INTERCOMM

STORE/FETCH FACILITY

ISOGON
CORPORATION
330 Seventh Avenue, New York, New York 10001

LICENSE: INTERCOMM TELEPROCESSING MONITOR

Store/Fetch Facility

Publishing History

Publication	Date	Remarks
First Edition	July 1976	This manual corresponds to Intercomm Release 6.2.
Second Edition	February 1994	Updates and corrections corresponding to Intercomm Releases 9 and 10.

PREFACE

Intercomm is a state-of-the-art teleprocessing monitor system of Isogon Corporation, executing on the IBM System/370 and System/390 family of computers and operating under the control of IBM Operating Systems (XA and ESA). Intercomm monitors the transmission of messages to and from terminals, concurrent message processing, centralized access to I/O files, and the routine utility operations of editing input messages and formatting output messages, as required.

This manual documents the Store/Fetch Facility.

The following manuals are to be used in conjunction with this manual:

- Operating Reference Manual
- COBOL Programmers Guide
- PL/1 Programmers Guide
- Assembler Language Programmers Guide

INTERCOMM PUBLICATIONS

GENERAL INFORMATION MANUALS

- Concepts and Facilities
- Planning Guide

APPLICATION PROGRAMMERS MANUALS

- Assembler Language Programmers Guide
- COBOL Programmers Guide
- PL/1 Programmers Guide

SYSTEM PROGRAMMERS MANUALS

- Basic System Macros
- BTAM Terminal Support Guide
- Installation Guide
- Messages and Codes
- Operating Reference Manual
- System Control Commands

CUSTOMER INFORMATION MANUALS

- Customer Education Course Catalog
- Technical Information Bulletins
- User Contributed Program Description

FEATURE IMPLEMENTATION MANUALS

- Autogen Facility
- ASMF Users Guide
- DBMS Users Guide
- Data Entry Installation Guide
- Data Entry Terminal Operators Guide
- Dynamic Data Queuing Facility
- Dynamic File Allocation
- Extended Security System
- File Recovery Users Guide
- Generalized Front End Facility
- Message Mapping Utilities
- Model System Generator
- Multiregion Support Facility
- Page Facility
- Store/Fetch Facility
- SNA Terminal Support Guide
- Table Facility
- TCAM Support Users Guide
- Utilities Users Guide

EXTERNAL FEATURES MANUALS

- SNA LU6.2 Support Guide

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Chapter 1

INTRODUCTION

1.1 OVERVIEW

The Store/Fetch facility provides an application program (subsystem) with the facilities to:

- STORE: save data either in main storage or on disk to be used at a later time by this subsystem or related application subsystems.
- FETCH: retrieve stored data from main storage or disk.
- UNSTORE: free stored data from main storage or disk when the information is no longer needed by an application subsystem.

Stored data may consist of counters, switches, messages, subsystem parameters, print lines or any information which a subsystem may wish to save and/or retrieve. Each Store/Fetch data string is saved and retrieved using a unique application-assigned key. Data string processing is described in Chapter 2 and string access service routines are detailed in Chapter 3. Store/Fetch installation is in Chapter 4.

1.1.1 Modular Programming Applications

The Store/Fetch facility is quite useful in a modular programming environment. In this type of environment, each application subsystem typically performs one specific function. A group of related subsystems may work together to perform a major task, with each subsystem working on a specific portion of the task. Results are communicated via the Store/Fetch facility. For example, an application subsystem is activated by an input message from a terminal, performs initial processing and saves the input message and/or resultant data for the next related subsystem. The first subsystem makes some entries in a table, accesses files, sets switches, and updates counters, based on the content of the input message. A second subsystem then analyzes the findings of the first subsystem and continues the processing to produce a second intermediate result. A third subsystem analyzes the output of the second subsystem, updates files, and produces a final output message for the terminal.

Without Store/Fetch, this process would most likely be implemented through the use of the File Handler to save intermediate results. The overhead for CALLs to File Handler Service Routines would then be included in the processing time of the application subsystems. Developing logic to interface with the File Handler to perform all of these functions would add to the implementation time and to the storage requirements for each application subsystem. The Store/Fetch facility simplifies application development.

Figure 1 illustrates the logic flow of a message which is processed in turn by Subsystems A, B and C through the use of the Store/Fetch facility. The subsystems share switches, counters and data saved and retrieved via Store/Fetch.

