# INTERCOMM

# FILE RECOVERY USERS GUIDE

ISOGON CORPORATION

330 Seventh Avenue, New York, New York 10001

# File Recovery Users Guide

## Publishing History

| Publication | Date | Remarks |
|---|---|---|
| First Edition | December 1973 | This manual corresponds to Intercomm Release 6.0. |
| Second Edition | July 1982 | This edition corresponds to Intercomm Release 9.0. |
| SPR 237 | April 1989 | Updates for Release 9 and additions for Release 10 (where indicated). |

## PREFACE

Intercomm is a state-of-the-art teleprocessing monitor system executing on the IBM System/370 family of computers and operating under the control of IBM Operating Systems (MVS/370, XA, 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 document presents the concepts and facilities of the coordinated Message Restart/File Recovery facility under Intercomm, and details the coding requirements, implementation and installation specifications for the File Recovery Facility, which is offered as a Special Feature of the Intercomm system.

In this manual, the term file(s), used interchangeably with data base, refers to OS or VS files, specifically ISAM, BDAM and VSAM data sets. Readers seeking specific recovery information on data base management systems should refer to the Intercomm DBMS Users Guide.

As prerequisite reading for this publication, the user is referred to the Intercomm Concepts and Facilities. In addition, the following Intercomm publications should be used in conjunction with this manual:

- Basic System Macros
- Operating Reference Manual

## 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

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

# RESTART/RECOVERY

## 1.1 Introductory Concepts

Intercomm has been designed to anticipate, detect and recover from most error situations without bringing down the entire teleprocessing system. Teleprocessing devices and teleprocessing application programs can and will fail. In most instances following failures, Intercomm continues to operate in a degraded mode, minus the failed components. Alternatively, Intercomm can come down gracefully after a failure by completing all work in progress at the time of failure.

Certain conditions, however, can and will occur that effect immediate termination of all processing in Intercomm. These include power failure, machine failure, data base destruction, and operating system failure. In these and other total failure situations, Intercomm automatically provides for the complete recovery of teleprocessing applications. This recovery includes the restarting of all messages in progress at the time of failure, the recovery of message queues, the recovery of checkpointed data, and the coordinated recovery of files and data bases.

Key points to the Intercomm Restart/Recovery facility are summarized below:

- No User Support Required

  All mechanics of implementing message restart and file recovery are supported by Intercomm-supplied software. User application programs that are to be restarted are no different in format or content than nonrestarted programs. The restart facility is generally transparent to the application programmer. User responsibility in restart is limited to table entries that delineate those programs and terminals for which restart is to be performed. Other table entries specify those programs that may update data base files. These table entries are generally the extent of user responsibility in providing restart capability.

- Fast Restart

  The method of recovery in Intercomm is a rapid "warm" restart. It does not require an off-line program following a failure; live Intercomm is merely restarted. Restarted Intercomm first reads the system logging (journal) file backwards from the point of the failure as far back as necessary for recovery. With the necessary items recovered, live Intercomm starts up, typically only a few minutes later.

- Data Integrity

  The user is afforded complete data integrity following a failure. If a file or its indexes were physically or logically destroyed as part of the failure, the file can be reconstructed from its last backup by Intercomm-supplied software. If the file is intact following a failure, the file is recovered in such a manner that updates in progress at failure time can be reinstated without duplication. The recovery is coordinated for data files, DBMS files, checkpointed system table entries, and message traffic.

- Message Integrity

  All messages and queues are recovered and restarted as appropriate. In certain cases, to ensure data integrity, subsystem messages previously completed must also be restarted to effect necessary interaction with files or data bases.

- Selective Restart

  Users can specify the extent of the restart on a terminal-by-terminal and program-by-program basis. The overhead of restart can be limited to necessary system components, via the restart "always" or "never" specifications. Additionally, a third specification is provided; the restart desirable condition. If restart is desirable, but not mandatory for a program or terminal, then in the course of restart for those mandatory programs and terminals, restart will also be performed for desirable components. However, once restart has completed for all mandatory items, the restart is deemed complete regardless of whether or not desirable components have been fully restarted. Restart will proceed as far backwards through the Intercomm system log as necessary to retrieve all messages for mandatory restarted components. During this read-back process, messages encountered for desirable restart components will be retrieved as appropriate. However, the read-back point of the file is defined as the point necessary to retrieve all mandatory items and is not influenced by desirable items. Those terminals or subsystems for which the restart specification is neither desirable nor mandatory utilize logging as a user option, again on an individual program and terminal basis. Selective logging can reduce system overhead in logging that would otherwise be wasted on noncritical components.

- Automated Restart

  Under Release 10, the same set of JCL can be used for normal startup and restart, with no system operator intervention other than resubmitting the Intercomm job, as described in the Release 10 Operating Reference Manual.

- <u>Coordinated Checkpointing</u>

  As described in the <u>Operating Reference Manual</u>, checkpoints are taken, at a user specified interval, of critical system table entries and counters. A pointer to the checkpoint file data is logged on the system log at checkpoint time. Checkpointed system data, and optional user data, is also restored during message restart.

- <u>Single Log File</u>

  Only a single log is required for Front End, Back End and file image logging. This log can be either tape or disk. Further, a technique for writing on this log has been implemented such that any size record can be written to this log (up to the block size specified for the log file INTERLOG) without devoting buffer space for maximum record sizes. Additionally, log records are blocked for noncritical entries and unblocked for critical entries.

- <u>Log File Recovery</u>

  For tape and/or disk logging, an off-line utility, ICOMFEOF, is provided to determine the end of file after a system failure where the operating system does not properly close the file. For on-line disk logging, a sequential file Flip/Flop facility (x37 abend recovery) is provided to handle logging in smaller file increments than an entire day (or days). Both of these features are described in the <u>Operating Reference Manual</u>.

In the following section, message recovery concepts of the Intercomm system are described. This description is basic to the application of the File Recovery facility. Additionally, in subsequent chapters, all aspects of File Recovery under Intercomm are detailed, including specifications for coding control information. The last chapters present the procedures for implementing the File Recovery facility and the on-line Backout-on-the-Fly facility for immediate file recovery after application program failure. Data Base file recovery is described separately in the <u>DBMS Users Guide</u>.

## 1.2 <u>Message Restart</u>

The concepts of the message restart capability of Intercomm are described in this chapter. An explanation of the elements and logic of message restart is provided, essentially without consideration for file or data base recovery. The next chapter specifically covers file recovery concepts.

Intercomm is an event-driven system whereby work activities are initiated in response to a message. Therefore, the core of Intercomm recovery involves the recovery and/or restarting of appropriate messages. The basis for determining what is required for a particular restart/recovery operation is the Intercomm log. INTERLOG contains entries for all messages that are subject to recovery. It includes entries that make possible a determination of message status at the time of failure. Message status is an important factor in determining the read-back point and is defined by one of the following categories:

- Received and completely processed prior to the last checkpoint;
- Received and completely processed subsequent to the last checkpoint;
- Received and in process at failure;
- Received but processing not started.

The analysis of the message data in the log is performed during restart by reading the log file backwards from the point of failure. A technique of message accounting permits backward reading to proceed only as far as is necessary to retrieve those messages needed for restart.

When messages are recovered from the log they are placed on the queues for their destined subsystems or terminals as the restart phase concludes. Thus, queues are rebuilt, not recovered, following failures.

The restart process is initiated when the RESTART parameter is found in the PARM field of the Intercomm execution (EXEC statement) JCL. This RESTART parameter is the only change required to distinguish a restarted Intercomm run from a normal Intercomm run. When a RESTART is recognized, the restart phase of Intercomm analyzes the restart log and rebuilds the queues. Then the normal mode of Intercomm starts reprocessing messages placed in the queues by restart, while simultaneously receiving and processing messages from the live terminal network (the specification of FIFO queues insures that restarted messages are processed prior to live messages). Optionally, a serial restart facility (described in the Operating Reference Manual) may be used to force processing of all restarted messages prior to accepting new input from the network.

Under Automated Restart (Release 10 only), the RESTART parameter is not used. The system status is recorded in a special data set and the restart requirement is determined from that data set when the Intercomm job is resubmitted.

The entire Intercomm message recovery system is coordinated with the recovery of data files and/or DBMS files, as appropriate. In some circumstances, an original message from a terminal will proliferate messages to many subsystems as a result of a design approach utilizing subsystem-to-subsystem message switching. Thus, one or more chains of processing will occur in response to the original message. If two or more subsystems in these chains may make potentially interdependent changes to the same data files, then the corresponding Subsystem Control Table (SCT) entries must all specify LOG=YES and RESTART=YES. This will ensure that the originating (mother) subsystem is restarted if any subsequent program did not complete.

In a complete system failure (for example, machine or power failure) Intercomm software cannot determine the status of terminal transmissions in process at the time of failure. Therefore, following complete failure, the remote terminal operator must verify the conclusion of the last operation: if it was an update operation and if so whether or not all results from that operation had been received. This is the only terminal operator interaction relevant to restarting Intercomm.

## 1.2.1 Message Logs

The log facility for Intercomm utilizes a data set whose ddname is INTERLOG. This log contains entries reflecting the status of messages for subsystems and terminals, and entries of before and after images of data files being updated. Also included on INTERLOG are user log entries, checkpoint records, message accounting records, etc. (See Figure 1).

INTERLOG may be specified on its DD statement as being tape or disk. If the DD statement for INTERLOG is omitted then no logging will be performed and of course no restart is possible. The computer operator is notified at startup time if INTERLOG is not specified, or if the file cannot be opened. If you intend to use the restart facility, the computer operator's response to this message should be to cancel Intercomm and request programming assistance before proceeding.

INTERLOG records appear as standard undefined records (that is, RECFM=U) but can be read using QSAM (RECFM=VB). Special techniques are utilized in creating the log, as follows:

- <u>Access Method</u>

  BSAM rather than QSAM is employed.

- <u>Variable Buffer Size</u>

  "Average" length buffers (whose length is specified for SPALIST by the parameter LGBLK) are normally used. Where possible, log records are treated in blocked mode to form a buffer whose size approximates LGBLK. However, if a record to be logged exceeds the size of LGBLK it will be logged in its own buffer of the appropriate length. Any size record (up to the DD statement BLKSIZE specified for INTERLOG) can be logged.

- Synchronous Logging

  Recognizing the need for priority in logging items pertaining to critical user components (critical relative to restart), certain records are written immediately. For example, log entries for an important subsystem are made immediately by adding the entry to the current buffer, then immediately writing out the buffer. INTERLOG is blocked for noncritical items, and effectively unblocked for critical items. Critical subsystems are specified by the SYCTTBL parameter LSYNCH=YES, and critical terminals by the BTERM/LU parameter LSYNCH=YES. For Multiregion interregion queues, the LSYNCH=YES parameter is specified on the SUBSYS macro in the Region Descriptor Table used by the Control Region.

- Logging for terminals, subsystems, and regions is controlled by the LOG parameter. Restartability is controlled by the RESTART parameter. For a detailed description of these and the LSYNCH parameters, see the following:

  Basic System Macros: BTERM, SYCTTBL macros;
  SNA Terminal Support Guide: LCOMP, LUNIT macros;
  Multiregion Support Facility: REGION, SUBSYS macros.

Figure 1 describes the Intercomm log records. See the Restart Use column for those log entries that are utilized in Restart/Recovery. All other log entries, including user log entries, are ignored.

## 1.2.2 Message Accounting

To make the warm restart concept function as rapidly as possible, restart involves reading the log backwards only as far as is required to recover all necessary messages. This "how far back" information is developed by the Message Accounting routine, MSGAC, a subprogram of LOGPUT, the message logging routine. MSGAC operates as part of the live Intercomm environment. LOGPUT examines every log entry to determine if this new entry reflects a change in the "read-back point" of the log tape. For every 255 completed messages, MSGAC will insert message accounting records onto INTERLOG.

