low Level Name File.	V	7-8
ISN\$UC	Uncatalog (delete) server's Low Level Name.	V	7-9
ISREM\$	Determine whether an open file system object is local or remote.	II	4-56

JSTR\$A	Left-justify, right-justify, or center a string.	IV	10-10
KLM\$IF	Enable a program to obtain serialization data from a specified file.	III	5-8a
L\$xy series	FORTTRAN compiler complex number loading.	I	B-5
LDISK\$	Return information on the system's list of logical disks.	II	4-58
LIMIT\$	Set and read various timers.	III	8-36
LINEQ	Solve a system of linear equations (single precision).	IV	18-7
LIST\$CMD	Return a list of commands valid at mini-command level.	II	6-16
LN\$SET	Modify user's search rules to permit dynamic linking to EPF library.	II	5-26
LOGO\$\$	Log out a user.	III	2-29
LON\$CN	Switch logout notification on or off.	III	5-20
LON\$R	Read logout notification information.	III	5-21
LOV\$SW	Indicate if the Login-over-login function is currently permitted.	III	2-13
LSTR\$A	Locate one string within another.	IV	10-12
LSUB\$A	Locate one substring within another.	IV	10-14
LUDEV\$	Return a list of devices that a user can access.	III	2-31
LUISK\$	List the disks a given user is using.	II	4-61
LV\$GET	Retrieve the value of a CPL local variable.	II	6-18
LV\$SET	Set the value of a CPL local variable.	II	6-20
M\$xy series	FORTTRAN compiler multiplication routines.	I	B-8
MADD	Matrix addition (single precision).	IV	18-9
MADJ	Calculate adjoint matrix (single precision).	IV	18-11
MCHR\$A	Move a character from one packed string to another.	IV	10-16
MCOF	Calculate signed cofactor (single precision).	IV	18-13
MCON	Set matrix to constant matrix (single precision).	IV	18-16
MDET	Calculate matrix determinant (single precision).	IV	18-18
MGSET\$	Set the receiving state for messages.	III	9-5
MIDN	Set matrix to identity matrix (single precision).	IV	18-20
MINV	Calculate inverted matrix (single precision).	IV	18-22
MKLB\$F	Convert FORTRAN statement label to PL/I format.	III	7-20
MKON\$F	Create an on-unit (for FTN users).	III	7-21
MKON\$P	Create an on-unit (for any language except FTN).	III	7-23

MKONUS	Create an on-unit (for PMA and PL/I users).	III	7-25
MM\$MLPA	Make the last page of a segment available.	III	4-4a
MM\$MLPU	Make the last page of a segment unavailable.	III	4-4b
MMLT	Matrix multiplication (single precision).	IV	18-24
MOVEW\$	Move a block of memory.	III	6-34
MRG1\$S	Merge sorted files.	IV	17-33
MRG2\$	Return next merged record.	IV	17-37
MRG3\$S	Close merged input files.	IV	17-38
MSCL	Matrix addition (single precision).	IV	18-26
MSG\$ST	Return the receiving state of a user.	III	9-3
MSTR\$A	Move one string to another.	IV	10-18
MSUB	Matrix subtraction (single precision).	IV	18-28
MSUB\$A	Move one substring to another.	IV	10-20
MTRN	Calculate transpose matrix (single precision).	IV	18-30
N\$xy series	FORTTRAN compiler negation functions.	I	B-5
NAMEQ\$	Compare two character strings.	III	6-35
NLEN\$A	Determine the operational length of a string.	IV	10-22
NT\$LTS	Return characteristics of PRIMOS network terminal service line.	IV	8-36
O\$AA01	Write ASCII to terminal or command stream.	IV	6-6
O\$AC03	Parallel interface to card punch.	IV	7-31
O\$AC15	Parallel interface punch and print.	IV	7-32
O\$AD07	Write compressed ASCII to disk.	IV	E-2
O\$AD08	Write ASCII uncompressed to disk.	IV	5-10
O\$ALxx	Interface to various printer controllers.	IV	7-1
O\$AL04	Centronics line printer.	IV	7-3
O\$AL06	Parallel interface to MPC line printer.	IV	7-3
O\$AL14	Versatec printer/plotter interface.	IV	7-13
O\$AM05	Write ASCII to 9-track tape.	IV	E-7
O\$AM10	Write ASCII to 7-track tape.	IV	E-7
O\$AM11	Write BCD to 7-track tape.	IV	E-7
O\$AM13	Write EBCDIC to 9-track tape.	IV	E-7
O\$BD07	Write binary to disk.	IV	5-6
O\$BM05	Write binary to 9-track tape.	IV	E-7
O\$BM10	Write binary to 7-track tape.	IV	E-7
O\$BP02	Punch paper tape (binary).	IV	6-15
OPEN\$A	Open supplied filename.	IV	15-6
OPNP\$A	Read filename and open.	IV	15-8
OPNV\$A	Open filename with verification and delay.	IV	15-10
OPSR\$	Locate a file using a search list and open the file.	II	7-4