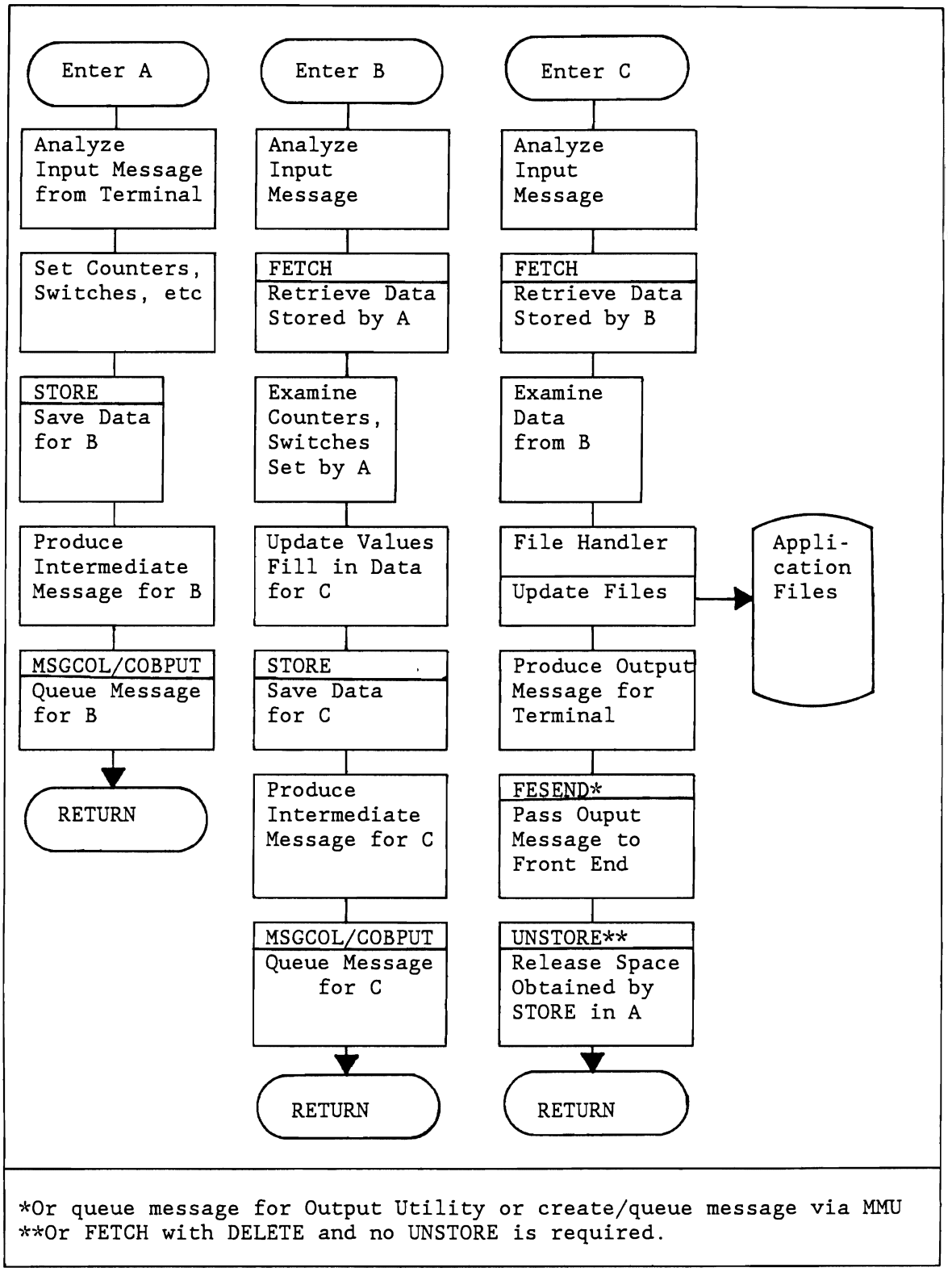

Figure 1. Using the Store/Fetch Facility

1.1.2 Conversational Subsystem Applications

Subsystems involved in conversational processing may use Store/Fetch to preserve the conversational environment. The first and subsequent input messages and/or related data may be saved through the use of STORE and retrieved through the use of FETCH until the conversation is complete. A more thorough discussion of implementation of conversations via Store/Fetch is found in the chapter on conversational subsystems in the Intercomm COBOL, PL/1, and Assembler Language Programmers Guides.

1.1.3 Other On-Line Applications

Subsystems which develop any type of data to be used by other subsystems may save and retrieve the data through Store/Fetch. The data may be transient in nature (perhaps cumulative totals for one Intercomm execution), and it would therefore be necessary only to preserve it from Intercomm Startup to Closedown. Rather than encounter the overhead of preserving the data on a data set, it may be defined as Store/Fetch data to be kept in (24-Amode) core.

A subsystem may create an output message and may wish to save it for later transmission. There may be a certain set of conditions which must be met before the message should be transmitted. For example, a subsystem might expect to receive a feedback message from the Front-End (via Front-End Control Message facility) to indicate one type of terminal output is complete before it would wish to transmit this message. Store/Fetch may be used to save a terminal-oriented message for later transmission, or to gather data to produce a printed report.

Under Release 10, when multiple data strings are created by an application, a Table Facility is available for storing them in core above the 16M line. Strings in a table may be retrieved later by the same or another subsystem under one Intercomm execution.

1.2 THE MULTIREGION ENVIRONMENT

Store/Fetch service routines are eligible for Link Pack Area residence. Thus, no additional storage overhead per region is encountered in the Multiregion environment. All I/O associated with Store/Fetch is overlapped with all processing regions as well as within the particular Intercomm region.

Store/Fetch data sets must be unique to each region. The Multiregion concept of independent decentralized application-oriented regions is maintained. Store/Fetch data may not be shared across regions (except for permanent strings on a read-only data set).

1.3 BATCH MODE OPERATIONS

The Store/Fetch facility may also be used by batch programs to save and later retrieve data within the same or a different program. Off-line "utility" programs might create/update Store/Fetch data for on-line program access. See Chapter 5.

1.4 STORE/FETCH UTILITY

A utility is provided to dump/restore, copy, or print (data strings on) a Store/Fetch data set. This utility is described in Chapter 6.

Chapter 2

PROCESSING CONCEPTS

2.1 STORE/FETCH DATA STRINGS

Store/Fetch operates upon logical entities defined as data strings. Each data string is given a unique user-defined name called a key. The key is specified in the parameter list passed to the Store/Fetch service routines. Keys beginning with DATA and ISYS are reserved for Intercomm use.

2.1.1 Data String Types

Data strings may be of any length and are defined as being either transient, semipermanent or permanent. Figure 2 summarizes the differences among the three types of data strings. Transient data strings may be core resident or disk resident, based on a STORE option. Semipermanent and permanent data strings reside only on disk.

The maximum main storage allotment for Store/Fetch transient data strings stored in core is a user-specified SPALIST macro parameter. When that value is exceeded, transient strings not recently accessed are flushed to their disk data set. A subsequent FETCH, or STORE with update, of such transient data causes it to again reside in core. Thus, retrieval of frequently accessed transient data strings is not I/O bound.

Similarly, in the event of a Resource Management low core condition, transient strings in core are flushed to their disk data set (when disk space is available) to alleviate the condition.

Data String Type	Availability	Storage Medium
Transient	From system startup/restart until normal or abnormal closedown.	Core or Disk
Semipermanent	Across system restart.	Disk
Permanent	Through each system execution until specifically UNSTOREd.	Disk

Figure 2. Store/Fetch Data String Types

NOTE: Transient data strings only exist for the execution of Intercomm within which they are created. Transient data strings that are on disk are deleted during system initialization of the next execution. Semipermanent strings are also deleted during system initialization when Intercomm is executed with PARM=STARTUP, but are not deleted if PARM=RESTART or RESTRNL.