These records reflect the read-back point. When restart begins reading INTERLOG backwards, the first message accounting record encountered will identify the location of the actual read-back point.

| Internal Code | External Code | Format | Description | Origin | Restart Use |
|---|---|---|---|---|---|
| X'00' | 00 | HT | Checkpoint Record | Checkpoint | Yes |
| C'2' | 01 | HT | Message queued for subsystem by Front End or a subsystem | Message Collection | User |
| C'R' | 02 | HT | Message restarted through the system | LOGPROC | User |
| C'P' | 03 | HT | Message restarted--related to Data Base Recovery | LOGPROC | User |
| C'T' | 30 | HO | Message passed to subsystem for processing | Subsystem Controller | User |
| C'Z' | 40 | HT | Message passed to Front End (test mode only) | FESEND | No |
| X'41'- X'6F' | 41- 6F | HT | User called LOGPUT | Any Subsystem | No |
| X'80'- X'8E' | 80- 8E | HT | File Recovery before-images | IXFLOG | User |
| X'8F | 8F | HO | Checkpoint Records indicator | IXFCHKPT | Yes |
| X'90'- X'9E' | 90- 9E | HT | File Recovery after-images | IXFLOG | User |
| X'9F' | 9F | HT | Intercomm Startup | LOGPUT | Yes |
| X'A0' | A0 | HO | Message restart begun | LOGPROC | Yes |
| X'A1' | A1 | HO | Message restart finished: all subsequent log entries produced by live Intercomm | LOGPROC | Yes |
| X'AA' | AA | HT | Intercomm Closedown | LOGPUT | No |
| X'C0' | C0 | HT | Region started (Multiregion only) (Text=Region-id(s)) | MRINTER | No |
| C'A' | C1 | HT | Message successfully queued for Satellite Region | MRQMNGR CR only | User |

| | |
|---|---|
| Internal Code: | Log code in core during processing (snaps and dumps) |
| External Code: | Log code after translation by LOGPUT (INTERLOG printout) |
| Format: | HT for header and text, HO for header only |
| Restart Use: | Yes, No, User (specified via user-coded system macros) |

Figure 1. INTERLOG Entries (Page 1 of 2)

| Internal Code | External Code | Format | Description | Origin | Restart Use |
|---|---|---|---|---|---|
| C'B' | C2 | HO | Message successfully passed to Satellite Region | MRQMNGR CR only | User |
| C'C' | C3 | HO | Message lost (Region/Hold Q full) or flushed (SR/SS down) | MRQMNGR CR only | User |
| C'I' | C9 | HT | Sign on/off processing, security violation messages | ESS | No |
| C'3' | FA | HO | Normal message complete | Subsystem Controller | User |
| C'5' | FB | HO | Unprocessed message--invalid subsystem/QPR code | Message Collection | User |
| C'6' | FC | HO | Unprocessed message--core and disk queue full | Message Collection | User |
| C'8' | FD | HO | Message cancelled--program error or time-out, I/O error, or flushed by command | Subsystem Controller | User |
| C'9' | FE | HO | Message flushed by Retriever, used when application program does not obtain (via GETSEG) all parts of a segmented message; or message failed security check | Retriever<br><br>Subsystem Controller | No |
| C'1' | F1 | HT | Message after verb verification | USRBTLOG (optional) | No |
| C'2' | F2 | HT | Message queued for transmission | FESEND | User |
| C'3' | F3 | HO | Message transmitted, discarded (MSGHUSR=Z), or flushed (MSGHUSR=F) | Front End | User |
| C'4' | F4 | HO | 3270 output message content invalid--message dropped. | BLHOT | No |
| C'5'-<br>C'8' | F5-F6<br>F7-F8 | HO<br>HT | Transmitted DDQ msg status: see <u>SNA Term. Support Guide</u> | Front End | No |
| X'FF' | FF | HT | Intercomm Restart Accounting | MSGAC | Yes |

Figure 1. INTERLOG Entries (Page 2 of 2)

## 1.3 Message Restart Logic

When data base or file recovery is not utilized, the restart logic is quite simple. When reading the tape tackwards, information from certain message headers is temporarily stored. This stored information is the basis for determining what to do with the header/text log entries as they are encountered. The information from the header is such that it can uniquely identify a message within a subsystem (including recursive entries to a subsystem). Since the log is read backwards, message log entries will be encountered in this order:

- Subsystem Completed (normally or abnormally)
- Subsystem Started.
- Message Queued for a subsystem

When the "message queued for a subsystem" log entry (header/text) is encountered the information stored from the previously encountered log entries for this message is examined and the following rules apply to the restart analysis:

- if the message completed successfully (a log code FA was encountered), the message is not restarted;
- if the message failed in processing by a subsystem (time-out, program check, failure to acquire all segments, etc.) then this message may be restarted.
- if the message began processing but had not completed at the time of failure (a log code entry 30 but no FA, FB, FC, or FD was encountered) then the message may be restarted and its log code on the queue is set to "R" indicating that it was an in-process message being restarted.

The above rules are the criteria applied to a single message out of context; they may be overridden by the considerations listed below:

- Ancestral Messages - if any "ancestor" of a message has been restarted for any reason, the message is discarded. This rule requires some clarification: if during the processing of message A, the subsystem generates message B and queues it for processing via MSGCOL or FESEND, message A is the mother of message B. Starting at any message, restart logic can work back to the original terminal input, going from the message to its mother, the mother's mother, and so on. These are collectively the message's ancestors. A message is only restarted if all its children are discarded. This applies to Front End as well as Back End messages. Thus, a user subsystem's message being restarted overrides restart of a descendent message to the Output Utility (or any other subsystem) and to the terminal. However, if the ancestor is not logged, or not marked for restart, only the child (subsystem or terminal output) is restarted.

- <u>Conversational Messages</u>

  If the message is part of a conversation, that is, part of a subsystem calling CONVERSE, and CNVREST=YES is coded in the subsystem's SYCTTBL macro, the message is restarted if it is the first message in the conversation (even if it completed), and discarded if it is not the first (even if it did not complete).

  <u>NOTE</u>: In order to insure file integrity, conversational subsystems performing data base updates should be designed such that either (a) a message is switched to a nonconversational subsystem to perform the update(s), or (b) the update(s), is (are) performed as processing logic for the last input message in the conversation. Otherwise, updates may be performed twice if CNVREST=YES is coded.

- <u>Segmented Messages</u>

  If a message is part of a segmented message sent to OUTPUT or CHANGE/DISPLAY, and SEGREST=YES is coded in the sending subsystem's SYCTTBL macro, the disposition of the trailer (final) segment determines what happens to other segments. They are restarted if the trailer is restarted, or discarded if the trailer is discarded.

  Messages that are lost because <u>terminal</u> queues are full will not be restarted.

In a Multiregion Intercomm system, message restart and file/data base recovery is possible only in those regions which create their own Intercomm log. In Satellite Regions, only 01-30-FA logging of subsystem processing is done, along with log records necessary to message restart, checkpointing and file recovery. In the Control Region, terminal transmission logging (F2-F3) and Multiregion message queues (for Satellite Region transmission) are rebuilt at restart time. Further details are described in <u>Multiregion Support Facility</u>.

Whether restart is actually performed depends on the user specification in the SYCTTBL, BTERM/LUNIT and SUBSYS macros. If RESTART=NO is coded then no messages are restarted regardless of the circumstances. RESTART=IFPOSBL has the same meaning in the restart analysis as RESTART=YES. However, RESTART=IFPOSBL affects message accountability such that the read-back point for restart analysis may or may not include all those IFPOSBL messages. The read-back point definitely includes all RESTART=YES messages.

In cases where data base or file recovery is not included, the only integrity problem concerning a restart involves those messages that were in process at failure time. Thus, if a message was being transmitted when a power failure occurred, the restarted Intercomm would retransmit that entire message or DDQ (FECMDDQ request). Potential integrity problems for files or data bases are discussed in the following chapter.

## 1.4 File Recovery Concepts

The previous section described the concepts utilized in Intercomm's restarting of messages where data base or file recovery was not involved. This section concerns the recovery of files and the coordinated recovery of related and unrelated Intercomm messages. The restart concepts previously presented for messages are also the basis for file restart/recovery. Restart facilities have been extended to provide for the special requirements of data file integrity following system failures.

In this manual the terms "file(s)" and "data base(s)" are used interchangeably to mean all on-line files. Figure 2 lists file access method support by the File Recovery facility.

| Access Method | Supported |
|---|---|
| BDAM | Yes |
| BISAM, QISAM | Yes |
| VSAM | Yes |
| IAM | Yes (ISAM) |
| BSAM, QSAM | No |
| AMIGOS | No |

Figure 2. Supported File Access Methods

However, the basic concepts of File Recovery also apply to specific data base managers supported under Intercomm. For additional details on Data Base Management System recovery, refer to the Intercomm DBMS Users Guide.

The focal points for file and data base recovery capabilities are the Intercomm checkpoint and log files, and the file/data base activity logs.

### 1.4.1 Checkpoints

The Intercomm checkpoint is not a "checkpoint" in the normal data processing usage where a picture image of core is written as a checkpoint to some data set. Rather, the Intercomm checkpoint function saves only a few critical table entries on disk, involving only minimum I/O activity for the checkpoint. To Intercomm, the checkpoint has special significance relative to file recovery in that it indicates all Intercomm subsystems performing file updates have been quiesced (no new messages started) until checkpoint processing has completed.

The Intercomm checkpoint functions as follows:

- Users specify a checkpoint time interval such that when this interval expires the checkpoint process begins;
- All subsystems that may at any time perform an update activity to a file that is to be recovered are identified by the subsystem SCT entries. During checkpoint, these update subsystems are marked as nonschedulable so that no new messages will be started through these subsystems. (Messages can be received for these subsystems but will remain in the queues.)
- Intercomm examines each of these update subsystems for current message processing activity. When all activity for these subsystems has concluded, the checkpoint can begin. Meanwhile, all other activity such as nonupdate subsystems, line control, the Output Utility, etc., continues uninterrupted.
- A checkpoint record(s) containing pertinent table data is written to the checkpoint data set.
- A checkpoint record is written to the Intercomm log noting the date/time of the checkpoint.
- Subsystems that had been marked nonschedulable are started up again. Normal file update activity resumes.
- The interval timer for the next checkpoint is set.

From the above it can be seen that a checkpoint merely represents a "clean point in time" to which a restore can be made. At this point in time the file was intact on disk and no modification or update activity was in progress. The basis for file recovery is to proceed backward to recover files from the point of failure to this checkpoint, then restore the checkpointed table information, and finally to proceed with the live system.

## 1.4.2 File Activity Logs

The Intercomm File Handler provides for update (delete, add, insert) activity logging for BDAM, ISAM, and VSAM files. These log entries include before- and after-images that are used to recover a file. The file activity log is the same data set (INTERLOG) for Intercomm File Handler entries as for message status entries.

### 1.4.3 Destruction of Files

As part of the system failure, it is possible that all or a portion of a data file (and its indices) are physically or logically destroyed. In this case it is the user's responsibility to restore the file back to its condition at failure time. This is accomplished in two distinct operations:

- first, the last backup (complete copy) of the file is reloaded, that is, an off-line file restore (tape restore, tape to disk) is performed;
- second, all after-images from the Intercomm log are applied to the file in their original order by an off-line utility included with the Intercomm File Recovery facility.

When the above procedures have been performed, the file is returned to the status at failure time and normal recovery processing can proceed.

Normal recovery (file reversal) is required for <u>every</u> file supported by restart/recovery following <u>every</u> failure. However, complete recovery (recreation) following data destruction is required <u>only</u> on those <u>specific</u> files where needed. For example, assume that there are VSAM files, each on one of three separate packs. If a head crash occurs on one of these packs then only the file on that pack need be completely recovered. This specific type of complete recovery is significant. Although complete recovery should be infrequently required, it is often a long running process. The normal recovery discussed below is typically quite rapid. Another implication of this specific recovery capability is that often the user can lessen the time burden for recovery by segmenting the data structure into smaller, discretely recoverable portions.

### 1.4.4 Normal Recovery

If a system failure has occurred without destroying the file, normal (on-line) recovery can be immediately started. (If data destruction occurred, then the above stated complete recovery must first be performed before proceeding with normal recovery.) Normal recovery involves backing out file changes to the last checkpoint. (Since Intercomm checkpoints are typically rapid with little performance interference, checkpoints can be frequent.) With frequent checkpoints, normal recovery need not involve much data restoration.

The backing out to checkpoint process involves applying before file images to files in the opposite order (backwards order) to their original update. Therefore, updates are reversed back to the checkpoint time. Additions are reversed by deletion; deletions are reversed by addition.