OPSR\$	Locate a file using a search list and a list of suffixes.	II	7-10
OPVP\$A	Read filename and open, or verify and delay.	IV	15-13
OVERFL	Check if an overflow condition has occurred.	III	10-7
P1IB	Input character from paper tape reader to Register A.	IV	6-17
P1IN	Input character from paper tape to variable.	IV	6-19
P1OB	Output character from Register A to paper-tape punch.	IV	6-18
P1OU	Output character from variable to paper-tape punch.	IV	6-20
PA\$DEL	Remove an object's priority access.	II	2-27
PA\$LST	Obtain the contents of an object's priority ACL.	II	2-28
PA\$SET	Set priority access on an object.	II	2-30
PAR\$RV	Return a logical value indicating ACL and quota support.	II	4-63
PERM	Generate matrix permutations.	IV	18-32
PHANT\$	Start a phantom process.	III	10-8
PHNTM\$	Start up a phantom process.	III	5-23
PL1\$NL	Perform a nonlocal GOTO.	III	7-27
POSN\$A	Position file.	IV	15-17
PRERR	Print an error message.	III	10-9
PRI\$RV	Return operating system revision number.	III	2-15
PRJID\$	Return the user's project identifier.	III	2-34
PRWF\$	Read, write, position, or truncate a file.	II	4-65
PTIME\$	Return amount of CPU time used since login.	III	2-35
PWCHK\$	Validate syntax of a password.	III	2-36
Q\$READ	Return directory quota and disk record usage information.	II	4-74
Q\$SET	Set a quota on a subdirectory of the current directory.	II	4-77
QUICK	Partition exchange sort.	IV	17-54
QUIT\$	Determine if there are pending quits.	III	3-62
RADXEX	Radix exchange sort.	IV	17-55
RAND\$A	Generate random number and update seed, using 32-bit word size and the linear congruential method.	IV	13-2
RD\$CE_DP	Return caller's current command environment breadth.	II	6-22
RDASC	Read ASCII from any device.	IV	4-5

RDBIN	Read binary from any device.	IV	4-9
RDEN\$\$	Position in or read from a directory.	II	A-9
RDLIN\$	Read a line of characters from a compressed ASCII disk file.	II	4-80
RDTK\$\$	Parse a command line.	III	3-16
READY\$	Display PRIMOS command prompt.	III	2-37
RECYCL	Tell PRIMOS to cycle to the next user.	III	10-18
REMEPF\$	Remove an EPF from a user's address space.	II	5-22
REST\$\$	Restore an R-mode executable image.	III	5-13
RESU\$\$	Restore and resume an R-mode executable image.	III	5-15
RLSE\$\$	Get input records after SETU\$.	IV	17-26
RMSGD\$	Receive a deferred message.	III	9-7
RNAM\$A	Prompt, read a pathname, and check format.	IV	11-2
RNDI\$A	Initialize random number generator seed.	IV	13-4
RNUM\$A	Prompt and read a number (in any format).	IV	11-4
RPL\$	Replace one EPF runfile with another.	II	5-24
RPOS\$A	Return position of file.	IV	15-18
RRECL	Read disk record.	IV	5-14
RSEGAC\$	Determine access to a segment.	III	2-16
RSTR\$A	Rotate string left or right.	IV	10-23
RSUB\$A	Rotate substring left or right.	IV	10-26
RTRN\$\$	Get sorted records.	IV	17-28
RVON\$F	Revert an on-unit (for FTN users).	III	7-28
RVONU\$	Revert an on-unit (for any language except FTN).	III	7-29
RWND\$A	Reposition file.	IV	15-19
S\$xy series	FORTTRAN compiler subtraction routines.	I	B-8
SATR\$\$	Set or modify an object's attributes.	II	4-82
SAVE\$\$	Save an R-mode executable image.	III	5-17
SCHAR	Store a character into an array location.	III	6-37
SEM\$CL	Release (close) a named semaphore.	III	8-17
SEM\$DR	Drain a semaphore.	III	8-19
SEM\$NF	Notify a semaphore.	III	8-21
SEM\$OP	Open a set of named semaphores.	III	8-23
SEM\$OU	Open a set of named semaphores.	III	8-23
SEM\$TN	Periodically notify a semaphore.	III	8-27
SEM\$TS	Return number of processes waiting on a semaphore.	III	8-29
SEM\$TW	Wait on a specified named semaphore, with timeout.	III	8-31
SEM\$WT	Wait on a semaphore.	III	8-33
SETRC\$	Record command error status.	III	5-9