2.1.2 Accessing Data Strings

Data strings are created through the use of the STORE function. They are placed in dynamic storage or written to a Store/Fetch data set as requested by the user. Data strings are retrieved through the use of the FETCH function. The retrieved strings are moved by Store/Fetch into an area provided by the calling subsystem. Assembler subsystems may request FETCH to obtain the area, but the subsystems are responsible for freeing it. The space in dynamic core or on disk occupied by a given data string may be released through the use of the UNSTORE function. An UNSTORE applied to a disk data string causes the space to be made available for subsequent STOREs of other new strings. STORE, FETCH and UNSTORE functions are performed by service routines. Application subsystems CALL the routines to perform these functions as follows:

Function	Service Routine
STORE	INTSTORE
FETCH	INTFETCH
UNSTORE	INTUNSTO

Figure 3. Store/Fetch Service Routines

Any stored data string, whether in storage or on disk, can also be deleted via the FETCH with DELETE option of the facility. A string can also be renamed, or even moved to a different data set. An update with a length change is also possible.

2.2 STORE/FETCH DATA SETS

Up to ten (10) Store/Fetch data sets may be used in one Intercomm environment (execution). The ddname convention is INTSTORn, where n may be 0 through 9. Application programs may specify use of a specific data set in their service routine requests, or the default data set, INTSTOR0, will be used.

Each data set used by Store/Fetch is a preformatted keyed BDAM file with the Extended Search option. The records are fixed-length and unblocked. Record length should be larger than most of the data string lengths the user intends to save on the file. Data strings longer than the stated record length are segmented and chained by Store/Fetch: shorter strings are padded with binary zeros. Management of the variable-length data strings is transparent to the application program.

The number of Store/Fetch data sets to define is an installation-dependent consideration. Several Store/Fetch data sets may be created to be accessed on-line for the following reasons:

- Different application subsystems executing concurrently and using Store/Fetch data sets may have quite different processing requirements.
- Record lengths required may vary greatly.
- Performance may be degraded if many subsystems contend for access to one data set.

2.3 SUBSYSTEM DESIGN CONSIDERATIONS

Data strings are accessed by user-specified keys up to 48 characters in length. The key must be unique for the data set defined at access time. If a data string with a given key may be updated by more than one subsystem, or by another thread of the same reentrant subsystem, subsystem logic must include a method of ensuring exclusive use of the string at that time via Store/Fetch control parameters. A data string to be created may not have a duplicate key of an already existing data string for the specified data set.

The following conventions are used by Intercomm system programs for Store/Fetch data string names:

- ISYS: for internal system use
- DATA: for Data Entry facility

Users must not use these four-character combinations as the first four characters of the data string key for their applications.

Suggestions for optimization of Store/Fetch usage are given in Appendix A.

2.3.1 Creating Data Strings

Application design should ensure that each stored data string has a unique key for the Store/Fetch data set requested. Store/Fetch, when processing a create function (via STORE with the Add option), does not check if a duplicate key (data string name) exists for the requested data set. If more than one subsystem, or another thread of the same reentrant subsystem, could create a data string with an identical key to an existing data string then the next FETCH for the newly created data string could have unpredictable results (that is, the FETCH might get the original data string with the same name, not the one just created).

To minimize Store/Fetch overhead for data string creation, the responsibility for ensuring unique keys is placed on the user. Each subsystem (thread) might enqueue upon the resource (data string name) to obtain exclusive control using the INTENQ/INTDEQ macros. High-level language subsystems must call a user-written Assembler Language service routine which issues the macros. With the enqueue in effect, others wishing to create the same string will be delayed in their processing. The subsystem in control will then issue a FETCH for the string to see if a string with this key has already been created. A Record Not Found return code indicates that no duplicate key exists. If it does exist, the subsystem can delete the string via UNSTORE. The subsystem will then create the string and call the service routine to issue a dequeue on the resource. Or, if a string already exists for this key, the subsystem can update the string returned by the FETCH. The subsystem will then STORE the updated data string and call the service routine for the dequeue function. See also Section 2.3.2 on updating strings.

Again, please note that this enqueuing technique is necessary only if other circumstances do not ensure that multiple subsystems (or multiple threads of the same reentrant subsystem) will not attempt to create the same data string concurrently.

2.3.2 Updating Data Strings

If a data string may be updated by more than one subsystem, or by another thread of the same reentrant subsystem, data integrity must be preserved by obtaining exclusive control during the updating process. This is accomplished by the Store/Fetch service routines based upon a user request (via the parameters passed). The service routines then issue an enqueue before the FETCH and a dequeue after the STORE.

2.3.3 Renaming Data Strings

A data string is renamed by storing it with a new key. For example, a data string contains information (such as counters) which the user wants to preserve as permanent data. However, that string is fetched and later stored in core as transient data for ready access, so it can be modified during processing. In this case, the stored transient data should be renamed to assure integrity of the original. This is to safeguard against the modified transient data string being flushed back to its disk data set (under the key of the original permanent data string) in the event of a low-core condition. The original data would be destroyed, in spite of its "permanence", if the modified data had the same key. That is, the original permanent string would be overlaid by the transient string in the flush processing. This situation could be avoided by renaming the modified transient data, as follows:

- FETCH original (Key 1): retrieve the data string,
- Modify the original data string and STORE in core (Key 2).

If a low-core condition should occur, the modified data string would be written to disk at Key 2 location as transient data. Key 1 location would still have the original permanent data. Note that periodic updates of the permanent string, using the modified transient data, could be a method of checkpointing cumulative subsystem data, such as transaction totals. The permanent data could later be accessed in batch mode for activity reports, etc.

2.4 STORE/FETCH DATA SET RECOVERY

Recovery of Store/Fetch strings written to a Store/Fetch data set may only be accomplished through installation of the Intercomm File Recovery special feature. The Backout-on-the-Fly facility of that feature may also be used.

A Store/Fetch data set containing transient strings should not be subject to file recovery. Excessive file update logging may take place when transient strings in core are flushed to disk during low core conditions. File recovery precedes Store/Fetch initialization at system initialization. File recovery performed for transient strings is unnecessary overhead because of their subsequent deletion.