### 1.4.5 Coordinated Message Recovery

While the normal file recovery is being performed, Intercomm message recovery proceeds concurrently. The coordination between Intercomm message and file recovery is software controlled, transparent to the computer operator. The order of starting and ending for these separate recoveries is irrelevant. File recovery is performed by the Intercomm restart itself, and thus proceeds simultaneously with message restart.

As part of data file recovery, the files subsequently need to be brought from the time of the checkpoint forward to the time of the system failure and into live mode. This is implemented by expanded Intercomm message recovery. Message recovery was previously described in detail. Essentially this message recovery involves returning to the queues those messages that were in the queues at failure time, and entering into the queues those messages that were in process at failure time. Those messages that were partially processed are requeued with a special log code of "R". Thus, only partially processed or totally unprocessed messages are normally put into the message queues following restart. However, if file recovery is involved, Intercomm will also restart those messages causing file updates which had completed but had done so after the last checkpoint was taken.

Thus, message restart involving file recovery proceeds as follows:

1. If necessary, complete recovery is performed off line including file reload and application of after-images for any destroyed file.

2. Data files are returned to checkpoint time status by backing out updates (before-image operation).

3. Concurrently with point 2, Intercomm recovers messages that were

   - queued but not started
   - started but not completed (may have updated a file)
   - completed but had updated files since the last checkpoint

4. After points 2 and 3 are completed, restarting of message processing begins, and updates that had been reversed by returning to checkpoint will be reapplied as the messages that caused those updates are reprocessed.

5. Concurrently with point 4, Intercomm resumes live mode.

6. If the Serial Restart Facility is implemented (see the Operating Reference Manual), then live mode Intercomm may be selectively controlled or postponed until point 4 is completed.

Output messages generated while processing restarted messages will be transmitted. Subsystem logic can identify restarted messages via the log code (C'R') in the message header and prepare special message text as required, to notify the terminal operator of the possibility of duplicate output.

## 1.5 BACKOUT-ON-THE-FLY

Backout-on-the-Fly provides on-line dynamic reversal of file updates following a subsystem failure, and is executed following the occurrence of these situations:

- Subsystem thread program check
- Subsystem thread timeout
- Specific requests by a subsystem

Backout-on-the-Fly follows the same methodology as file recovery: before-images are applied to updated files in the appropriate reverse sequence. The facility differs from restart file recovery in that Backout-on-the-Fly compares the failing subsystem's "after-images" with the file's current images to ensure that no intervening subsystem subsequently updated the same record. In the event of a mismatch of after-to-current image, an operator reply to a PMIWTOR can choose to either abend Intercomm or ignore the situation. If the abend is chosen, the normal file recovery scheme can be used to successfully apply the before-images back to the last checkpoint.

Backout-on-the-Fly requires the Dynamic Data Queuing Facility. Backout-on-the-Fly places the thread's before- and after-images on a DDQ. If the thread completes successfully, then the DDQ is deleted. If the thread fails, then reversal is performed.

Backout-on-the-Fly logs and reverses all files marked as reversible to standard file recovery through the normal FAR file recovery control options. Additionally, Backout-on-the-Fly is selected by subsystem. The overhead (creating and writing to DDQs) is incurred for only those subsystems deemed likely to fail. However, Backout-on-the-Fly will log only File Recovery's recoverable files regardless of selected subsystems.

If all file logging is shut off through the GPSS "stop log" command, Backout-on-the-Fly remains functional. If a subsystem performs both file and DBMS updates, then Backout-on-the-Fly may be incompatible. An exception is IDMS, which also provides a comparable facility. SYSTEM 2000 may be compatible if all subsystems use deferred updates as a programming standard, with the last subsystem action prior to GOBACK being to apply updates. DBMS situations should be examined carefully when using Backout-on-the-Fly.

See Chapter 5 for implementation procedures.

# Chapter 2

# FILE RECOVERY

## 2.1 Timing Considerations

File recovery incorporates two types of operations: reversal and recreation. Reversal takes a file as it was at system failure time and returns it to a state it was in at some time in the past: a checkpoint time status. Recreation uses an old copy of a file (backup) and brings it up to its status at system failure time.

The basis for file recovery is a log that includes: before-images of every record that is updated (deleted); and the key or address of every record that is added. During reversal these log entries are used to nullify changes and delete additions. This reversal process takes place at system startup in restart mode.

File recreation utilizes the after-image log entry of every change to the file. Recreation is handled off-line by a separate utility.

The Intercomm File Handler totally provides for making all file activity log entries. The File Handler's decision to log before-images (to reverse a file) and/or after-images (to recreate a file) is on a file by file basis as user-specified. This is performed independently of the subsystem addressing the file. The attributes of reversibility and recreatability belong to files for the duration of an Intercomm run and are not related to programs.

The user specifies file recovery information through File Attribute Records (FAR options). (Specific coding of FARs is described in Chapter 3; concepts of use are presented here.) Each file for which restart/recovery may be invoked must have a FAR. FARs are optional for those files not to be reversed or recreated. The FARs are read by the File Handler during initialization from a data set whose ddname is ICOMIN. This data set can be an input stream card-image file as part of the Intercomm execution JCL or may reside on any sequential input card-image file. Information from each FAR is used to create an appendage to the File Handler Data Set Control Table (DSCT) for that file, which contains various file-related data. The FAR appendage data is called a File Attribute Block (FAB). File Recovery Dsects are contained in the member FRDSECTS on the release library.

File activity logging is performed on the Intercomm log file, INTERLOG. Logging is selective by file as indicated by the FAR entries. Further, logging is only performed for updates, deletions, or additions; thus no logging overhead is associated with read-only or inquiry activities. Figure 3 details message header formats for file recovery and message status log entries.

| Message Header for File Recovery | BYTE | Standard Intercomm Message Header |
|---|---|---|
| | 00 | |
| MSGHLEN | 01 | MSGHLEN |
| ▨ | 02 | MSGHQPR |
| ▨ | 03 | MSGHRSCH |
| ▨ | 04 | MSGHRSC |
| ▨ | 05 | MSGHSSC |
| MSGHTXTL (record length) | 06 | |
| | 07 | MSGHMMN |
| MSGHKEYL (key length) | 08 | |
| | 09 | |
| | 0A | |
| MSGHDAT | 0B | MSGHDAT |
| | 0C | |
| | 0D | |
| | 0E | |
| | 0F | |
| | 10 | |
| MSGHTIM | 11 | MSGHTIM |
| | 12 | |
| | 13 | |
| | 14 | |
| | 15 | |
| | 16 | |
| MSGHBKID (Block Identifier for BDAM files) | 17 | |
| | 18 | MSGHTID |
| | 19 | |
| | 1A | |
| | 1B | |
| | 1C | |
| | 1D | MSGHCON |
| | 1E | |
| | 1F | MSGHFLGS |
| MSGHDD (file ddname) | 20 | |
| | 21 | MSGHBMN (Release 10) |
| | 22 | |
| | 23 | MSGHSSCH |
| | 24 | MSGHUSR |
| | 25 | |
| | 26 | MSGHBMN (Release 9) |
| MSGHLOG | 27 | MSGHLOG |
| MSGHMACR | 28 | MSGHBLK |
| ▨ | 29 | MSGHVMI |

Figure 3. Message Header Fields for File Recovery

Normally, the logging of file changes is not synchronized with the actual file access; if the log entry happens to fill an INTERLOG buffer, the entry will be written out before the access takes place. On the other hand, if the entry is short and the buffer is mostly empty, many file accesses may occur before the INTERLOG buffer is written. In the case of an interceptable abend, the Intercomm routine STAEEXIT will write out partially filled INTERLOG buffers; but log entries could be lost in situations such as hardware failures.

It is, however, possible to ensure that all file activity entries are logged before they are applied. Thus, two levels of reversibility may be defined:

- <u>Level 1</u> - (requested by coding REVERSE=YES in the FAR) is the normal, asynchronous logging with a buffered INTERLOG and the concomitant risk of incomplete file reversal in a major system failure;

- <u>Level 2</u> - (requested by coding REVERSE=CRITICAL in the FAR) means the INTERLOG entry is made <u>before</u> changing the file. Level 2 ensures a complete reversal regardless of the type of system failure. However, Level 2 logging processing speed is somewhat affected since at least two I/O activities (the I/O request itself plus the log entry) must complete before control is returned from update-related File Handler requests.

The overhead associated with file recovery support aside from logging overhead is as follows:

- QISAM and BISAM updates

  QISAM updates (GET-PUTX pairs) and BISAM updates (READ KU-WRITE K pairs) never require any additional I/O because an image of the record being updated is always in core when the update request comes to the File Handler.

- BDAM Updates

  BDAM updates require an extra read if the update logic is not a READ followed by a WRITE of the same record. The File Handler maintains its own buffer for reversible BDAM files; thus READ (1) - READ (2) - WRITE (1) logic destroys the first record read with the second read. Therefore an extra read is required to log the before-image at write time. Additions to files never require extra reads because only the key or block address is logged. If the subsystem uses a pointer on the BDAM file to find free blocks, such that file access logic is: read pointer, write new block, write pointer; then the File Handler will do an unnecessary read of the dummy data at the new block's location. There are certain cases where additions are not logged because of an inability to reverse them. (See Figure 4.)

- VSAM Updates

  VSAM updates never require any additional I/O because an image of the record being updated (Control Interval) is always in core when the update request (PUTV) comes to the file handler; that is, both before-image and after-image are logged when the record is updated.

The number of extra write operations required for INTERLOG, when REVERSE=CRITICAL is specified, is possibly one per File Handler WRITE or PUT operation. It may be more. To make the order of entries on the log reflect the order in which they were passed to LOGPUT, the queue of filled buffers is flushed before the critical entry is written. For a file specified REVERSE=YES there is no fixed formula. If the file block size is large compared to the log buffer size and mostly updates are performed, the ratio of log writes to output operations will be close to the maximum of one; if the block size is small or a lot of additions are performed, the ratio will be lower.

In special situations, certain specific records will be repeatedly updated while running. For example, RBNO is often updated following every update to a BDAM file. In this and similar situations the user can identify via a FAR CHECKPOINT parameter those frequently updated records. In these cases logging is not performed on updates to these records. Instead these records are logged only at checkpoint time. This technique will considerably reduce overhead in these special situations while preserving data integrity following system failures. There is no limit to the number of records that can be specified in the CHECKPOINT FAR option.

## 2.2 File Reversal

The log entries made by the File Handler (see Figure 1) follow standard Intercomm format: message header plus text. The message header contains the file ddname and space for a block ID, key length, and text length, as previously shown in Figure 3. The log code identifies the reversal action to take. For example, the log code corresponding to an ISAM update specifies that for reversal the data portion of this log entry is a record image to be applied with a READ KU, WRITE K sequence. Reversal actions are summarized in Figure 4.

Reversing updates is straightforward. Reversing file additions involves special considerations. IBM allows only deletions for BISAM files, and ERASE operations for VSAM KSDS and RRDS files. To make reversal as general as possible, Intercomm supports user-defined deletion codes in the File Handler as follows: users specify the delete field in the record and the value in that field specifying "this record is deleted". The delete field may be up to eight bytes long, and can be assigned any value. Delete codes are valid for all types of data sets except VSAM files. There are two parameters in the FAR relating to delete codes: DELETE specifies where and what the code is; and CHECK tells the File Handler when to look for it--on input, on output, or both.

CHECK=OUT can be specified on the FAR for ISAM files only. If a request to add a record results in a duplicate key situation, then the File Handler reads the record and examines it for an existing delete code. If the delete code is there, then it overlays the duplicate record. Otherwise the calling program is returned a "duplicate key" return code. The DELETE and CHECK options do not apply to VSAM files.

| | UPDATE | ADD |
|---|---|---|
| Bi | X | * |
| Q | X | * |
| BF | X | NS |
| BFK | X | X |
| BV | X | ** |
| BVK | X | *** |
| VS/ESDS | X | NS |
| VS/KSDS | X | X |
| VS/RRDS | X | X |

| | |
|---|---|
| Bi= | BISAM |
| Q = | QISAM |
| B = | BDAM |
| F = | Fixed Format |
| V = | Variable Format |
| K = | Keys |
| VS= | VSAM |
| X = | Complete Reversal of Operation |
| NS= | Not supported or not applicable |

* If the ISAM deletion option is supported (OPTCD=L), or if a delete code is specified in the File Attribute Record, the key will be logged when the record is added, and deleted during recovery. Otherwise, BISAM adds are not logged.