SETU\$\$	Prepare sort table and buffers for CMBN\$.	IV	17-22
SGD\$DL	Delete a segment directory.	II	4-88
SGD\$EX	Find out if there is a valid entry at the current position within the segment directory on a specified unit.	II	4-90
SGD\$OP	Open a segment directory entry.	II	4-92
SGDR\$\$	Position, read, or modify a segment directory.	II	4-94
SGNL\$F	Signal a condition.	III	7-30
SHELL	Diminishing increment sort.	IV	17-56
SID\$GT	Return user number of initiating process.	III	2-38
SIGNL\$	Signal a condition.	II	7-32
SIZE\$	Return the size of a file system entry.	II	4-100
SLEEP\$	Suspend a process for a specified interval.	III	8-39
SLEP\$I	Suspend a process (interruptible).	III	8-40
SLITE	Set the sense light on or off.	III	10-12
SLITET	Test sense light settings.	III	10-13
SMSG\$	Send an interuser message.	III	9-9
SNCHK\$	Check validity of system name passed to it.	III	2-18
SP\$REQ	Insert a file into the spool queue.	IV	7-12c
SPAS\$\$	Set the owner and nonowner passwords on an object.	II	2-32
SPOOL\$	Insert a file in spooler queue.	IV	7-9
SR\$ABSDS	Disable an optional search rule enabled by SR\$ENABL.	II	7-17
SR\$ADDB	Add a rule to the start of a search list or before a specified rule within the list.	II	7-20
SR\$ADDE	Add a rule to the end of a search list or after a specified rule within the list.	II	7-23
SR\$CREAT	Create a search list.	II	7-26
SR\$DEL	Delete a search list.	II	7-28
SR\$DSABL	Disable an optional search rule enabled by SR\$ENABL.	II	7-30
SR\$ENABL	Enable an optional search rule.	II	7-33
SR\$EXSTR	Determine if a search rule exists.	II	7-36
SR\$FR_LS	Free list structure space allocated by SR\$LIST or SR\$READ.	II	7-40
SR\$INIT	Initialize all search lists to system defaults.	II	7-42
SR\$LIST	Return the names of all defined search lists.	II	7-44
SR\$NEXTR	Read the next rule from a search list.	II	7-48
SR\$READ	Read all of the rules in a search list.	II	7-53
SR\$REM	Remove a rule from a search list.	II	7-57
SR\$SETL	Set the locator pointer for a search rule.	II	7-60
SR\$SSR	Set a search list via a user-defined search rules file.	II	7-63
SRCH\$\$	Open, close, delete, or verify existence of an object.	II	4-103

SRS\$GN	Get server name.	V	7-10
SRS\$GP	Get process numbers of all processes that have same server name.	V	7-11
SRS\$LN	List all active server names.	V	7-13
SRSFX\$	Search for a file with a list of possible suffixes.	II	4-112
SRTF\$\$	Sort several input files.	IV	17-16
SS\$ERR	Signal an error in a subsystem.	III	5-11
SSTR\$A	Shift string left or right.	IV	10-28
SSUB\$A	Shift substring left or right.	IV	10-30
SSWTCH	Test sense switch settings.	III	10-14
ST\$SGS	Return maximum number of static segments.	III	4-26
STR\$AL	Allocate user-class dynamic memory.	III	4-5
STR\$AP	Allocate process-class dynamic memory.	III	4-7
STR\$AS	Allocate subsystem-class dynamic memory.	III	4-8
STR\$AU	Allocate user-class dynamic memory.	III	4-10
STR\$FP	Free process-class dynamic memory.	III	4-11
STR\$FR	Free user-class dynamic memory.	III	4-12
STR\$FS	Free subsystem-class dynamic memory.	III	4-13
STR\$FU	Free user-class dynamic memory.	III	4-14
SUBSRT	Sort file on ASCII key (V-mode).	IV	17-10
SUBSRT	Sort file on ASCII key (R-mode).	IV	17-40
SUSR\$	Test if current user is supervisor.	III	2-39
SYN\$CHCK	Return total of notices or waiters on a synchronizer.	V	4-2
SYN\$CREA	Create an event synchronizer.	V	2-5
SYN\$DEST	Destroy an event synchronizer.	V	2-15
SYN\$GCHK	Return total of notices or waiters on an event group.	V	4-4
SYN\$GCRE	Create an event group.	V	3-5
SYN\$GDST	Destroy an event group.	V	3-18
SYN\$GLST	List total of groups in server and their identifiers.	V	4-12
SYN\$GRTR	Retrieve a notice from a group.	V	3-15
SYN\$GTWT	Perform a timed wait on a group.	V	3-13
SYN\$GWT	Wait on an event group.	V	3-11
SYN\$INFO	Return information about a synchronizer.	V	4-6
SYN\$LIST	List total of synchronizers in server and their identifiers.	V	4-10
SYN\$LSIG	List total of synchronizers in group and their identifiers.	V	4-8
SYN\$MVTO	Move a synchronizer into a group.	V	3-7
SYN\$POST	Post a notice on a synchronizer.	V	2-7
SYN\$REMV	Remove a synchronizer from a group.	V	3-9
SYN\$RTRV	Retrieve a notice from an event synchronizer.	V	2-13
SYN\$TMWT	Perform a timed wait on an event synchronizer.	V	2-11
SYN\$WAIT	Wait on an event synchronizer.	V	2-9

T\$AMLC	Communicate with AMLC driver.	IV	8-23
T\$CMPC	Input from MPC card reader.	IV	7-28
T\$LMPC	Move data to LPC line printer.	IV	7-6
T\$MT	Raw data mover for tape.	IV	7-37
T\$PMPC	Raw data mover for card reader.	IV	7-34
T\$SLC0	Communicate with SMLC driver.	IV	8-3
T\$VG	Interface to Versatec printer.	IV	7-16
T1IB	Read a character (function) from PMA into Register A.	III	3-23
T1IN	Read a character (procedure).	III	3-24
T1OB	Write one character from Register A.	III	3-47
T1OU	Write one character.	III	3-48
TEMP\$A	Open a scratch file.	IV	15-20
TEXT0\$	Check filename for valid format.	III	10-15
TI\$MSG	Display standard message showing times used.	III	2-40
TIDEC	Read a decimal number.	III	3-26
TIHEX	Read a hexadecimal number.	III	3-27
TIMDAT	Return timing information and user identification.	III	2-42
TIME\$A	Return time of day.	IV	12-7
TIOCT	Read an octal number.	III	3-28
TL\$SGS	Return highest segment number.	III	4-27
TMR\$CANL	Cancel a timer.	V	5-15
TMR\$CREA	Create a timer.	V	5-6
TMR\$DEST	Destroy a timer.	V	5-8
TMR\$GINF	Return permanent time information.	III	2-43b
TMR\$GTIM	Return current system time.	III	2-43d
TMR\$GTMR	Return information about a timer.	V	5-16
TMR\$LIST	List total number of timers in server and their identifiers.	V	5-19
TMR\$LOCALCONVERT	Convert local time to Universal Time.	III	2-43e
TMR\$SABS	Set an absolute timer.	V	5-9
TMR\$SINT	Set an interval timer.	V	5-11
TMR\$SREP	Set a repetitive timer.	V	5-13
TMR\$UNIVCONVERT	Convert Universal Time to local time.	III	2-43g
TNCHK\$	Verify a supplied string as a valid pathname.	II	4-118
TNOU	Write characters to terminal, followed by NEWLINE.	III	3-40
TNOUA	Write characters to terminal.	III	3-41
TODEC	Write a signed decimal number.	III	3-42
TOHEX	Write a hexadecimal number.	III	3-43
TONL	Write a NEWLINE.	III	3-44
TOOCT	Write an octal number.	III	3-45
TOVFD\$	Write a decimal number without spaces.	III	3-46