Use of the recovery feature should be confined to one user program-accessed Store/Fetch data set to minimize processing overhead.

2.4.1 Backout-on-the-Fly

The File Recovery option to back out updates in the event of a subsystem failure may be implemented for user program-accessed Store/Fetch data sets. All types of data strings may be subject to Backout-on-the-Fly (BOF), with the following exceptions:

- Transient strings created with a "keep in core" request (SFCW=ØCTA) are not recoverable.
- If a transient string created with a write to disk (SFCW=ØWTA) is later updated/replaced using a "keep/hold in core" request (SFCW=ØCTU,ØHTU,ØCTR,ØHTR), that request is also not recoverable.

The SFCW values and their meaning are described in Chapter 3.

2.4.2 Subsystem Design

It is advisable that subsystem coding be organized in such a way that all processing is done before an updated string is returned to Store/Fetch or a new string is added. It is also recommended that FETCH with the Delete option not be used for a Store/Fetch data set with File Recovery options defined; instead use FETCH, followed by an UNSTORE just before returning control to the Subsystem Controller.

Chapter 3

STORE/FETCH SERVICE ROUTINES

3.1 INTRODUCTION

Application programs call Store/Fetch service routines using the standard conventions of the language in use. Reentrant COBOL subsystems and PL/1 subsystems call the service routines via the interface routines COBREENT and PMIPL1, respectively. PL/1 and Assembler programs may call the routines directly.

3.2 STORE/FETCH PARAMETERS

The parameters passed to the Store/Fetch routines qualify the action to be performed. With the exception of the disk data set ddname, all parameters specify areas of storage unique to a message thread in progress and must be defined in the dynamic working storage of the subsystem. The areas of storage associated with Store/Fetch service routines and summarized in Figure 4 are:

- Store/Fetch Control Word (SFCW), a four-byte (fullword aligned) area specifying service routine options requested by the call, and containing a return code indicating status of the function after the call.

- Key, which must be left-justified and may be alphanumeric character data and/or a hexadecimal value, though treated as a binary value by the facility. The length of the Key area must be big enough to hold the number of bytes specified for the key length. Do not use low-order blank padding. The key may not start with X'FF' (high-values). Keys beginning with the words DATA and ISYS are reserved for Intercomm internal use.

- Key Length, which must be a binary halfword value on a halfword boundary. A key may be from 1 to 48 bytes long. This allows the use of qualified data set names, for example, as keys.

WARNING: If the key length passed to STORE is greater than the actual length of nonblank key bytes, a Space Not Found condition may happen due to randomizing.

- Data String Area, which contains the data to be stored, or into which data will be moved for a FETCH. This area belongs to the caller, and will not be freed by the facility.

- Data String Length, which must be a binary halfword value, on a halfword boundary, from 1 to 32,767. For a STORE, this is the length of the string to be stored from the data area, while for a FETCH this must be initialized to the length of the data area itself. The actual data string length is always returned in this field from a FETCH request.

- DDname, which is the eight-byte ddname field and must contain INTSTORn to define a Store/Fetch data set. The n is a character value from 0 (zero) to 9. If there are not enough dummy records on the requested data set within the specified search limits to STORE a data string, the data string is not stored, and a Space Not Found error code is returned. If the calling program may be loaded above the 16M line under Release 10, the ddname field must be in the caller's dynamic storage area, except if the calling program is VS COBOL II (may be in Working Storage).

If the ddname parameter is omitted in the service routine call, the default data set (INTSTOR0) is selected. If INTSTOR0 is not available for a disk data string request, an appropriate error code is returned.

Assembler Language and PL/1 direct call programs may specify 0 (zero) as the data-area address for a FETCH. In this case, the facility acquires the dynamic storage for the string and passes back the address (in the data-area address field of the parameter list) and length (in the halfword data-length field) of the area (string). The user program is then responsible for freeing this storage area. The area is always in 24-Amode storage.

Parameter	Meaning
REENTSBSname	The label of a halfword (aligned) binary value specifying the REENTSBS code for the service routine to be called (COBOL and PL/1 via PMIPL1)
SFCWname	The label of the Store/Fetch Control Word (fullword aligned)
KEYname	The label of the area containing the data string key
KLENname	The label of the halfword (aligned) binary value specifying the key length
DATAname	The label of the data string area to be Stored, or the data string area for data to be Fetched
DLENname	The label of the halfword (aligned) binary value specifying data string length
DDname	The label of the area (eight bytes) containing the ddname of the Store/Fetch data set

Figure 4. Parameter Summary for Store/Fetch Calls

Figures 5, 6, and 7 summarize the conventions for calls to Store/Fetch service routines. Assembler Language and PL/1 programs calling Store/Fetch directly also use the service routine names listed in Figure 5. For PL/1 calls, these names are also in COPY member PLIENTRY or may be declared individually with ENTRY OPTIONS (ASM INTER). (For details of which parameters apply to which calls, see Figures 9, 12, and 15.)

Service Routine	REENTSBS Code	ICOMSBS name (COBOL)	PENTRY name (PL/1 via PMIPL1)
INTFETCH	39	INTFETCH	INTFETCH
INTSTORE	43	INTSTORE	INTSTORE
INTUNSTO	47	INTUNSTO	INTUNSTO

Figure 5. REENTSBS Codes (COBOL and PL/1)

Byte	Purpose
1	Return code from Service Routine
2	Processing Request Character
3	Data Type (Transient, Semipermanent or Permanent)
4	STORE-related options

Figure 6. SFCW Format

3.3 STORE/FETCH RETURN CODES

The return code is placed in byte 1 of the SFCW by each of the Store/Fetch service routines. It is a numeric character value. The character zero always indicates successful completion of the function and options requested. The return code placed in the SFCW is also multiplied by 4 and returned in Register 15, in binary form. Return codes from INTSTORE, INTFETCH and INTUNSTO are detailed in Figures 10, 13, and 16, respectively.