** To completely nullify the effect of an add, the capacity record of the track receiving the new record would have to be logged, then set back to its old value during update; but it cannot be done. If you specify a delete code, the record's location will be logged when it is added and the delete code inserted during update. To compress the file, develop a simple program using the File Handler to read the file sequentially and write it to a new data set, specifying CHECK=IN on the File Attribute Record.

*** The above remarks apply, and use of keys introduces a further problem: if a record is added with a certain key, and then, after restart, a write is performed on another record with the same key on the same track, the second record is inaccessible to a READ DK. To avoid this problem, specify a delete code that will set the record's key to a value that will not be duplicated by any other key in your file.

Figure 4. File Recovery Reversal Action

## 2.3 File Recreation

A file is recreatable by Intercomm if RECREATE is specified on its corresponding FAR. When RECREATE is specified, then a log entry is made of every after-image of that file. Again, the log entry is standard header/text format with the ddname of the file included in the header, and the text portion being the after-image. File recreation is performed by an Intercomm-supplied off-line utility, IXFCREAT. The user specifies which files are to be recreated. Optionally, the user may also specify a starting date and time to be searched for in the Intercomm log. By including this information, only those file changes made after that time will be reapplied to the file. Several files may be recreated in a single execution of the utility.

## 2.4 File Recovery Log Entries

The contents and sources of the various file recovery log-entry fields are summarized in Figures 5 and 6. The following are explanatory notes to the table.

MACRO - the output macros issued by the File Handler

# - the number passed by the File Handler to IXFLOG identifying the macro it is about to issue (see description of IXFLOG)

LOG CODE - the byte inserted in MSGHLOG. Note that different macros may map to the same log code.

FORMAT - H stands for header, K for key, and R for record.

INFORMATION SOURCES -

- A name in capital letters means the contents of the storage location with that name. A value in small letters refers to an EQU symbol (dsctbdhd) or another table entry on the same line (record length).
- F means fixed format file; V means variable format file.
- RDW means record descriptor word; BDW means block descriptor word.
- The names beginning with PARM... are symbolic names for elements of the File Handler parameter list: PARMAREA is the third parameter (I/O area), PARMRKEY is the fourth (key for ISAM/BDAM/VSAM with keys, block-id for BDAM without keys, RRN for VSAM RRDS, RBA for VSAM ESDS), and PARMRBLK is the fifth (block-id for BDAM with keys).

| ENTRY TYPE | | | | INFORMATION SOURCES | | | | |
|---|---|---|---|---|---|---|---|---|
| MACRO | # | LOG CODE | FORMAT | RECORD ADDRESS | RECORD LENGTH | KEY ADDRESS | KEY LENGTH | BLOCK-ID/RBA/RRN |
| PUTX | 0 | 80 | HKR | DSCTBUFR | F=DCBLRECL V=RDW | record- address +DCBRKP** | DCBKEYLE | - |
| PUT | 1 | 81 | HK | - | - | PARMAREA +DCBRKP | DCBKEYLE | - |
| WRITE DIX | 2 | 82 | HR | DSCTBDBF +dsctbdhd | F=DCBBLKSI V=BDW | - | - | DSCTBLOK |
| WRITE DI | 3 | 82 | HR | DSCTBDBF | F=DCBBLKSI V=BDW | - | - | PARMRKEY |
| WRITE DKX | 4 | 83 | HKR | DSCTBDBF +dsctbdhd +keylength | F=DCBBLKSI V=BDW | PARMRKEY | DCBKEYLE | DSCTBLOK |
| WRITE DAF (fixed) | 5 | 84 | H | - | - | - | - | DSCTBLOK |
| WRITE DK | 6 | 83 | HKR | DSCTBDBF +dsctband +keylength* | F=DCBBLKSI V=RDW | PARMRKEY | DCBKEYLE | PARMRBLK |
| WRITE DAF (variable) | 7 | 85 | H | - | - | - | - | DSCTBLOK |
| WRITE K | 8 | 80 | HKR | DECBLOGR | F=DCBLRECL V=RDW | DECBKEY | DCBKEYLE | - |
| WRITE KN | 9 | 81 | HK | - | - | PARMRKEY | DCBKEYLE | - |
| WRITE DIX (keyed) | 10 | 86 | HKR | DSCTBDBF +dsctbdhd +keylength | F=DCBBLKSI V=BDW | DSCTBDBF +DSCTBDHD | DCBKEYLE | PARMRBLK |
| WRITE DI (keyed) | 11 | 86 | HKR | DSCTBDBF +dsctbdhd +keylength* | F=DCBBLKSI V=BDW | DSCTBDBF +DSCTBDHD* | DCBKEYLE | |
| PUTX (ixfqisam) | 12 | 80 | HKR | Register 10 | F=DCBLRECL V=RDW | record address +DCBRKP | DCBKEYLE | - |
| WRITE KN (V-Format update) | 13 | 87 | HKR | DECBLOGR* | F=DCBLRECL V=RDW | PARMRKEY | DCBKEYLE | - |
| VSAM PUT (keyed/RRN update) | 14 | 88 | HKR | RPAREA | SHOWRECL | record address +RKP | ACKEYL or default of 4 for RRDS | RPLARG for RRDS |
| VSAM PUT (addressed update) | 15 | 89 | HR | RPAREA | SHOWRECL | - | - | SHOWRBA |
| VSAM ERASE | 16 | 8A | HKR | RPAREA | SHOWRECL | record address +RKP | ACKEYL | - |
| VSAM PUT (keyed add) | 17 | 8B | HK | - | - | PARMAREA +RKP | ACKEYL | - |
| VSAM PUT (addressed add or seq. ESDS) | 18 | *** | - | - | - | - | - | - |
| checkpoint marker | | 8F | H | - | - | - | - | - |

* IXFLOG will call READ to set up before-image in the buffer if it's not already there
** Unless the file is unblocked and RKP=0; then the key address is in DSCTOIKE
*** Only recreate (from after-image) is supported; before-image for reversal does not apply

Figure 5. Table Summary of Before-Image Log Entries

| ENTRY TYPE | | | | INFORMATION SOURCES | | | | |
|---|---|---|---|---|---|---|---|---|
| MACRO | # | LOG CODE | FORMAT | RECORD ADDRESS | RECORD LENGTH | KEY ADDRESS | KEY LENGTH | BLOCK-ID/RBA/RRN |
| PUTX | 0 | 90 | HKR | PARMAREA | F=DCBLRECL V=RDW | record +DCBRKP | DCBKEYLE | - |
| PUT | 1 | 97 | HR | PARMAREA | F=DCBLRECL V=RDW | - | - | - |
| WRITE DIX | 2 | 92 | HR | PARMAREA | F=DCBBLKSI V=BDW | - | - | PARMRKEY |
| WRITE DI | 3 | 92 | HR | PARMAREA | F=DCBBLKSI V=BDW | - | - | PARMRKEY |
| WRITE DKX | 4 | 93 | HKR | PARMAREA | F=DCBBLKSI V=BDW | PARMRKEY | DCBKEYLE | PARMRBLK |
| WRITE DAF (fixed) | 5 | 94 | HKR | PARMAREA | DCBBLKSI | PARMRKEY | DCBKEYLE | PARMRBLK |
| WRITE DK | 6 | 93 | HKR | PARMAREA | F=DCBBLKSI V=BDW | PARMRKEY | DCBKEYLE | PARMRBLK |
| WRITE DAF (variable) | 7 | 95 | HR | PARMAREA | BDW | - | - | PARMRKEY |
| WRITE K | 8 | 90 | HKR | PARMAREA | F=DCBLRECL V=RDW | DECBKEY | DCBKEYLE | - |
| WRITE KN | 9 | 91 | HKR | PARMAREA | F=DCBLRECL V=RDW | PARMRKEY | DCBKEYLE | - |
| WRITE DIX (keyed) | 10 | 96 | HKR | PARMAREA | F=DCBLKSI V=BDW | PARMRKEY | DCBKEYLE | PARMRBLK |
| WRITE DI (keyed) | 11 | 96 | HKR | PARMAREA | F=DCBBLKSI V=BDW | PARMRKEY | DCBKEYLE | PARMRBLK |
| PUTX (ixfqisam) | 12 | 90 | HKR | PARMAREA | F=DCBLRECL V=RDW | record +DCBRKP | DCBKEYLE | - |
| VSAM PUT (keyed/RRN update) | 14 | 98 | HKR | PARMAREA | F=ACLRECL V=RDW | RPAREA +RKP | ACKEYL or default of 4 for RRDS | RPLARG for RRDS |
| VSAM PUT (addressed update) | 15 | 99 | HR | PARMAREA | F=ACLRECL V=RDW | - | - | SHOWRBA |
| VSAM ERASE | 16 | 9A | HK | - | - | RPAREA +RKP | ACKEYL | - |
| VSAM PUT (keyed add) | 17 | 9B | HKR | PARMAREA | F=ACLRECL V=RDW | PARMAREA +RKP | ACKEYL | - |
| VSAM PUT (addressed add or seq ESDS) | 18 | 9C | HR | PARMAREA | F=ACLRECL V=RDW | - | - | PARMRBA or low values if seq. ESDS |
| startup marker | | 9F | H | - | - | - | - | - |

Figure 6. Table Summary of After-Image Log Entries

# Chapter 3

# FILE ATTRIBUTE RECORDS

## 3.1 Introduction

Different files require different provisions for recovery. Certain files like read-only or write-only (SYSOUT) data sets, require no provisions. For others it may be sufficient to recreate the file from a backup copy, without reversing updates during restart; for example, a customer directory that is expanded when an order comes in with a new name on it. There is no point in deleting a directory entry during file reversal; it would even be dangerous if it contained an item like a customer code that might get assigned a different value the second time around. On the other hand, if the directory contained billing figures as well as names and addresses, reversal of updates is necessary, to ensure that people are not charged twice. In short, recovery options must be specified file by file.

File Attribute Record (FAR) control statements are used to specify file recovery options. The FARs are read during File Handler initialization after all the internal DSCTs have been initialized. The information from the FARs is encoded in DSCT appendages called File Attribute Blocks (FABs). ICOMIN is the ddname of the card-image FAR data set, which may be instream, a PDS member, or a sequential data set. It is accessed via QSAM GET.

Each File Attribute Record follows a general coding format:

```
ddname,attribute1,attribute2,...,attributen
```

where the individual parameters (or attributes) are associated with a particular ddname defined in the execution JCL. The FAR statements do not have to be in the same order as the DD statements in the Intercomm execution JCL.

IXFFAR, the FAR processing module, contains logic to send the image of each FAR it reads to the operator console. These WTOs are indispensible to detecting errors in the FAR deck, because they identify the FAR to which it refers. Once the FARs are error-free, the user can suppress FAR image messages. A control statement containing the single value NOMESSAGES as the first statement in the FAR data set stops FAR images from going to the console. Error messages will still be printed.

Rules describing syntax for coding a FAR that the FAR-handling module can interpret are listed below. Additional specifications are listed in the individual parameter descriptions and further in the applicable error message explanations.

FAR syntax rules are:

- Separate parameters by commas

- A FAR may span as many as five statements. Place a nonblank character in column 72 if the next card is a continuation. Columns 73-80 are ignored; they can be used for sequence numbers. A ceiling on the number of continuation cards is necessary because the whole FAR has to be read into a fixed-length buffer before it can be processed. The statement that reserves space for the buffer in the FAR handling module, IXFFAR, is:

  ```
  FARBUF   DS   (5*71)C
  ```

  If a FAR spans more than five cards, raise the coefficient of 71 to the number of statements spanned by the longest FAR and reassemble IXFFAR.

- A FAR statement may have leading and trailing blanks. A continuation FAR can begin in any column before column 72. Parameters can break off in column 71 and finish on the next statement. Example 9 in Figure 9 is a whimsically coded FAR illustrating these points.

- Embedded blanks are not allowed; that is, between the first nonblank character on the card and the last nonblank before column 72, no blanks are allowed outside of character strings (for example, expressions of the form C'...', specifying keys or delete codes).

- The first item has to be a ddname, except in the case of a NOMESSAGES statement, or a comment statement which must begin with an * in column 1. The rest of the parameters may be coded in any order, as consistency checking is performed after the entire FAR is processed.

- The list of checkpoint identifiers must be enclosed in parentheses, even if it contains a single item.