TREE\$A	Test for pathname.	IV	10-32
TRNC\$A	Truncate a file.	IV	15-22
TSCN\$A	Scan the file system tree structure.	IV	15-23
TSRC\$\$	Open, close, delete, or find a file anywhere in the file structure.	II	A-17
TTY\$IN	Check for unread terminal input characters.	III	3-63
TTY\$RS	Clear the terminal input and output buffers.	III	3-65
TYPE\$A	Determine string type.	IV	10-35
UID\$BT	Return unique bit string.	III	6-39
UID\$CH	Convert UID\$BT output into character string.	III	6-40
UNIT\$A	Check for file open.	IV	15-28
UNIT\$\$	Return caller's minimum and maximum file unit numbers.	II	4-121
UNO\$GT	List users with same name as caller.	III	2-44
UPDATE	Update current directory (PRIMOS II only).	III	10-17
USER\$	Return user number and count of users.	III	2-20
UTYPE\$	Return user type of current process.	III	2-45
VALID\$	Validate a name against composite identification.	III	2-48
WILD\$	Return a logical value indicating whether a wildcard name was matched.	II	4-122
WRASC	Write ASCII.	IV	4-3
WRBIN	Write binary to any output device.	IV	4-7
WRECL	Write disk record.	IV	5-17
WTLIN\$	Write a line of characters to a compressed ASCII file.	II	4-124
YSNO\$A	Ask question and obtain a yes or no answer.	IV	11-7
Z\$80	Clear double-precision exponent.	I	B-5

A

Absolute timers, 5-1
 cancelling, 5-15
 creating, 5-7
 destroying, 5-8
 getting information about, 5-16
 setting, 5-3, 5-10
 states of, 5-3

ACL groups
 null, 13-2
 of session initiator, 8-8

Allocate message buffer, 10-5

ARID command, 13-1

AttributeIdentityBlock, 14-5, E-2

Authentication
 of session initiator, 8-2, 8-6, 8-8
 of session recipient, 8-14
 possible uses of, 8-8
 user ID, 13-1
see also: AttributeIdentityBlock,
 InitiatorAuthBlock,
 TargetAuthBlock

B

BIND, ISN\$ subroutines, 7-3

Buffer
see: Message buffer

BufferAvailable synchronizer
 configuring, 9-6
 exception notice on, 10-6
 use of, 10-6

C

Child process
 ICE -SERVER, 7-1
 server name of, 7-1

Configuration
see: Session configuration

Connect messages, 1-3, 6-3, 12-1
 actual length, 12-1
 after sending, 12-1
 configuration, 9-4, 12-2
 during session request, 8-3, 8-6
 during session termination, 11-3
 maximum length, 9-4

receiving, 8-6, 8-13, 11-5
 sending, 8-3, 8-9, 11-3
 truncated, 8-7, 8-13, 11-6
 use of MessageSpecifier, 10-3

Control part, 6-2, 10-3
 actual length, 10-3
 Connect messages, 6-3, 10-3
 Expedited messages, 6-3
 location, 10-3
 maximum length, 9-4
 message array for, 6-5
 modifying, 10-11
 Normal messages, 6-2
 sending, 10-1

Creating a message, 10-1, 10-3

D

Data part, 6-2, 10-4
 actual length, 10-4, 10-6
 allocate buffer, 10-5
 allocated length, 10-5
 location, 10-4
 maximum length, 9-4
 modifying, 10-11
 Normal messages, 6-2
 sending, 10-1

Data structures
see: Structures

Data transfer
see: Message exchange

Data type equivalents, F-1

Delivery failure
 detecting, 11-5
 of connect messages, 12-1
 recovery from, 13-2
 virtual circuit reset, 13-2

E

ER\$PRINT, 1-4

Error messages
 ERRD, 1-4, 7-2
 for ISN\$ subroutines, 7-2
 for PRIMOS subsystems, 1-4
 retrieving, 1-4
see also: Status codes