Byte	Value		
1	Reserved for return code indicating status of call.		
2	Processing requests are coded as follows:		
	FETCH	X	Enqueue before retrieving data string. This must be followed by a STORE or an UNSTORE with a dequeue request, from the same thread.
		D	Retrieve, then delete original data string.
		K	Retrieve and keep original data string.
	STORE	C	Keep in core (valid for transient data only).
		W	Do not keep in core, but write string to disk data set specified (valid for any data type).
		X	Dequeue after saving updated data string on disk. (If data type is transient, it will be written to disk).
		H	Dequeue after saving updated transient string in main storage.
	UNSTORE	X	Dequeue after deleting data string. Dequeue is automatic, even if return code is non-zero.
		blank	If X not requested, leave blank.
3	Data Type -- specify one of the following:		
	T	Transient Data	
	S	Semipermanent Data	
	P	Permanent Data	
4	STORE-related options. Must be coded for STORE:		
	A	Add a new data string	
	R	Replace an existing string of the same length	
	U	Update with length change	

Figure 7. Service Routine Options Specified via SFCW

3.4 INVOKING THE STORE FUNCTION

INTSTORE is invoked by a standard subroutine CALL. The SFCW must be initialized prior to the CALL to specify STORE options; after the CALL, the status of the operation is reflected by a return code in byte 1 of the SFCW.

The data string to be stored may be one of the following:

- a new data string (specified by A in byte 4 of the SFCW)
- an updated data string with no length change (specified by R in byte 4 of the SFCW)
- an updated data string with length change (specified by U in byte 4 of the SFCW)

No test is made for a duplicate (existing key) on the Add (A) option. An existing string is not necessarily replaced; the next FETCH will have unpredictable results (see Section 2.3.1).

Note that if a STORE with dequeue is requested and a processing error occurs (non-zero return code) the dequeue is not honored, except for an I/O error.

SFCW options are summarized in Figure 8, parameters for the CALL are illustrated in Figure 9, and the INTSTORE return codes are listed in Figure 10.

Function Desired	SFCW Setting (Character Values)			
	Byte 1	Byte 2	Byte 3	Byte 4
STORE a new string in core	blank	C	T	A
STORE an updated string in core	blank	C	T	R/U
STORE a new string on disk	blank	W	T/S/P	A
STORE an updated string on disk	blank	W	T/S/P	R/U
STORE an updated transient string in core, then dequeue	blank	H	T	R/U
STORE an updated string on disk, then dequeue	blank	X	T/S/P	R/U

Figure 8. SFCW Specification for INTSTORE

```
COBOL:
        CALL 'COBREENT' USING REENTSBSname,SFCWname,KEYname,
             KLENname,DATAname,DLENname[,DDname].
-------------------------------------------------------------------

PL/1:
        CALL PMIPL1 (REENTSBSname,SFCWname,KEYname,KLENname,
                     DATAname,DLENname[,DDname]);

        If INTSTORE is called directly, omit the REENTSBSname
        parameter.
-------------------------------------------------------------------

Assembler:

        CALL INTSTORE,(SFCWname,KEYname,KLENname,DATAname,
                       DLENname[,DDname]),VL,MF=(E,list)
```

Figure 9. Parameters for INTSTORE (DDname is optional)

Code	Meaning
0	Successful STORE.
1	I/O Error during write to disk.
2	Record Not Found: R/U option had been specified, and the key of the updated string does not exist for the specified data set.
3	Dequeue requested for key not previously enqueued upon (X/H option) but string successfully updated/replaced.
4	Space Not Found: A/U option had been specified, and no dummy records available within search limits on the specified data set.
6	Data Length Error: R option had been specified, but length of updated string differs form original.
7	Key Error: key field contains binary zeros or hexadecimal 'FF' in first byte, or key length value not between 1 and 48.
8	Specification Error: invalid options combination in SFCW or required option missing, or parameter list not long enough for function requested, or incompatible data string type for key requested.
9	Specified/default data set not available, or invalid DDname specified.

Figure 10. Return Codes for INTSTORE

3.5 INVOKING THE FETCH FUNCTION

INTFETCH is invoked by a standard subroutine CALL. The SFCW must be initialized prior to the CALL to specify FETCH options; after the CALL, byte 1 of the SFCW indicates the status of the operation.

The data area length specified in the parameter list to INTFETCH must be large enough to hold the data string to be fetched. If not, the length required is returned in the data length field, along with a Data Length Error return code. After a successful FETCH, the data string is in the area specified and the actual length is in the data length field. The data length field must then be reinitialized to the original maximum data area length for subsequent FETCH requests.

If a FETCH with enqueue is requested and a processing error occurs (non-zero return code), the enqueue request has not been honored.

SFCW options are summarized in Figure 11, parameters for the CALL are shown in Figure 12, and the INTFETCH return codes are listed in Figure 13.

Function Desired	SFCW Setting (Character Values)			
	Byte 1	Byte 2	Byte 3	Byte 4
FETCH (retrieve) a data string	blank	K	T/S/P	blank
FETCH a string for update with enqueue	blank	X	T/S/P	blank
FETCH string for informational use, delete original	blank	D	T/S/P	blank

Figure 11. SFCW Specification for INTFETCH

Note that a FETCH with the Delete Original option (C'D' in byte 2 of the SFCW) may be used instead of the 2-step FETCH, followed by an UNSTORE, process within the same thread.

```
COBOL:
        CALL 'COBREENT' USING REENTSBSname,SFCWname,KEYname,
             KLENname,DATAname,DLENname[,DDname].
-------------------------------------------------------------------

PL/1:
        CALL PMIPL1 (REENTSBSname,SFCWname,KEYname,KLENname,
                     DATAname,DLENname[,DDname]);

        Or, if INTFETCH is called directly, omit the REENTSBSname
        parameter, and DATAname may be 0 (zero-refer to Section 3.2).
-------------------------------------------------------------------

Assembler:

        CALL INTFETCH,(SFCWname,KEYname,KLENname,DATAname,
                       DLENname[,DDname]),VL,MF=(E,list)

        DATAname may be 0 (zero-refer to Section 3.2).
```

Figure 12. Parameters for INTFETCH (DDname is optional)

Code	Meaning
0	Successful FETCH.
1	I/O Error during retrieval from disk.
2	Record Not Found: key does not exist for data set specified.
4	Segmented string retrieved from disk contains invalid data (last update not successful due to system failure or I/O error before completed). Delete completed if requested.
6	Data Length Error: value not between 1 and 32,767 or data area not big enough for string. NOTE: required length will always be returned in data length field.
7	Key Error: key field contains binary zeros or hexadecimal 'FF' in first byte, or key length value not between 1 and 48.
8	Specification Error: invalid options combination in SFCW or required option missing, or parameter list not long enough for function requested, or incompatible data string type for key requested.
9	Specified/default data set not available, or invalid DDname specified.