- All FAR parameters for the same ddname must be coded together (use continuation statements if necessary). IXFFAR treats a new statement for a previously processed ddname as a replacement (not an add-on).

- The DD Statements for all data sets requiring FAR specifications must be coded before the //PMISTOP statement in the Intercomm execution JCL. The DD statement for a duplex file must be within the first 255 file DDs.

  Generalized FAR parameters are summarized in Figure 7 and described in detail in the Operating Reference Manual.

| PARAMETER | FUNCTION |
|---|---|
| ALIAS=ddname | To define an alias for a data set, in order to route I/O operations to the alias data set. The originating ddname will have the FAR attributes of the alias file; no other attribute may be coded on this statement. |
| B37 | Invokes an automatic facility to protect Intercomm from a x37 abend resulting from running out of space on this file. Applies only to BSAM (sequential output) disk files and the Intercomm log (if to disk). Installation specifications are in the <u>Operating Reference Manual</u>. |
| COREINDEX | Requests that the highest-level index of a BISAM file be kept in main storage. This option applies only to files large enough that the index hierarchy goes above the cylinder level. |
| DSN | Causes the specified VSAM KSDS file to be opened with the 'data set name sharing' attribute. Used for a base cluster and associated path(s) when file referenced (updated) via more than one ddname. |
| DUPLEX=ddname | Specifies the ddname of one or more duplex output files. When a duplex output operation is performed, the status code returned to the caller is C'0' if any output operation was successful. Otherwise, the status code from the first unsuccessful operation is returned. |
| ERRLOCK | Requests marking a data set permanently down when any I/O call to the File Handler results in a status code of C'1' or C'9'. Automatically assigned to duplexed files. |
| ICOMBDAMXCTRL | Indicates that Intercomm logic is to be used for BDAM exclusive control, rather than that of the access method. |

Figure 7. Summary of General FAR Parameters (Page 1 of 3)

| PARAMETER | FUNCTION |
|---|---|
| LOCK | Specifies that any request to Select the file will be rejected (Release 10 only). FILE command must be used to unlock the file. |
| LSR | Causes a valid VSAM data set to be connected to the VSAM local shared resources pool at ACB OPEN time. The data set must be a VSAM data set which is currently loaded (LSR cannot be used to load a data set or with the WRITEOVER option) and the resource pool must have buffers large enough to contain the data set's control intervals. Coding LSR forces OPEN=VSAM (see below). See the <u>Operating Reference Manual</u> for implementation details. |
| NCPWAIT | Forces Intercomm into the WAIT state when the number of pending I/Os for a sequential file has reached NCP for that file. Intercomm becomes active again when the first I/O in the series is posted complete. This option is forced for INTERLOG, the Intercomm log data set. |
| OPEN={BASIC }<br>{QUEUED}<br>{BOTH }<br>{VSAM } | Requests that one or two DCBs be opened for the file at startup time, rather than waiting for the first I/O request. The meanings of this subparameter depend on the file organization:<br><br>direct:<br>BASIC -- open BDAM DCB<br>QUEUED -- not applicable<br>BOTH -- not applicable<br>indexed sequential:<br>BASIC -- open BISAM DCB only<br>QUEUED -- open QISAM DCB only<br>BOTH -- open both BISAM and QISAM DCBs<br>sequential:<br>BASIC -- open BSAM DCB only<br>QUEUED -- open QSAM DCB only<br>BOTH -- open both BSAM and QSAM DCBs<br>VSAM:<br>-- open VSAM ACB (all data set types) |

Figure 7. Summary of General FAR Parameters (Page 2 of 3)

| PARAMETER | FUNCTION |
|---|---|
| READONLY | To define an input only data set. Cannot be used with File Recovery FAR options. |
| UPDATEONLY | To define a BISAM data set allowing updates, but not inserts. |
| VSAMCRS | Indicates that a VSAM Shareoption 2 or 4 file will be shared by more than one region in the same CPU and that updates will be performed by at least one region. Intercomm will augment VSAM shared file processing and provide read integrity for Shareoption 2 files and read/write integrity for Shareoption 4 files by means of OS ENQs, QNAME=INTERCOM, RNAME=VSAM-dsn (up to 44 characters). This FAR specification must be coded for the file in question for every region which will share the file. See also the Operating Reference Manual. |
| WRITEOVER | Allows a complete rewrite of an existing physical sequential file (DSORG=PS,DISP=OLD) or a VSAM ESDS. If this option is not specified, any data written to the file will be added at the end of existing data (that is, DISP=MOD assumed). If WRITEOVER and READONLY are specified for the same file, READONLY will be used and no writing to the file will be allowed. That is, READONLY suppresses WRITEOVER. For an empty VSAM ESDS, OPEN=VSAM is forced and READONLY is ignored. |
| XCTL={QISAM }<br>{MULTIREG} | Indicates that ISAM exclusive control updates are performed using QISAM or from multiple regions. These specifications are functionally equivalent, and result in an OS ENQ at the file level. This is the least efficient means of assuring exclusive control, and can be avoided by restricting the updates to BISAM and to within a single region. |

Figure 7. Summary of General FAR Parameters (Page 3 of 3)

## 3.2 FAR Parameters for File Recovery

FAR parameters applicable only to File Recovery are summarized in Figure 8 and described in detail in this section.

| PARAMETER | FUNCTION |
|---|---|
| ddname | Required. Each FAR must begin with a ddname (except the NOMESSAGES FAR). |
| CHECK={IN } {OUT } {INOUT} | Defines how to treat records containing delete flags. IN requests records be checked for delete code when read in. OUT checks BISAM inserts: if an insert fails because of duplicate keys and the record with that key on the file contains the delete code, the insert will be changed to an update. INOUT requests both kinds of checking.* |
| CHECKPOINT=identifier-type= (identifier-list) | To avoid overhead of logging a before-image on frequently updated records, designate as checkpoint records. They will be copied onto the log each time a checkpoint is taken. After-images will still be logged if the file is marked RECREATE. |
| DELETE=nnnn={C'delete-code'} {X'delete-code'} | nnnn is the offset into the record, in decimal. The first byte has offset 0. The delete code may be up to eight bytes long.* |
| RECREATE | After-images logged for recreation. |
| REVERSE={YES } {CRITICAL} | YES is level 1 support recovery guaranteed only after Intercomm abends. CRITICAL is level 2 support recovery guaranteed after hardware failures, system crashes or Intercomm abends. |
| * indicates this option does not apply to VSAM files. | |

Figure 8. Summary of File Recovery FAR Parameters

Each following parameter description lists the parameter name, meaning, error message identifiers (complete messages are detailed in Messages and Codes) and file attribute block flags and fields affected by the parameter. Figure 9 illustrates a typical input FAR data set, which should be referenced along with the descriptions of the FAR parameters. Figure 10 summarizes FAB flags and fields. Dsects for control blocks are contained in the member FRDSECTS. The notational convention used below for specifying Dsect bit settings is:

Control-block-name (field-name) = setting description

ddname

Each FAR, except the NOMESSAGES statement, must begin with a ddname. If a matching dd statement is not present, a message will be sent to the operator and the FAR will be ignored.

FR002I, FR006I

No control block fields are set. The ddname is used to locate the corresponding internal DSCT. After FAR processing is complete, DSCT (DSCTFABX) will contain either a short FAB (two bytes of flags) or the offset to a long FAB in the FAB table.

REVERSE=YES or CRITICAL

Specifies the logging of before-images to provide for file reversal during restart. Before an update is made to the file, the record to be updated will be located (read in if necessary) and logged. The kind of file determines how inserts (adds) are treated as described below:

a) BDAM, fixed-format, no keys - no such thing as an add to this kind of file.

b) BDAM, fixed-format, keys - feedback is requested and the block-ID of the newly added record is logged after the write completes. (From the block-ID, a dummy record is constructed and written out to reverse the add.)

c) BDAM, variable-format - if delete code is provided (see DELETE parameter description) the record's address is logged after the write, as in (b).

d) ISAM - if the ISAM delete option was chosen (OPTCD=L) or a delete code is specified, the key of the insert is logged.

e) VSAM, key-sequenced data set - the key of the newly added record is logged before the PUT is issued and is reversed by getting the old record for update and then issuing a VSAM ERASE request.

f) VSAM, relative record data set - the relative record number of the newly added record is logged (as a key) before the PUT is issued and is reversed by getting the old record for update and then issuing a VSAM ERASE request.

g) VSAM, entry-sequenced data set (direct access by RBA) - reversal is not supported because VSAM does not allow the ERASE request by RBA for ESDS files. Additionally, the RBA specified must always point to an existing record in the data set (refer to IBM's VSAM Programmer's Guide). Therefore, a direct access add implies that a destructive write is to be performed. If file reversal is required, the programmer should first issue a call to GETV with the RBA and the file handler status word set for update mode and then issue a call to PUTV without specifying the RBA. Otherwise, the RECREATE FAR option should be used to recreate the data set from the after-image log.

h) VSAM, entry-sequenced data set (sequential output) - reversal is not supported because VSAM does not allow the ERASE request by RBA for ESDS files. The RECREATE FAR option must be used to recreate the data set from the after-image log. It should be noted that recreation of ESDS files might alter the VSAM RBA assigned to records when using IXFCREAT as opposed to on-line synchronous VSAM processing. Therefore, it is recommended that if application cross-reference files are used to access ESDS files by RBA, care should be taken in the systems design for the off-line recovery of such cross-reference files after the IXFCREAT run (apply after-images) has completed.

If REVERSE=YES is coded, logging is done asynchronously. That is, although LOGPUT is called before the File Handler does the update or insert (except in cases b and c above), there is no guarantee that the entry will actually be on the INTERLOG data set when the File Handler transaction is started. LOGPUT moves the entry into a buffer and writes out the buffer only when it is full. Intercomm closedown or abend flushes the INTERLOG buffers and closes the file. But, if an operating system or hardware failure occurs, there is no cleanup processing, and the set of before-images on INTERLOG may be left incomplete. In particular, if the lost log entries recorded inserts or updates to records previously unchanged, file reversal will be incomplete. Note that Intercomm abend cleanup processing requires the inclusion of the module STAEEXIT in the Intercomm on-line execution linkedit; see Messages and Codes for a detailed description of STAEEXIT processing.

If the risk of system/hardware failure is unacceptable, then code REVERSE=CRITICAL for synchronous logging. This informs LOGPUT not to return to the File Handler until the entry has been written to INTERLOG. Then the worst that can happen if the system crashes is that the log is ahead of the file, so file reversal performs writes reversing updates that were never made, or gets harmless errors deleting records that were not inserted. The trade-off is more overhead. The time it takes to do an output operation on a critical file is added to the time to process the queue of write requests for previously filled log buffers, plus the time to complete the current write to the log, plus the time to write the entry for the critical file.

FR009I, FR018I.

FAB (FABFLGS1)=FABRYES, if REVERSE=YES coded.
FAB (FABFLGS1)=FABRCRIT, if REVERSE=CRITICAL coded.
FAB (FABFLGS1)=FABNOADD, if before-image logging not applicable (inserts)

RECREATE
(Code just that; the format 'keyword=value' is not followed when a keyword will do by itself.) Specifies that after-images be logged to provide for recreation of the file from a backup copy. Before every output operation, the data necessary to reconstruct the call to the File Handler is logged. Logging is always asynchronous.

No error messages.

FAB (FABFLGS1)=FABLGAFT

CHECKPOINT=identifier-type=(identifier-list)
Singles out certain records in a reversible file for special treatment: that is, before-images are logged at check-point time, not when updated. Reversal by definition still works, it means reconstructing the contents of a file at some past checkpoint. As long as it is known what a record looked like at every checkpoint, there is no need to log updates to it. There are two ways checkpointing can reduce overhead. First, by saving writes to INTERLOG: specify checkpointing for records that are regularly updated more than once between checkpoints. Second, by allowing asynchronous logging (see REVERSE=CRITICAL explanation) for files that contain only a few records that must be restored after a hardware failure: checkpoint them and take a chance on the rest. Any number of records may be checkpointed. If RECREATE is specified, the after-image is logged when a checkpoint is taken.

An identifier list may consist of a single item or many items as represented in the descriptions below, but it must always be enclosed in parentheses. Note the following: field means either a character expression, written C'....' (example: C'KEY'), or a hexadecimal expression, written X'....' (example: X'01F3'), address means either a hexadecimal expression or a decimal number (example: 42); number means a decimal number.