ErrorMsgHdlr.INS.language, 1-4

ER\$TEXT, 1-4

Event groups, 1-1, 2-1, 3-1, 9-12
 creating, 3-2, 3-6
 destroying, 3-4, 3-18
 ExceptionPending synchronizer,
 11-2
 getting count of synchronizers in,
 4-9
 getting identifiers of synchronizers
 in, 4-9
 identifiers of, 3-2
 in PL/I sample program, B-1,
 B-14
 number of priority levels specified
 by SYN\$GCRE, 3-6
 performing a timed wait on, 3-4,
 3-14
 priority levels of, 3-2
 retrieving information about, 4-1,
 4-5
 retrieving notices from, 3-4, 3-16
 subroutines that access, 3-1
 system-defined limits for, D-1
 waiting on, 2-1, 3-3, 3-12

Event synchronization, 1-1
 used by ISC, 1-2
 used by timers, 1-2

Event synchronizers, 1-1, 2-1
 creating, 2-2, 2-6
 deallocated by ICE -SERVER,
 2-4
 destroying, 2-2, 2-4, 2-16
 in F77 sample program, B-4
 in PL/I sample program, B-1
 maximum number of, D-2
 moving into event groups, 3-8
 moving into groups, 3-3
 performing a timed wait on, 2-3,
 2-12
 posting notices on, 2-3, 2-7
 removing from event groups, 3-3,
 3-9
 retrieving information about, 4-1,
 4-6
 retrieving notices from, 2-3, 2-14
 subroutines that access, 2-2

- Event synchronizers (continued)
 system-defined limits for, D-1
 using, 2-2
 waiting on, 2-3, 2-10
see also: ISC synchronizers
- ExceptionPending synchronizer, 11-2
 configuring, 9-6
 if not configured, 11-2
 using, 11-6
- Exceptions, 6-2, 11-1
 clearing, 11-7
 Connect message, 11-6
 delivery failure exception, 11-5, 13-2
 getting, 11-5
 handling, 11-1
 number during session, 14-11
 number pending, 14-8
 server terminate exception, 11-5
 status code, 11-6
 system terminate exception, 11-5, 13-2
 types, 11-5
- Expedited messages, 1-3, 6-3
 configuring, 9-4
 length, 9-5
 number sent & received, 14-11
 receiving, 9-12, 10-12
 remote session, 13-2
 sending, 10-9
- F**
- For Client Use field (event groups)
 possible uses of, 3-8, 9-12
 returned by SYN\$GRTR, 3-15
 returned by SYN\$GTWT, 3-4, 3-13 to 3-14
 returned by SYN\$GWT, 3-3, 3-11 to 3-12
 returned by SYN\$INFO, 4-6
 specified in call to SYN\$MVTO, 3-7
 specified in call to SYN\$MVTO, 3-3
- For Client Use field (Low Level Name)
 changing, 7-8
 specifying, 7-3
 updating, 7-4
- use during session request, 8-3, 8-6
- FORTRAN
see: FTN
- Free message buffer, 10-7
- FTN, key values, 6-4, C-1
- G**
- Global parameters
see: Session configuration
- H**
- High Level Name File
see: HLNLF
- HLNLF
 ACL protection of, 7-3
 cataloging LLN in, 7-5
 creating, 7-5
 deleting, 7-9
 filename of, 7-3
 looking up LLN in, 7-7
- I**
- ICE command, 7-1
- ICE -SERVER command, 7-1
 child processes, 7-1
 ISC synchronizers, 9-9
 server name, 7-1
 session termination, 7-1
 SessionRequestPending synchronizer, 9-8
 synchronizer deallocation, 2-4
 timer deallocation, 5-3
- Include files, 1-3, 6-3
 different values for FTN, 2-4, 5-4
 for event synchronizers, 2-4
 for ISC keys, 6-4, C-3
 for ISC structures, 6-4
 for server names, 7-2
 for SRS\$ subroutines, 6-4
 for timers, 5-4
 language support, 6-4
- Initialize Command Environment
see: ICE -SERVER command
- Initialize server
see: ICE -SERVER command
- Initiator
see: Session initiator
- InitiatorAuthBlock, 8-8, E-2
- InterServer Communications
see: ISC
- Interval timers, 5-1
 cancelling, 5-15
 creating, 5-7
 destroying, 5-8
 getting information about, 5-16
 setting, 5-3, 5-12
 states of, 5-3
- IS\$AB, 10-5
- IS\$AS, 8-9
- ISC, 1-1 to 1-2, 6-1
 and event groups, 3-1
 and synchronizers, 2-1
- ISC synchronizers, 1-2, 6-1
 configuring, 9-5, 9-7
 deallocated by ICE -SERVER, 9-9
 destroying, 7-1, 9-8
 exception notices on, 11-2
 identifiers for, 8-3, 9-9, 14-4
 in event groups, 9-12
 list session recipient's, 8-9
 non-ISC synchronizers, 9-8, 9-12
 posting a notice, 6-1, 9-10
 retrieving a notice, 6-5, 9-10
 too many notices, 9-9
 unusable synchronizers, 9-9
 unused synchronizers, 9-9
 using, 9-10
 waiting on, 9-10
see also: Event synchronizers
- IS\$CE, 11-7
- ISC_KEYS.INS.language, 1-4, 6-4
- ISC_NETWORK_SERVER, 13-1
- ISC_STRUCTURES.INS.language, 1-4
- IS\$FB, 10-7
- IS\$GE, 11-5
 exception codes, 6-3, C-5
- IS\$GRQ, 8-6
- IS\$GRS, 8-12
 response codes, 6-3, C-4
- IS\$GSA, 14-4
- IS\$GSO, 14-2
- IS\$GSS, 14-7
 phase codes, 6-3, C-5
- ISN\$ subroutines, using BIND with, 7-3