Figure 13. Return Codes for INTFETCH

3.6 INVOKING THE UNSTORE FUNCTION

INTUNSTO is invoked by a standard subroutine CALL. The SFCW must be initialized prior to the CALL to specify UNSTORE options; after the CALL, status of the request is indicated in byte 1 of the SFCW.

Figure 14 lists SFCW options; Figure 15 lists the service routine parameters; Figure 16 lists the return codes from INTUNSTO.

Function Desired	SFCW Setting (Character Values)			
	Byte 1	Byte 2	Byte 3	Byte 4
UNSTORE (delete) an existing transient string	blank	blank	T	blank
UNSTORE (delete) an existing string from disk	blank	blank	S or P	blank
UNSTORE (delete) an existing string, then dequeue	blank	X	T/S/P	blank

Figure 14. SFCW Specification for INTUNSTO

COBOL:

```
CALL 'COBREENT' USING REENTSBSname,SFCWname,KEYname,
     KLENname[,DDname].
```

PL/1:

```
CALL PMIPL1 (REENTSBSname,SFCWname,KEYname,KLENname,
            [,DDname]);
```

If INTUNSTO is called directly, omit the REENTSBSname parameter.

Assembler:

```
CALL INTUNSTO,(SFCWname,KEYname,KLENname[,DDname]),
              VL,MF=(E,list)
```

Figure 15. Parameters for INTUNSTO (DDname is optional)

Code	Meaning
0	Successful UNSTORE
1	I/O Error
2	Record Not Found
3	Key not previously enqueued (X option only)
7	Key Error
8	Specification Error: incorrect data type for key requested
9	File Not Available/DDname Error

Figure 16. Return Codes for INTUNSTO (see Figure 10 for further details)

Chapter 4

INSTALLATION

4.1 INTRODUCTION

Installation and use of Store/Fetch requires the following preparatory steps:

- Store/Fetch Data Set(s) Creation
- SPA Specification
- Intercomm Linkedit
- JCL for Execution

Some Intercomm facilities use Store/Fetch data sets, as described in Section 4.6. Section 4.7 describes statistics on Store/Fetch processing. Tuning and usage optimization suggestions are given in Appendix A.

4.2 CREATING A STORE/FETCH DATA SET

A Store/Fetch data set is a preformatted keyed BDAM file with fixed-length unblocked records. The key length must be 52 bytes to contain a user-defined key up to 48 bytes in length (left-justified and padded with binary zeros by the facility), plus an internal chaining sequence number for segmented data strings. The record length (BLKSIZE) should be larger than the common data string lengths the user intends to STORE on the data set, plus a 24-byte header appended by Store/Fetch. The header is not accessible to the user's processing modules. Therefore space for it does not have to be provided in user programs. The block size chosen must be a multiple of 8.

The user may use his own program to create the data set with dummy records or may use the Intercomm utility KEYCREAT. In either case, the following DCB parameters are required on the input DD card in addition to the DSN, DISP, UNIT, SPACE and VOL=SER parameters:

```
DCB=(DSORG=DA,RECFM=F,BLKSIZE=nnn,KEYLEN=52)
```

Do not specify OPTCD or LIMCT on the JCL used to create a Store/Fetch data set.

The input PARM to the KEYCREAT program specifies the number of blocks desired in the data set. If zero is coded, or the PARM is omitted, dummy records are created to the limits of the primary SPACE allocation only. The input ddname for KEYCREAT must be INTKEYFL.

Sample execution JCL for KEYCREAT is shown in Figure 17; additional documentation on KEYCREAT is contained in the Operating Reference Manual.

```
//          EXEC PGM=KEYCREAT(,PARM=bbb)
//STEPLIB   DD   DSN=libraryname,DISP=SHR
//INTKEYFL  DD   DSN=name,DISP=(NEW,CATLG,DELETE),
//               SPACE=({nnn,(bbb,e)}),UNIT=uuuu,VOL=SER=vvvvvv,
                        {TRK,(i,e)  }
                        {CYL,(i,e)  }
//               DCB=(DSORG=DA,RECFM=F,BLKSIZE=nnn,KEYLEN=52)
```

where:
nnn is record length
bbb is number of records to be created in the file
i is initial space allocation
e is secondary space allocation (if desired).

Figure 17. KEYCREAT JCL Example.

4.3 SPA SPECIFICATION

The maximum core to be occupied by Store/Fetch transient data strings is specified in K (Kilobytes) by the SPALIST macro STOCORE parameter. The default value is 5K or 5120 bytes. As with any other SPALIST specification, the SPA module (INTSPA) must be reassembled following a change in any parameter specification.

Under Release 10, the current STOCORE value may be displayed via the TALY$SU command, and dynamically changed for the current Intercomm execution via the SCTL command (to reduce excessive flushing).

4.4 LINKEDIT REQUIREMENTS

Required linkedit control cards for Store/Fetch are generated by specification of STORFCH=YES on the ICOMLINK macro. Figure 18 lists linkedit requirements. INTSTORF may be resident in the Link Pack Area (see Operating Reference Manual for installation).

```
      .
      .
      .
INCLUDE SYSLIB(STOSTART)          Initialization
INCLUDE SYSLIB(INTSTORF)          Main Processing
      .
      .
      .
```

Figure 18. Linkedit Requirements.