The listing of identifiers varies according to type of file. For an ISAM file or a VSAM key-sequenced data set, list the keys of the records to be checkpointed:

CHECKPOINT=KEY=(<u>field</u>,....,<u>field</u>).

For a BDAM file without keys or a VSAM relative record data set, list the VSAM RRN's or BDAM RBN's or relative track addresses (TTR's) of the records to be checkpointed (physical addresses are not handled):

CHECKPOINT=BLK=(<u>address</u>,....,<u>address</u>).

For a BDAM file with keys, a relative track address <u>and</u> a key are required to identify a record. Each two-part identifier is enclosed in parentheses, and a second pair of parentheses encloses the list. Code the address first and the key second:

CHECKPOINT=BLK=((<u>address</u>,<u>field</u>),...,(<u>address</u>,<u>field</u>))

FR003I,FR009I,FR010I,FR012I, FR013I,FR016I,FR017I

FAB(FABFLGS1)=FABCP. FAB(FABFLGS2)=FABCPKEY, for ISAM, VSAM KSDS
FAB(FABFLGS2)=FABCPBLK, for BDAM without keys, VSAM RRDS
FAB(FABFLGS2)=FABCPBKY, for BDAM with keys.
FAB(FABCPCNT)=number of checkpoint records
FAB(FABCPLEN)=length of record identifier, for example, key length of VSAM file.

The list of identifiers starts at FAB(FABCPIDS). The presence of checkpoint records is indicated on INTERLOG by a log record with code X'8F'.

Record checkpointing for a file is skipped when the file is locked or dynamically deallocated (cannot be Selected).

DELETE=offset=flag-value

Specifies a flag that will tell the File Handler a record is deleted. This option does not apply to VSAM files. The first subparameter is the offset into the record of the flag field; the second is the value of the field that means "deleted". The offset is decimal; the first byte of the record has offset 0. If the file is variable format, the record descriptor word is counted as part of the record and an offset less than four is an error. The flag-value can be either a character expression or a hexadecimal expression. In the shorthand developed for the CHECKPOINT description, the parameter is stated:

DELETE=number=field

The flag field may be up to eight bytes long; its length is inferred from the flag-value.

When the file reversal routine, IXFRVRSE, finds that a record has been added to a file from which records cannot be deleted in a way the access method recognizes (that is, variable-format BDAM, ISAM without OPTDC=L), it checks the file's FAB for a delete flag. If a flag is specified, IXFRVRSE reads the record in, moves the flag into the flag field, and writes it back out. A delete flag is a signal to the File Handler; as far as the access method is concerned, the record is indistinguishable from the others in the file. IXFRVRSE does not insert a delete flag if the access method supports deletion. What the File Handler does with flagged records during on-line processing depends on how the CHECK parameter (see description below) is coded.

FR003I,FR011I,FR015I

FAB(FABFLGS1)=FABDL. FAB(FABDLOFF)=offset to flag field.
FAB(FABDLLEN)=length of flag field. FAB(FABDLCDE)=flag-value.
Note: The maximum delete code length is set by the symbol FABDLMAX; to provide for longer delete codes, increase FABDLMAX (in FRDSECTS) and reassemble the file recovery modules.

CHECK=IN/OUT/INOUT

The CHECK parameter specifies to the File Handler how to treat records containing delete flags. This parameter is valid only if DELETE is coded.

If CHECK=IN is coded, records in the specified file will be checked, when they are retrieved by a call to GET or READ, for the presence of a delete flag. If the delete code is found, the File Handler will signal its presence by a return code of 2 in the FHCW (a no record found condition). The record will be in your I/O area. CHECK=OUT applies only to BISAM inserts; if an insert fails because of duplicate keys, the File Handler will issue a READ KU for the record with that key, and, if it contains a delete flag, will replace it with the insert via a WRITE K (update). CHECK=INOUT requests both kinds of checking.

You can specify a delete flag and request checking, without asking for logging, but then you must insert the delete flags yourself; Intercomm inserts them only during reversal.

FR009I,FR010I,FR014I

FAB(FABFLGS1)=FABCHKIN if CHECK=IN coded.
FAB(FABFLGS1)=FABCHKOT if CHECK=OUT coded.
Both flags set if CHECK=INOUT coded.

```
    //ICOMIN  DD   *
1   NOMESSAGES
2   SMLOG,ALIAS=FRLOG
3   DD1,REVERSE=YES,RECREATE
4   DD2,RECREATE
5   DD3,REVERSE=YES,CHECKPOINT=BLK=((X'000000',C'MMMM'),                 X
5                                   (X'000000',C'QQQQ'))
6   DD4,REVERSE=CRITICAL,CHECKPOINT=BLK=(1,2)
7   DD5,REVERSE=YES,CHECKPOINT=KEY=(C'KEY1'),DELETE=4=C'XX',CHECK=OUT
8   DD6,READONLY
9                                             DD7,                       X
9                                           UPDATEONLY
```

Notes:

1. Stops IXFFAR from sending card images to the operator console, must be the first FAR.
2. Causes resource management output, that is, thread dumps and statistics, to go to the same SYSOUT data set as images of the file-recovery log entries (see description of IXFSNAPL for details on FRLOG); SMLOG DD statement omitted from execution JCL.
3. Requests logging of before- and after-images.
4. Requests logging of after-images only.
5. This is a FAR for a keyed BDAM file. Two blocks are to be checkpointed: their keys are MMMM and QQQQ; BDAM will look for both at the beginning of the file (relative track address 0).
6. This is a FAR for a BDAM file without keys or a VSAM RRDS file. Before-images will be logged synchronously. Again, two blocks are to be checkpointed: RBN/RRN numbers 1 and 2.
7. This is a FAR for an ISAM file or a VSAM KSDS file (excluding the DELETE and CHECK options). The record with key KEY1 is to be checkpointed. The characters 'XX' in bytes 4 and 5 of a record mean the record is deleted (ISAM only); File Handler will check for the delete code whenever an insert fails because of duplicate keys.
8. No output operations will be allowed on this file.
9. IXFFAR will concatenate all the nonblanks in the FAR before trying to decode it, so it will see: DD7,UPDATEONLY

Figure 9. Sample FAR Data Set

## 3.3 File Attribute Block (FAB)

If any of the parameters related to file recovery (REVERSE, RECREATE, CHECKPOINT, DELETE, or CHECK), or if x37 abend protection is coded in a FAR, a FAB is built for the file. If only REVERSE or RECREATE is coded, a short FAB is built: two bytes of flags at location DSCTFABX in the file's internal DSCT. If any of the other parameters are coded, a long FAB is built in the FAB table and DSCTFABX contains the offset to the FAB from the beginning of the table. A long FAB starts with the same flag bytes as a short FAB. The Dsect, called FAB, is in the member FRDSECTS.

Figure 10 lists the FAB names and descriptions.

## 3.4 FAB Table

The FAB table is a block of dynamic storage acquired by IXFFAR to hold long FABs and entries that identify aliased files. The address of the FAB table is stored in the header of the internal DSCT area at location FABADD. If there are no long FABs or aliased files, the table is not built and FABADD is set to 0. The table starts with a header (Dsect FABTABLE in FRDSECTS), consisting of two halfwords: FABOFFST is the offset to the first long FAB; FABALCNT is the number of alias blocks, that is, entries that identify aliased files. After the header come the alias blocks, if there are any. One of these blocks (Dsect FABALIAS in FRDSECTS) is built for each aliased file without a DD statement (for the primary file); it consists of the eight-character primary ddname (FABALDD) followed by the address of the internal DSCT (FABADSCT) for the alias file. The long FABs follow the alias blocks as shown in Figure 11.

| Field Name | Offset HEX | Offset DEC | Bytes | Description |
|---|---|---|---|---|
| FABFLGS1 | 00 | 0 | 1 | First Flag Byte:<br>FABIMM: X'80' Short FAB<br>FABRYES: X'40' REVERSE=YES<br>FABRCRIT: X'20' REVERSE=CRITICAL<br>FABLGAFT: X'10' RECREATE<br>FABDL: X'08' Delete flag specified<br>FABCHKOT: X'04' CHECK=OUT<br>FABCHKIN: X'02' CHECK=IN<br>FABCP: X'01' Checkpoint record list specified |
| FABFLGS2 | 01 | 1 | 1 | Second Flag Byte:<br>FABCPBLK: X'80' Checkpoint list for unkeyed BDAM file, VSAM RRDS<br>FABCPKEY: X'40' Checkpoint list for ISAM file, or VSAM KSDS<br>FABCPBKY: X'20' Checkpoint list for keyed BDAM file<br>FABNOADD: X'10' No Logging of before-image when a record is inserted in an ISAM file (see error message FR018I)<br>FABNOAFT: X'08' FABLGAFT set by Backout-on-the-Fly<br>FABNCPWT: X'04' Force hard wait when no. I/O=NCP |
| FABDLLEN | 02 | 2 | 1 | Length of delete flag field. |
| FABCPLEN | 03 | 3 | 1 | Length of checkpoint record id. |
| FABDLOFF | 04 | 4 | 2 | Offset to delete flag field. |
| FABCPCNT | 06 | 6 | 2 | Number of records in checkpoint list. |
| FABDLCDE | 08 | 8 | 8 | Delete flag. |
| FAB37COM | 10 | 16 | 8 | B37 companion ddname. |
| FAB37ECB | 18 | 24 | 4 | B37 PMIWTOR ECB. |
| FAB37SYN | 1C | 28 | 4 | B37 synchronous ECB. |
| FABCPIDS | 20 | 32 | - | Beginning of list of checkpoint ids. |

Figure 10. File Attribute Block

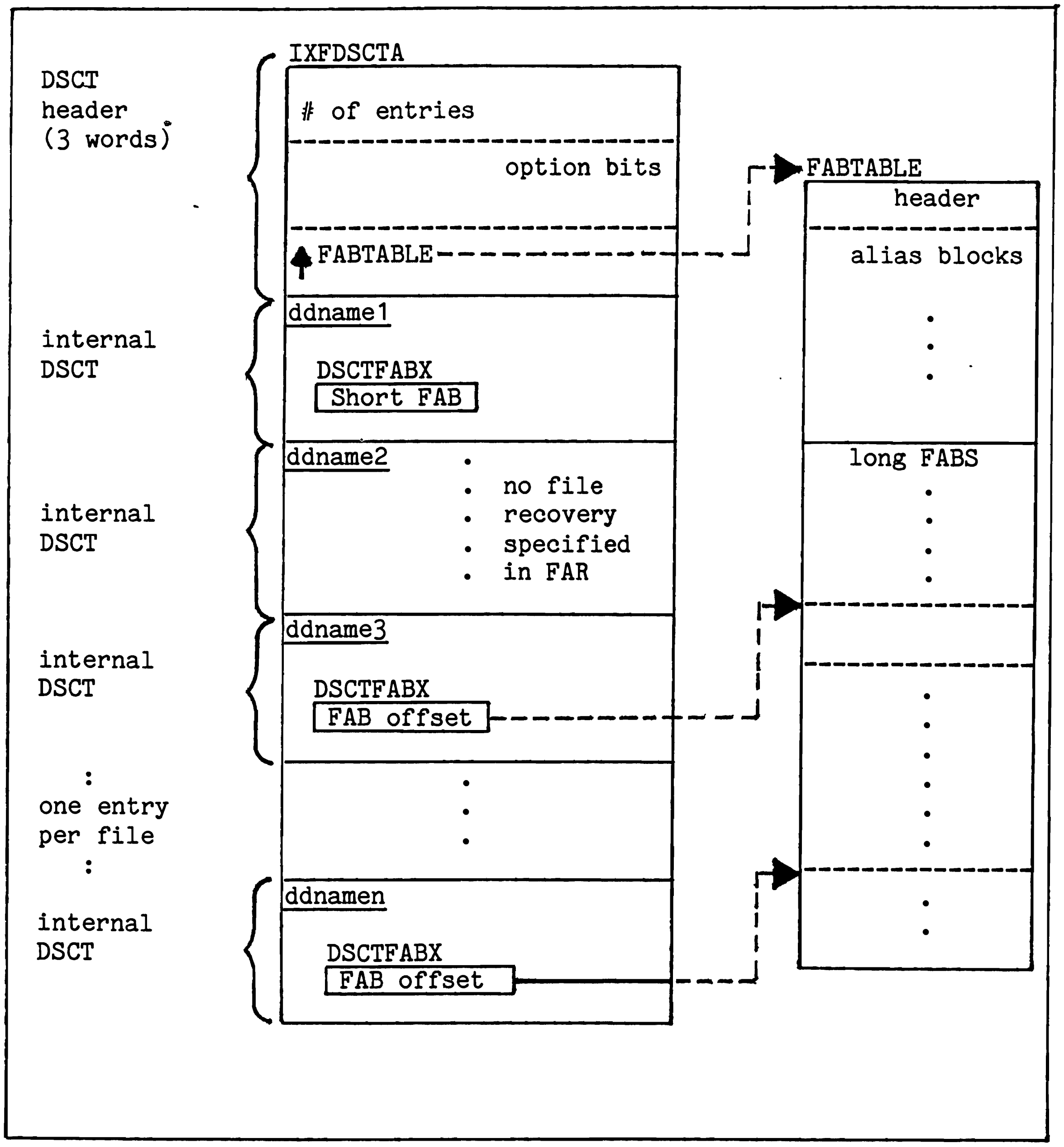

Figure 11. FAB Table

# Chapter 4

# INSTALLING FILE RECOVERY

## 4.1 Introduction

This chapter begins with a description of the File Recovery modules, and continues by giving the assemblies, linkedit changes, and execution JCL needed for the following jobs:

A - running Intercomm in startup mode with message logging, checkpointing, and logging of before- and after-images
B - restarting Intercomm with file reversal
C - recreating a destroyed file from a backup copy, off-line, using logged after-images.