ISNSC, 7-5

ISNSL, 7-7

ISNSRC, 7-8

ISNSUC, 7-9

ISSRM, 10-12

ISSRS, 8-3

ISSSM, 10-9

examples, 9-11

ISSSTA, 14-10

ISSTS, 11-3

to reject a session request, 8-10

K

Key files

see: Include files

L

Language support

include files, 6-4, C-1

structure files, 6-4

LIST_SERVER_NAMES command,
6-3, 7-1

LIST_SESSIONS command, 6-3

LLN, 7-1

cataloging, 7-3, 7-5

looking up a server's, 7-7

node name, 13-1

recatalog in HLNF, 7-8

session request using, 8-3, 8-6

structure, 7-2, E-3

Local parameters

see: Session configuration

Login

cataloging LLN during, 7-3 to 7-4

server name assigned during, 7-1

SessionRequestPending

synchronizer, 9-8

Logout

LLN invalidated, 7-4

SessionRequestPending

synchronizer, 9-8

terminates server's sessions, 6-2

uncatalog HLNF, 7-9

Low Level Name

see: LLN

M

Message area, 9-5

block size, 9-5

configuring, 9-5

default size, 9-5

maximum number of, D-1

maximum size, 9-5

no available space, 10-6

not allocated, 9-5, 14-9, D-1

number of blocks, 9-5

percent in use, 14-12

percent used, 14-12

sharing, 9-7, 14-9

where allocated, D-1

Message buffer

allocating, 10-5

allocation failure, 10-6, 14-12

freeing, 10-7, 10-11

modifying after sending, 10-11

ownership of, 10-8

pointer to, 10-6

reusing, 10-8

Message exchange, 6-2, 10-1

creating a message, 10-1

sequence of events, 10-1

status, 14-8

Message queues

configuring, 9-4

not configured, 9-4

receive queue status, 10-2

selecting, 10-9

send queue status, 10-2

setting lengths, 9-4

status of, 14-9

using send queue, 9-11

Messages

after sending, 10-11, 12-1

data loss, 13-2

number pending, 14-9

see also: Connect messages,

Control part, Data part,

Expedited messages, Normal

messages, Null messages

MessageSpecifier

connect messages, 12-1

control part fields, 10-3

data part fields, 10-4

data part location, 10-6

structure, 10-3, E-3

using, 10-1, 10-3

Multiple sessions, 6-1

between the same servers, 6-1

coordinating events, 9-12

event groups for, 9-12

listing, 14-2

maximum number of, D-1

requesting a session, 8-7

sharing a message area, 9-7, 14-9

synchronizer identifiers, 9-10

N

Node name, 7-3

determining your own, 7-3

in LLN, 7-3, 13-1

of other server, 14-5

of session initiator, 8-8

of session recipient, 8-14

Nodes, session between different,
1-3, 13-1

Normal messages, 1-3, 6-2

allocate data part buffer, 10-5

configuring, 9-4

control part length, 9-4

data part length, 9-4, 10-4

null messages, 6-2

number sent & received, 14-11

receiving, 10-12

sending, 10-9

see also: Message area

Notices

count returned by SYN\$CHCK,
4-3

initial count, 2-2, 2-5 to 2-6

order in which returned, 3-2

posting on event synchronizers,

1-1, 2-1, 2-3, 2-7

retrieving from a synchronizer,

2-1, 2-3, 9-11

retrieving from an event group,

3-4, 3-16, 9-10

when posted, 9-6

Null messages, 10-3, 10-10

P

Phantom, 6-1

server name of, 7-1

PRIMENET, 1-3, 6-1, 13-1

PRIMIX, 1-2

Priority levels
 and processing speed, 3-6
 determine order of response to events, 3-3
 number of specified by SYN\$GCRE, 3-2, 3-5 to 3-6
 setting number of, 3-2, 3-5
 specifying a level for a synchronizer, 3-3

Process
 child process, 6-1
 list all, 7-10
 server name of, 1-2, 7-1, 7-10
 sharing a server name, 7-11
 slave process, D-2

Program examples
 InterServer Communications, B-8
 synchronizers and groups, B-1
 timers, B-6

Programming considerations
 BIND with ISN\$ subroutines, 7-3
 include files, 1-3, 6-4
 limits, D-1
 retrieving error messages, 1-4
 structure template files, 6-4

Project ID
 null, 13-2
 of session initiator, 8-8

Q

Queue thresholds
 configuring, 9-6
 examples, 9-11
 invalid value, 8-10
 using, 9-11

Queues
see: Message queues

R

ReadyToReceive synchronizer
 configuring, 9-6
 exception notice on, 10-14
 using, 10-13

ReadyToReceiveExpedited synchronizer
 configuring, 9-6
 exception notice on, 10-14
 in an event group, 9-12
 using, 10-13

ReadyToSend synchronizer
 configuring, 9-5
 exception notice on, 10-11
 using, 9-11, 10-11
 ReadyToSendExpedited synchronizer
 configuring, 9-5
 exception notice on, 10-11
 using, 10-11
 Receiving a message, 10-1, 10-12
 Recipient
see: Session recipient
 Remote ID, 13-1
 Remote session, 6-1
 maximum number of, D-2
 Repetitive timers, 5-1
 cancelling, 5-15
 creating, 5-7
 destroying, 5-8
 getting information about, 5-16
 setting, 5-3, 5-14
 states of, 5-4