4.5 JCL FOR EXECUTION

To access a Store/Fetch data set on-line, the DD statement must include the following:

```
//INTSTORn    DD    DSN=name,DISP=SHR,
//    DCB=(DSORG=DA,OPTCD=EF,LIMCT=n)
```

Because the Extended Search option and relative track addressing are used, the LIMCT=number of tracks (to be searched) must be coded. While the value chosen for LIMCT depends on the size of the Store/Fetch data set, its block size, and the amount of data it contains, a value of at least 2 is advisable. The 'n' in the ddname must be a character value from 0 to 9. The default data set is that accessed by the ddname INTSTOR0.

For on-line execution efficiency, it is advisable to code a FAR (File Attribute Record) statement for each INTSTOR data set, as follows:

```
INTSTORn,ICOMBDAMXCTRL
```

Refer to the Operating Reference Manual for further details.

If a READONLY attribute is defined for a Store/Fetch data set, it is not effective until after Intercomm startup processing is complete. During startup, Store/Fetch initialization requires opening of the data set for input/output processing (to delete transient and semi-permanent strings depending on startup/restart mode - see Section 2.1.1). Therefore, read-only password protection may not be specified for a Store/Fetch data set. A READONLY FAR should not be defined for the MMU maps Store/Fetch data set if the LMAP command may be used under Release 10 (see Message Mapping Utilities).

FARs for File Recovery processing may be coded for user Store/Fetch data sets (see File Recovery Users Guide).

4.6 INTERCOMM STORE/FETCH REQUIREMENTS

When calculating the number of Store/Fetch data sets to be available, or shared, for user application subsystem processing, consider the following system uses of Store/Fetch:

- Message Mapping Utilities (MMU); 2 data sets:
  - Loaded Map Definitions (dedicated permanent strings)
  - Intermediate output mapping form (transient strings-deleted after MAPEND or MAPURGE processing)
- Autogen Facility

- Data Entry Facility
- BTAM Simulator SIM3270 processing (INTSTOR9)-see Operating Reference Manual.

4.7 STATISTICS ON STORE/FETCH USAGE

SAM (System Accounting and Measurement) statistics on application subsystem calls to Store/Fetch (including via MMU processing) may be gathered, including whether the processing is for a string in core or on disk, and for an update with length change. See the Operating Reference Manual for installation and reporting details.

The System Tuning Statistics give information on Store/Fetch strings and flushes across all Store/Fetch data sets for Release 9, or broken down by data set (with totals for all data sets) under Release 10. See the Operating Reference Manual for further details.

Chapter 5

BATCH MODE OPERATION

5.1 USAGE

Store/Fetch may be used in Batch mode exactly as for on-line execution. It may be used to create, update, or delete a permanent string on an INTSTOR disk data set used in on-line execution, or may be used with one or more unique Store/Fetch data sets not referenced online.

The parameters for calls to Store/Fetch routines are coded exactly as for on-line execution, except omit the REENTSBSname (always directly call the desired entry point if a hi-level language used for the batch program). In Batch mode, the first call will trigger Store/Fetch initialization, which is normally performed during Intercomm startup in on-line mode. Initialization will delete all transient and semipermanent strings on INTSTORn data sets defined in the Batch mode JCL. Therefore, it is not advisable to execute a Batch mode program accessing on-line data strings while Intercomm is executing. Nor should a Batch mode job accessing on-line Store/Fetch data sets be executed between Intercomm restarts.

To the linkedit for Batch mode jobs add the following:

```
INCLUDE    SYSLIB(INTSTORF,STOSTART)
INCLUDE    SYSLIB(IXFHND00,IXFHND01)
INCLUDE    SYSLIB(BATCHPAK)
```

Figure 19. Batch Mode Store/Fetch Linkedit

Note that BATCHPAK contains SPA and SPAEXT Csects which use the defaults for all parameters; it is not necessary to include an on-line version of INTSPA.

To execute in Batch mode, DD statements must be present in the JCL for each INTSTOR data set to be accessed. The DD statement must be coded exactly as for on-line execution. Particularly, the LIMCT parameter must be exactly the same for Store/Fetch data sets which are also defined in the execution JCL of any other job (batch or online).

Chapter 6

OFF-LINE STORE/FETCH UTILITY

6.1 INTRODUCTION

An off-line utility, SFDMPRST, is provided to perform the following operations:

- Dump a Store/Fetch data set to tape (for backup)
- Restore (load) a Store/Fetch Data set from tape backup
- Copy (merge) the contents of one Store/Fetch data set to another (to allow for expansion of the data set)
- Print non-deleted strings on a Store/Fetch disk data set in EBCDIC and hexadecimal (for reference or analysis).

Installation of the utility may require linkediting it onto MODLIB with the modules given in Chapter 5 of this manual for batch mode operation. Include SFDMPRST ahead of the listed INCLUDE statements. The recommended load module name is DUMPREST. A prelinked load module by that name is provided on the released MODREL.

6.2 EXECUTION SYNTAX

The syntax of the EXEC statement is as follows:

```
//        EXEC  PGM=DUMPREST,PARM='{DUMP}[,{S}]'
                                   {LOAD}  {P}
                                   {COPY}
                                   {LIST}
```

where:

DUMP specifies that a Store/Fetch data set is to be dumped to a tape file.

LOAD specifies that a Store/Fetch data set is to be loaded from a tape previously created via DUMP.

COPY specifies that the contents of one Store/Fetch data set are to be copied to another Store/Fetch data set.

LIST specifies that non-deleted non-transient strings on a Store/Fetch data set on disk are to be printed.

S specifies that both semipermanent and permanent strings are to be included.

P specifies that only permanent strings are to be included (default).

The DD statement for the input disk data set (for COPY, DUMP, and LIST) must be labeled INTSTOR0, and the DD statement for the output disk data set (for LOAD or COPY) must be labeled INTSTOR1. These labels must be used no matter what INTSTOR number is being used for these data sets in the Intercomm execution JCL. In addition, there must be a DD statement labelled SFDMPRST, referencing the same input data set as INTSTOR0; it is not required for LOAD operations. The DCB parameters are as usual, including the LIMCT subparameter. The output disk data set must be created as explained in Section 4.2 prior to running the utility.