This discussion is meant to be self-contained but not comprehensive; some material on restart is included, but nothing on data base recovery. Please refer to the Operating Reference Manual for more specifics on message restart and checkpointing and the DBMS Users Guide for data base recovery implementation.

Under Release 10 of Intercomm, if the Automated Restart facility is used, jobs A and B are combined. The EXEC statement PARM (STARTUP or RESTART) is ignored (if coded) and execution of restart depends on the successful (NRCD or IMCD closedown) or unsuccessful (abend or CPU failure) conclusion of the previous execution of the Intercomm region. For implementation see the Release 10 Operating Reference Manual.

Detailed procedures for calling File Handler service routines are discussed in the Intercomm Programmer's Guides. File Handler service routines are called by application programs, passing (as one of the parameters) the File Handler Control Word (FHCW). The FHCW is a fullword control field, used for communication between the File Handler and the calling subsystem. Prior to calling each service routine, the subsystem must clear the FHCW with blanks or initialize it with a predefined request code, as applicable. Several File Recovery modules use the FHCW to enable special functions of a service routine, such as before-image logging. Upon completion of the request, the File Handler will communicate the status of the operation back to the subsystem.

## 4.2 FILE RECOVERY MODULES

There are six modules in the File Recovery facility as summarized in Figure 12, along with other modules required for checkpointing, message restart, and coordinated file recovery.

| Member | CSECT | Job | Residency | Function |
|---|---|---|---|---|
| IXFFAR | IXFFAR | A, B, C | Startup overlay with IXFMON00 | FAR processor |
| IXFLOG | IXFLOG | A, B | Resident | Constructs file recovery log entries |
| IXFCHKPT | IXFCHKPT | A, B | With CHCKPTSS | Logs checkpoint records |
| IXFRVRSE | IXFRVRSE | B | With LOGPROC | Applies before-images |
| IXFCREAT | IXFCREAT | C | Resident (batch job) | Applies after-images |
| IXFSNAPL | IXFSNAPL | B, C | With IXFRVRSE for Job B, resident for Job C | Prints file recovery log entries |
| DBCHKDSP | CHECKPT | A, B | Resident | Sends message to CHCKPTSS |
| CHECKPT3 | CHECKPTO | A, B | Resident | Checkpoint file record writing |
| CHCKPTSS | CHCKPTSS | A, B | Resident or overlay | Quiesces sub-systems |
| MSGAC | MSGAC000 | A, B | Resident | Message accounting |
| LOGPUT | LOGPUT | A, B | Resident | Message logging |
| LOGPROC | LOGPROC | B | Startup overlay | Message restart |
| RESTORE3 | RESTORE | B | Startup overlay | Resets check-point fields |

Figure 12. Summary of Message Restart/File Recovery Modules

The IXFFAR routine reads the FARs and builds the FABs as discussed in the previous chapter. Logically, it is part of the File Handler initialization Csect, IXFMON00, in the sense that it is called only from IXFMON00, and from there only once. (IXFMON00 is in IXFHND00.) IXFFAR must be included in all three jobs; the FARs specify logging in jobs A and B, which files get reversed in job B, and which files get recreated in job C.

The IXFLOG routine constructs file recovery log entries and passes them to the entry point LOGPUTF in LOGPUT, to be written to INTERLOG. A file recovery log entry is divided into header and text like a message status log entry. The header is the same length as a message header, but the only common fields are those used for the length, date, time and log code. (See Figure 3 for a comparative description of the header fields.) The file recovery header contains the ddname, the BDAM block-ID, the key length, and the record length. The zone bits of the log code indicate whether the entry represents a before-image or an after-image; the numeric bits identify the output operation. A key (or VSAM RRN), a record, or both, may follow the header depending on the type of entry; see Figure 5 and 6: FORMAT.

The File Handler calls IXFLOG just before it issues a VSAM, BDAM or ISAM output macro (unless it is a WRITE DAF, which is logged after completion because the feedback address is part of the log entry). The File Handler passes a code to identify the macro issued. The code is arbitrary--a number between 0 and 18 (see Figure 5 for the numbers associated with each macro)--it identifies the sources of log-entry information for different macros: if a record in an ISAM file is updated with a PUTX, the before-image address is in the DSCT; if it is updated with WRITE K, the address is in the DECB, for a VSAM file the address is in the RPL, and so on.

IXFLOG is normally linkedited in the resident portion of Intercomm. It is reentrant and Link Pack eligible.

The IXFCHKPT routine handles file checkpointing (see the description of the CHECKPOINT parameter). It is called by the checkpoint subsystem CHCKPTSS. The first thing IXFCHKPT does is log a header containing the log code X'8F', a signal to the reversal routine, IXFRVRSE, to release files and reset internal switches. Since restart with file reversal processes the log tape back to an Intercomm checkpoint, one of these headers will always be the last log entry IXFRVRSE examines, and restart will complete without SELECTs outstanding. The rest of the header is meaningless.

Next, IXFCHKPT logs the records specified in the FAB checkpoint lists, using a special File Handler option; if a READ is issued with L in the second byte of the File Handler Control Word, the File Handler will cause the record to be logged after it is read in, by calling IXFLOG with the appropriate macro code. Ordinarily, of course, only output operations are logged. Read-time logging keeps checkpoint I/O to a minimum, and the L gives IXFLOG a way to isolate a checkpoint record request from a regular update. All the subsystems that change reversible files are supposed to be quiesced while checkpoints are taken; an attempt to update a reversible file during checkpointing causes an error message (FR030A) to be issued.

The IXFRVRSE routine uses logged before-images to reverse updates and additions to files. LOGPROC, the routine that restarts messages from the old log, calls IXFRVRSE every time it finds a before-image entry. IXFRVRSE in turn prepares a parameter list and calls the File Handler to do the I/O. Reading the log backwards, there is no way to detect redundant before-images until it is too late; the earliest before-image, that is, the last one found on the tape, is the one that counts. One record may be rewritten several times during reversal; however if Intercomm checkpointing is used properly it should be fairly unusual to encounter two before-images of the same record in a single checkpoint interval.

To reverse an add to a keyed BDAM file, IXFRVRSE writes a dummy record. A dummy record is identified by X'FF' in the key. Since the File Handler does not allow a change to the key of a BDAM record, another option was added. If WRITE is called for a keyed BDAM file with I in the second byte of the File Handler Control Word, the File Handler issues a WRITE DI. An I specifies exclusive control; J means normal read. A block-ID and a key must still be supplied, but the block-ID will be taken as the precise address of the record to be changed, not the starting point of a search. The key will replace the key of the record at that address. For completeness, reading by ID from a keyed file is also supported. The address of an area for a key must be supplied in the parameter list; it is filled in by the READ like the I/O area. See the appropriate Intercomm Programmer's Guide (COBOL, PL/1, Assembler Language) for additional details.

The IXFCREAT routine applies logged after-images to backup copies of files. It runs as the main program in an off-line job, reading the log forward and calling the File Handler to write after-images to the files being recreated.

The Intercomm log tape may represent several consecutive executions of the system where the restart log became the live log following the restart operation. For example, in one day's time, the system may be initiated, terminated with IMCD, restarted and subsequently terminated with an abend. Assume a backup copy of a file was made after the IMCD closedown and that file recreation is necessary following the abend. Given a log representing several executions of Intercomm, starting recreation with the first after-image will mean pointlessly performing updates and additions that are already represented in the backup. For this reason, IXFCREAT incorporates a feature to skip to a specific log entry to begin recreation. For example, each time Intercomm starts up, it issues a message RECREATION STARTPOINT=, followed by a date and time; at the same time it creates a startup log entry with a special code and the same date and time. To make IXFCREAT reposition the log to the start of a particular run, put the date and time from the RECREATION STARTPOINT= message of that run in the EXEC statement PARM field of the recreation job, exactly as it appears in the message. A specific starting date and time are requested by coding:

```
PARM='yy.ddd,hhmmssth'
```

IXFCREAT will read the tape, ignoring after-images, until it finds a log entry with a date and time greater than or equal to the parm value. A starting day of 001 of the current (or previous) year, and a starting time of zeros (00000000), will force all after-images to be processed.

The IXFSNAPL routine formats and prints file recovery log entries. It expects two parameters, the address of the log entry and the address of either a selected DSCT for a QSAM data set, or the address of an open DCB for a QSAM or BSAM data set. IXFSNAPL determines what kind of control block has been passed and PUTs, WRITEs, or calls the File Handler accordingly. The data set must be fixed-format and have an LRECL of at least 120. There are CALLIFs to IXFSNAPL in IXFRVRSE and IXFCREAT; the output goes to a SYSOUT data set called FRLOG (see setup directions for Jobs B and C below).

IXFCREAT defaults to snap all after-images applied. However, it may optionally snap only after-images that are unsuccessful due to I/O or logic errors. This is specified by adding SNAP=ERR following the date/time on the EXEC statement PARM field as follows:

```
PARM='yy.ddd,hhmmssth,SNAP=ERR'
```

If snaps are not desired, then specify SNAP=OFF or omit the FRLOG DD statement. However, this is cautioned against because critical information is printed for errors; for VSAM files, the feedback code is provided.

Figure 13 depicts IXFSNAPL output. If an after-image cannot be applied due to an error, an appropriate error message with the hexadecimal representation of the File Handler Control Word will appear on the line following the snap.

End of job statistics are produced and written to FRLOG (if present) for run controls. The following information is reported:

```
NUMBER OF RECORDS APPLIED :   x,xxx,xxx
NUMBER OF RECORDS IN ERROR:   x,xxx,xxx
```

## 4.3 Job A - Intercomm with File Recovery

Note - if all you want to log is after-images, to allow recreation but not reversal, many of the steps in the following setup procedure should be skipped; the steps that also apply to systems with reversible files are indicated by an asterisk.

```
**PRE-IMAGE*** DDNAME=ISV      ,ENTRY LEN=  70,BLK ID/RBA=000000/00000000,LOGCODE=80,MACRO=00,KEYLEN=  4,TEXTLEN=  24
TYPE=  PUTX/WRITE K                                                                  DATE: 83042  HOUR:  17490790
KEY=   F0F1F0F6                                                                     0106
TEXT=  00180000E5F0F1F0 F6D9C5C3D6D9C45C F0F1F0F6F0F1F0F6                           Q  V0106RECORD*01060106
-------------------------------------------------------------------------------------------------------------------

**POST-IMAGE** DDNAME=ISV      ,ENTRY LEN=  70,BLK ID/RBA=000000/00000000,LOGCODE=90,MACRO=00,KEYLEN=  4,TEXTLEN=  24
TYPE=  PUTX/WRITE K                                                                  DATE: 83042  HOUR:  17490790
KEY=   F0F1F0F6                                                                     0106
TEXT=  00180000E5F0F1F0 F6D9C5C3D6D9C45C F0F1F0F6F0F1F0F6                           Q  V0106RECORD*01060106
```

**PRE-IMAGE***: a file-recovery before-image entry (since the zone digit of the log-code is 8).

**POST-IMAGE**: a file-recovery after-image entry (the log code zone digit is 9).

DDNAME=: ddname of the file. This is the MSGHDD field in the header. This field will be blank for a checkpoint entry (log code=8F).