S

Security
see: Authentication, HLNf
 Sending a message, 10-1, 10-9
 available queue space, 9-11
 programming options, 9-11
 retrieving notices, 10-11
 when requesting a session, 8-5
 when terminating a session, 11-3
 Server name, 1-2, 7-1
 determining, 7-10
 invalidate, 7-1
 list all, 7-1, 7-13
 LIST_SERVER_NAMES
 command, 7-1
 process numbers for, 7-11

Servers
 and PRIMIX, 2-1
 defined for ISC, 1-2, 6-1
 event groups are private to, 3-2
 event synchronization within, 1-2, 2-3, 2-8
 event synchronizers are private to, 2-1
 getting count of event groups in, 4-13
 getting count of synchronizers in, 4-11

getting count of timers in, 5-19
 getting identifiers of groups in, 4-13
 getting identifiers of synchronizers in, 4-11
 getting identifiers of timers in, 5-19
 initialize server, 7-1
 list sessions of, 14-2
 process numbers for, 7-11
 shared by multiple processes, 7-11
 timers are private to, 5-1
see also: Server name
 Session, 1-2, 6-1, 8-1
 attributes of, 14-4
 establishment, 6-2, 8-1
 get request for, 8-6
 get response to request, 8-12
 identified by number, 8-2
 initiator, 8-12
 listing server's sessions, 14-2
 maximum number of, D-1
 message exchange, 10-1
 no request pending, 8-7
 participants in, 1-2
 phases of, 6-2, 14-8
 recipient, 8-9
 remote session, 6-1, 13-1
 request accepted, 8-9, 8-12
 request rejected, 8-12
 requesting, 6-2, 8-2 to 8-3
 requests pending, D-2
 server terminate, 11-5
 statistics, 14-10
 status of, 14-7
 system cannot establish, 8-12
 system terminate, 11-5
 termination, 6-2, 7-1, 11-1
 too many requests, 8-4
 too many sessions, 8-4, 8-7
 with oneself, 8-2, D-2
 Session configuration, 9-1
 actual parameter values, 14-6
 connect message exchange, 12-2
 defaults, 8-3, 9-1
 errors in, 9-9
 existing sessions, 9-7
 get global values, 8-6
 get parameter values, 14-4
 global parameters, 8-3, 9-1, 9-3

- Session configuration (continued)
 - local parameters, 9-1, 9-5
 - overview, 9-3
 - recipient's local parameters, 8-9
 - reserved fields, 8-4, 8-10
 - setting global parameters, 9-3
 - setting local parameters, 8-11, 9-5
 - setting parameters, 8-3, 9-3
- Session establishment, 8-1
 - calling subroutines during, 14-1
 - halting, 11-3
 - messages during, 12-1
 - requesting, 8-1
 - session attributes during, 14-6
 - status, 14-8
- Session initiator, 8-1, 8-3, 14-6
 - program example, B-8
- Session number, 8-2
 - initiator's, 8-3
 - recipient's, 8-6
- Session recipient, 8-1, 8-6, 14-6
 - program example, B-14
- Session request
 - accept or reject, 8-8
 - accepting, 8-2, 8-9
 - maximum number of, D-2
 - rejecting, 8-2, 11-4
 - requesting, 8-2 to 8-3
- Session synchronizers list
 - see*: SessionSyncList
- Session termination, 11-1
 - calling subroutines during, 11-1, 14-1
 - messages during, 12-1
- SessionConfigurationBlock, 9-2
 - null pointer, 9-1
 - reserved parameters, 9-3
 - structure, 9-2, E-4
 - version parameter, 9-3
 - see also*: Session configuration
- SessionRequestPending synchronizer, 9-8
 - identifier of, 7-10
 - using, 8-2, 8-7
- SessionResponsePending synchronizer
 - configuring, 9-6
 - deletion of, 9-8
 - notice posted on, 8-14
 - using, 8-2
- Sessions, no sessions, 14-2
- SessionStatisticsBlock, 14-11, E-5
- SessionStatusBlock, 14-8, E-5
- SessionSyncList, 9-9, 14-4, E-6
 - for session initiator, 8-3
 - for session recipient, 8-9
- Slave processes, D-1 to D-2
- Software requirements, 13-1
- SRS_CODES.INS.language, 1-4, 6-4, 7-2
- SRSSGN, 7-10
- SRSSGP, 7-11
- SRSSLN, 7-13
- Status codes, 6-3, C-1
 - FTN language, 6-4
 - listing, C-1
 - message text, 6-4
 - returned by ISS subroutines, C-3
 - returned by SRS\$ subroutines, C-5
 - returned by synchronizer subroutines, C-1
 - returned by timer subroutines, C-2
- Structures, 6-4
 - allocating space for, 6-4
 - listing of, E-1
 - programming example, 6-5
 - template files for, 6-4
 - used by ISC, E-2
 - used by SYN\$INFO, E-1
 - used by TMR\$GTMR, E-1 to E-2
 - version field, 6-5
- Subroutine quick reference, A-1
- SYNC_CODES.INS.language, 1-4, 2-4, 6-4
- SYN\$CHCK, 4-2
- Synchronizers
 - see*: Event synchronizers, ISC synchronizers
- SYNSCK
 - see*: SYN\$CHCK
- SYNSCR
 - see*: SYN\$CREA
- SYN\$CREA, 2-2, 2-5
- SYN\$DE
 - see*: SYN\$DEST
- SYN\$DEST, 2-4, 2-15
- SYN\$GC
 - see*: SYN\$GCRE
- SYN\$GCHK, 4-4
- SYN\$GCRE, 3-2, 3-5
- SYN\$GD
 - see*: SYN\$GDST
- SYN\$GDST, 3-4, 3-18
- SYN\$GK
 - see*: SYN\$GCHK
- SYN\$GL
 - see*: SYN\$GLST
- SYN\$GLST, 4-12
- SYN\$GR
 - see*: SYN\$GRTR
- SYN\$GRTR, 3-2, 3-4, 3-15
- SYN\$GT
 - see*: SYN\$GTWT
- SYN\$GTWT, 3-4, 3-13
- SYN\$GW
 - see*: SYN\$GWT
- SYN\$GWT, 3-3, 3-11
- SYN\$IF
 - see*: SYN\$INFO
- SYN\$INFO, 4-6
 - structure used by, E-1
- SYN\$LG
 - see*: SYN\$LSIG
- SYN\$LIST, 4-10
- SYN\$LS
 - see*: SYN\$LIST
- SYN\$LSIG, 4-8
- SYN\$MV
 - see*: SYN\$MVTO
- SYN\$MVTO, 3-2 to 3-3, 3-7
 - specifies a priority level, 3-3, 3-8
- SYN\$PO
 - see*: SYN\$POST
- SYN\$POST, 2-3, 2-7
- SYN\$REMV, 3-3, 3-9
- SYN\$RM, SYN\$REMV, 3-9
- SYN\$RTRV, 2-3, 2-13
- SYN\$RV
 - see*: SYN\$RTRV
- SYN\$TMWT, 2-3, 2-11
- SYN\$TW
 - see*: SYN\$TMWT
- SYN\$WAIT, 2-3, 2-9
- SYN\$WT
 - see*: SYN\$WAIT
- SYSCOM directory, 1-3, 2-4, 5-4
- SYSOVL directory, 1-4