The DD statement for the input or output tape data set (for DUMP or LOAD) must be labeled TAPE. The DCB parameters should be specified as follows:

```
DCB=(RECFM=VBS,BLKSIZE=blksize,LRECL=dslrecl+92)
```

where blksize is any convenient block size, and dslrecl is at least the length of the average string on the Store/Fetch data set (+92).

The DD statement for the listing data set (for LIST) must be labeled PRINTOUT. The DCB parameters should be specified as follows:

```
DCB=(RECFM=FBA,LRECL=133,BLKSIZE=133*n)
```

where n is any convenient blocking factor.

In addition, two SYSOUT=A DD statements are required, for SYSUDUMP and for SYSPRINT. There is no SYSIN for the program.

6.3 EXAMPLES

To copy or merge a Store/Fetch data set (including both semipermanent and permanent strings) to a larger Store/Fetch data set, the following JCL might be used:

```
//          EXEC  PGM=DUMPREST,PARM='COPY,S'
//STEPLIB   DD    DSN=INT.MODREL,DISP=SHR
//SFDMPRST  DD    DSN=INT.STORFET0,DISP=SHR,
//                DCB=(DSORG=DA,OPTCD=EF,LIMCT=n)
//INTSTOR0  DD    DSN=INT.STORFET0,DISP=SHR,
//                DCB=(DSORG=DA,OPTCD=EF,LIMCT=n)
//INTSTOR1  DD    DSN=INT.STORFET1,DISP=SHR,
//                DCB=(DSORG=DA,OPTCD=EF,LIMCT=2)
//SYSUDUMP  DD    SYSOUT=A
//SYSPRINT  DD    SYSOUT=A,DCB=(LRECL=137,BLKSIZE=141,RECFM=VA)
/*
```

In this example, INT.STORFET1 has been previously created via KEYCREAT.

NOTE: when merging two Store/Fetch data sets, duplicate keys are not checked for and will produce unpredictable results if they exist on the output data set. To merge MMU map data sets, use the MMU LOADMAP utility instead (see Message Mapping Utilities) which does check for duplicate keys.

To restore only the permanent strings from a tape backup of a Store/Fetch data set to a new Store/Fetch data set, the following JCL might be used:

```
//          EXEC  PGM=DUMPREST,PARM='LOAD'
//STEPLIB   DD    DSN=INT.MODREL,DISP=SHR
//INTSTOR1  DD    DSN=INT.STORFET1,DISP=SHR,
//                DCB=(DSORG=DA,OPTCD=EF,LIMCT=2)
//TAPE      DD    DSN=INT.SFBACKUP,DISP=OLD,
//                DCB=(RECFM=VBS,LRECL=192,BLKSIZE=1200)
//SYSUDUMP  DD    SYSOUT=A
//SYSPRINT  DD    SYSOUT=A,DCB=(LRECL=137,BLKSIZE=141,RECFM=VA)
/*
```

Again in this example, INT.STORFET1 has already been created via KEYCREAT.

6.4 ERROR CONDITIONS

Error messages and abends due to error conditions on output are documented in Messages and Codes.

An input tape error condition during LOAD is handled according to the tape error option specified on the JCL.

6.5 END OF JOB I/O STATISTICS

At normal end-of-job, a WTO is put out to the JES Job Log, giving counts of Reads/Writes/Gets/Puts, I/O errors, and records with delete flag on (for the input Store/Fetch data set). Gets and Puts apply only to the input or output tape backups, respectively. Writes apply only to an output Store/Fetch disk data set.

Appendix A

STORE/FETCH TUNING AND OPTIMIZATION

The following suggestions should help to make usage of the Store/Fetch facility as efficient as possible.

- For Store/Fetch data sets:
  - -- Segregate the data sets by data string size.
  - -- The block size for a data set should be larger than most data strings assigned to that data set to reduce chaining.
  - -- At least 30% free space on each data set should be planned to reduce searching.
  - -- Do not specify too high a LIMCT for an INTSTOR data set; a LIMCT of 2 or 3 should be sufficient (see Section 4.5). It is better to increase the size of the data set (see Chapter 6) than to increase the LIMCT.
  - -- A FAR card with the parameter ICOMBDAMXCTRL should be specified to reduce exclusive control overhead in the File Handler (see Section 4.5). However, a READONLY attribute may not be specified.
  - -- Be generous with the size of the Store/Fetch data set used by MMU to temporarily hold flushed output mapping strings.
- For main storage use:
  - -- The total core used is controlled by the STOCORE parameter of the SPALIST macro; a System Tuning Statistic (see Operating Reference Manual) provides the number of flushes: increase STOCORE to reduce flushing to disk. Under Release 10, the STOCORE value may be dynamically changed via the SCTL command, and the current value and the number of flushes may be dynamically displayed via the TALY$SU command, as described in System Control Commands.
  - -- Transient data strings may be stored in core or on disk.
  - -- Those transient data strings which will be needed 'soon' (relative to other transient data strings) should be stored in core.
  - -- Do not place all transient data strings in core just because it is possible, as this will cause excess paging.
  - -- Data strings are flushed on a least recently used (LRU) basis.
  - -- On a low core condition (the storage cushion is released) all strings are flushed.
  - -- INTSTORF is eligible for the Link Pack area.

- For data string handling:
  - -- Do not request exclusive control if it is not needed
  - -- Only enqueue (INTENQ) around ADDs if necessary.
  - -- Avoid FETCHs for nonexistent data strings.
  - -- Assembler Language and PL/1 (using direct CALLs) subsystems may allow FETCH to acquire the main storage for the data string (see Section 3.2).
  - -- Avoid data string updates with length changes. Use REPLACE instead as this reduces storage management and I/O processing. The data string might be created with the maximum required size, and blanks in unused areas. These areas would be filled in by later processing.
  - -- It is more efficient to perform UPDATE or REPLACE rather than to use FETCH with DELETE, followed by ADD (unless the new string has an unrelated function requiring a new key).
  - -- It is more efficient to perform FETCH with DELETE, than to perform FETCH followed by UNSTORE.

INDEX