T

Target
 see: Session recipient
 TargetAuthBlock, 8-14, E-6
 Templates
 see: Structures
 Terminating a session, 6-2, 11-1,
 11-3
 status, 14-8
 Thresholds
 see: Queue thresholds
 TIMERMIK.INS.language, 1-4, 5-4
 Timers, 1-1 to 1-2, 2-1, 5-1
 absolute, 5-1
 and event groups, 3-1
 cancelling, 5-3, 5-15
 creating, 5-2, 5-7
 deallocated by ICE -SERVER,
 2-4
 destroying, 5-3, 5-8
 getting information about, 5-16
 in F77 sample program, B-7
 in PL/I sample program, B-6
 interval, 5-1
 maximum number of, D-1
 post notices, 5-1
 repetitive, 5-1
 resetting, 5-3
 setting, 5-2
 subroutines that access, 5-1
 system-defined limits for, D-1
 TMR\$CANL, 5-15
 TMR\$CN
 see: TMR\$CANL
 TMR\$CR
 see: TMR\$CREA
 TMR\$CREA, 5-2, 5-6
 TMR\$DE
 see: TMR\$DEST
 TMR\$DEST, 5-8
 TMR\$GTMR, 5-16
 structure used by, E-1 to E-2
 TMR\$LIST, 5-19
 TMR\$LS
 see: TMR\$LIST
 TMR\$SA
 see: TMR\$SABS
 TMR\$SABS, 5-2, 5-9
 TMR\$SI
 see: TMR\$SINT

TMR\$SINT, 5-11
 TMR\$SR
 see: TMR\$SREP
 TMR\$SREP, 5-13
 TMR\$TI
 see: TMR\$GTMR

U

User ID, 13-1
 null, 13-2
 of other server, 14-5
 of session initiator, 8-8
 of session recipient, 8-14
 remote ID, 13-1

V

Version number, 6-5, 9-3
 Virtual circuits, 13-2, D-2

W

Waiting processes
 causing to resume execution, 2-3
 count for group returned by
 SYN\$GCHK, 4-5
 count for synchronizer returned by
 SYN\$CHCK, 4-3
 time limits for, 2-11, 3-14
 What_happened codes, returned by
 event synchronizer
 subroutines, C-2

Reader Response Form

Subroutines Reference V: Event Synchronization DOC10213-1LA

Your feedback will help us continue to improve the quality, accuracy, and organization of our user publications.

1. How do you rate this document for overall usefulness?

☐ *excellent* ☐ *very good* ☐ *good* ☐ *fair* ☐ *poor*

2. What features of this manual did you find most useful?

3. What faults or errors in this manual gave you problems?

4. How does this manual compare to equivalent manuals produced by other computer companies?

☐ *Much better* ☐ *Slightly better* ☐ *About the same*
☐ *Much worse* ☐ *Slightly worse* ☐ *Can't judge*

5. Which other companies' manuals have you read?

Name: _____

Position: _____

Company: _____

Address: _____

Postal Code: _____

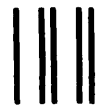
First Class Permit #531 Natick, Massachusetts 01760

BUSINESS REPLY MAIL

Postage will be paid by:



Attention: Technical Publications
Bldg 10
Prime Park, Natick, Ma. 01760



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