ENTRY LEN=: length of the log entry; this is the value of MSGHLEN in the header.

BLK ID/RBA=: for BDAM, this field contains the RBN or TTR of the record altered, in hex; for ISAM/VSAM the field contains zeros. Header field MSGHBKID.

LOGCODE=: header field MSGHLOG.

MACRO=: code indicator for type of file access macro used (see Figures 5 and 6). Header field MSGHMACR.

KEYLEN=: length of the key; zero if the log entry does not include a key. Header field MSGHKEYL.

TEXTLEN=: length of the record image; zero if the log entry does not include a record image. Header field MSGHTXTL.

TYPE=: access method macro issued by File Handler to update or add the record. If two macros generate entries with the same log code, both macros are listed in this field; use the macro code to determine which macro was used.

DATE: date logged in message header (yyddd format) from header field MSGHDAT.

HOUR: time stamp from message header (hhmmsstt format). Header field MSGHTIM.

KEY=: value of the key, hex on the left and EBCDIC on the right. If omitted, the log entry does not contain a key.

TEXT=: value of the record, hex on the left and EBCDIC on the right. If omitted, the log entry does not contain a record.

Figure 13. Sample IXFSNAPL Output

### 4.3.1 FAR, SCT, and SPA Specifications

1. Code a FAR for each on-line file to be logged.

2. Create the FAR data set, ICOMIN. Generally, this means coding statements to define a SYSIN data set:

```
//ICOMIN   DD *
     far
       .
       .
       .
       .
     far
/*
```

   and inserting it somewhere after the EXEC statement for the Intercomm execution step. The FARs can be put in a PDS member or a sequential data set of their own, if desired. In any case, include a DD statement for ICOMIN.

   As released, IXFFAR treats all FAR errors except missing DD statements (FR006I) as fatal; that is, after IXFFAR detects a coding error it will analyze the rest of the FARs to turn up all the errors it can, but it will stop building FABs and abend when ICOMIN is exhausted. It is a waste of time initiating Intercomm just to find out you omitted a quote in a character expression. In creating a FAR data set for the first time it might be a good idea to use the file recreation program (Job C) to check it: dummy AFTERIM, include DD statements for all the files for which you have coded FARs and run with the new FAR data set. Job C will run in about 30K.

3.* Identify those subsystems which update reversible files. File reversal will not operate correctly unless output activity on reversible files is stopped while an Intercomm checkpoint is taken; otherwise, there is no way of ensuring that all the file changes caused by the first processing of the message have been nullified when a message is restarted. Output activity is stopped by flagging the subsystems that cause changes to reversible files; these subsystems are quiesced by the checkpoint subsystem. The flagging is done by a parameter in the SYCTTBL macro; if a subsystem alters a file that has REVERSE=YES in its FAR, the subsystem's SYCTTBL macro must specify RVFILE=YES. Update the applicable SYCTTBLs and reassemble the Subsystem Control Table (member INTSCT).

4.* It is not sufficient to just flag subsystems that call the File Handler to alter reversible files. For example, if F represents a reversible file, and A and B represent subsystems, A does not access F itself, so it is not flagged, but A sends a message to B, and B updates F. Suppose A continues processing after it sends the message to B, and while A is still active, B completes, and a checkpoint is taken. Then the system crashes and Intercomm must be restarted. Subsystem A's message in progress would be restarted, Subsystem B's update may be done twice.

There are a number of ways to prevent situations like this:

a) recode A so that sending the message to B is the last thing it does;

b) leave A as it is, but make A and B mutually exclusive, that is put them in different overlay groups in overlay A, so B cannot start until A is finished

c) flag A, that is, code RVFILE=YES in the SYCTTBL macro for A, so no checkpoint will be taken until A is finished. (This approach just pushes the problem back one level; if A is invoked by a message sent from another subsystem, C, you will have to do something about C, too.)

d) best of all, perform all updates in A.

5.* Quiescing the subsystems flagged RVFILE=YES is the job of an Intercomm subsystem CHCKPTSS, also used for data base support. Code a SYCTTBL macro for it in INTSCT (or user-coded COPY member USRSCTS) with the following parameters:

```
SYCTTBL  SUBC=Q,SBSP=CHCKPTSS,LANG=NBAL,MNCL=1,           X
         NUMCL=1,LSYNCH=YES,RESTART=NO,PRTY=0,            X
         TCTV=27962
```

CHCKPTSS can be resident or assigned to an overlay. If CHCKPTSS is placed in an overlay area, it should share the area with subsystems that do not update reversible files, thus competing for the area with subsystems that do.

A SPALIST parameter, GENSW, must be coded to identify the available checkpoint areas. See also the SPALIST parameters CKPTLIM and TCHP, described in Basic System Macros.

6. Specify number and size of INTERLOG buffers to be obtained for LOGPUT; there are SPALIST parameters specifying the number of buffers to obtain (LGNUM) and the <u>average</u> buffer length (LGBLK). These numbers should be chosen with care, because if logging requests accumulate faster than LOGPUT can handle them, the performance of the whole system degrades. LGBLK should be large enough so that any frequently generated log entry (message or file recovery image) will fit in a buffer. Logging an entry bigger than LGBLK effectively ties up two of LOGPUT's buffers; the active, partially filled, buffer to be written is queued, storage for a temporary buffer to hold the large entry is obtained, then another of LOGPUT's own buffers is marked full so its control fields can be used (a buffer contains a DSCT, a save area, and chain pointers as well as space for log entries).

   <u>NOTE</u>: The more synchronous the logging, the smaller and more numerous the buffers should be (recall that synchronous logging, requested for subsystems or terminals by coding LSYNCH=YES and for files by coding REVERSE=CRITICAL, means LOGPUT does not return until the log buffer is written). A synchronous logging request causes the buffer containing the entry to be queued to be written immediately, whether or not the buffer is full. Any leftover space is wasted. So there is no point in making buffers big enough to hold ten messages, say, if one log request in five is synchronous. Tuning information for optimizing logging performance is provided by the System Tuning Statistics facility described in the <u>Operating Reference Manual</u>.

   File Recovery log entries will tend to come in bursts; in logging before- and after-images each update to a file produces two entries, and one thread may do several updates. Check for bursts like this and make sure LOGPUT has enough buffer space to handle them. The size of a file recovery log entry is generally 42 (the header length) + the key length + the length of the logged record. Some are shorter. (See Figures 5 and 6.)

7. Reassemble INTSPA (SPALIST macro).

8.* A consequence of the remarks in step 3, above, is that a reversible file must never be changed by an Intercomm module; Intercomm system modules do not have anything like SCT specifications to quiesce activity on system files, such as queues. Verify that all reversible files are the exclusive property of application subsystems. (RECREATE can be specified for any file.)

## 4.3.2 Linkedit

The following steps are automatic if the applicable parameters are coded for the ICOMLINK macro assembly to generate the Intercomm linkedit control statements (see Basic System Macros).

9.* Include DBCHKDSP. This is a small resident module that schedules checkpointing by sending a message to CHCKPTSS (see step 3), then dispatches itself for the checkpoint interval, SPATCHP. The Csect name of DBCHKDSP is CHECKPT, which is also the name of a Csect in the standard Intercomm checkpoint module, CHECKPT3; therefore, include DBCHKDSP before CHECKPT3. Note: if you want to assign CHCKPTSS a subsystem code other than Q, you have to change the MVI in DBCHKDSP that sets up the receiving subsystem code in the message header (00086000).

10.* Include CKCKPTSS; if it is to execute in Overlay Region A, follow INSERT CHCKPTSS with INSERT IXFCHKPT.

11.* Include CHECKPT3. CHECKPT3 must be included even though one of its Csects is overridden by DBCHKDSP. The other Csect, CHECKPT0, is the one that actually writes the checkpoint and it is invoked by a CALLOVLY from CHCKPTSS.

12. Three file recovery modules are needed for Job A:
Include IXFFAR, IXFLOG, and IXFCHKPT. IXFLOG is Link Pack eligible.

13. Follow the startup Overlay A INSERT card for IXFMON00 with INSERT IXFFAR, if an overlay structure is used.

14.* Make sure the Csect CHECKPT0 is inserted in the transient overlay region (if used).

15. Include MSGAC (message accounting processing) and, if logging to disk, IXFB37.

## 4.3.3 Execution JCL

16. Code PARM='STARTUP...' on the EXEC statement.

17. Code the INTERLOG DD statement.

a) DCB=(...,NCP=number-equal-to-LGNUM,...)
Recall that LOGPUT writes the log using BSAM from buffers acquired by the startup routine (see step 6); this DCB parameter allows LOGPUT to start writes on all its buffers before entering a hard wait state to wait for the first write to complete. See the description of the FAR

parameter NCPWAIT in Figure 7. The greater the number of specified buffers, the smaller the likelihood of a periodic hard wait state.

b) DCB=(...,OPTCD=C,...)
Requests chained scheduling, that is, chaining of I/O blocks (IOBs) when more than one write request is queued.

c) Verify that BUFNO is not specified in the DCB parameter list. LOGPUT does not use OS buffers.

d) BLKSIZE specifies the actual maximum length block that LOGPUT may use. You can specify BLKSIZE=32760 to make sure everything gets logged, but if the log file is ever used in a restart run you do have to specify the largest block size written (unless you can spare 32K per buffer). It is preferable to settle on a single figure for Jobs A, B and C. On the one hand, an entry larger than BLKSIZE will not get logged in Job A, but on the other, Jobs B and C will run; both Jobs B and C abend if the input log file (RESTRTLG in Job B, AFTERIM in Job C) contains a block bigger than BLKSIZE. The minimum block size is LGBLK+4. The extra four bytes are for the block descriptor word. The largest block size should account for the largest possible message (including header) + 4, or the largest file recovery image (plus key) plus message header +4 (see step 6 above), whichever is larger.

e) The DD statement for logging to tape looks like:

```
//INTERLOG DD DSN=INTERLOG,DISP=(NEW,KEEP),
//            UNIT=unit,VOL=SER=abc,LABEL=(,BLP),
//            DCB=(DSORG=PS,RECFM=VB,BLKSIZE=blksize,
//            OPTCD=C,NCP=lgnum,LRECL=blksize-4)
```

f) The DD statement for logging to disk looks like:

```
//INTERLOG DD DSN=INTERLOG,DISP=(NEW,KEEP),
//            UNIT=unit,VOL=SER=volser,
//            SPACE=(CYL,(primary)),
//            DCB=(DSORG=PS,OPTCD=C,NCP=lgnum,
//            RECFM=VB,BLKSIZE=blocksize,LRECL=blksize-4)
```

If disk log file flop/flip (x37 abend protection) is desired, see the <u>Operating Reference Manual</u> for additional installation considerations.

18. Include a DD statement for ICOMIN, plus the FAR data set, if ICOMIN is SYSIN.

19. Include a DD statement for CHEKPTFL. This direct access file is used by CHECKPTO (see step 11) to store checkpoint information. It must be formatted before the run by CREATEGF, with a block size of at least 100, and at least 40 RBNs. See the Operating Reference Manual for precise calculation of block size.

## 4.4 Job B - Intercomm Restart with File Reversal

Since file recovery at Intercomm restart is coordinated with message restart and checkpointing, it is necessary to review the discussion of the installation of these features, particularly log recovery via the ICOMFEOF utility, as described in the Operating Reference Manual.

### 4.4.1 FAR, SCT, SPA Specifications

Same as Job A, steps 1-8. There is one new point:

1. Be careful about changing the FAR data set between failure and restart. The REVERSE parameter serves a double purpose in a restart run: in addition to identifying the files for which before-image logging is to be done it tells IXFRVRSE which files to reverse. Even if before-images for a file exist on the input log, they will not be applied unless the file is defined as reversible for the restart run.

### 4.4.2 Linkedit

Follow directions for Job A and also ensure that the following are in the Intercomm linkedit:

2. LOGPROC and its subroutines READBACK and INTDBLOK, the routines that read the log backwards and restart messages. LOGPROC should go in the startup overlay.

3. IXFRVRSE (follow INSERT LOGPROC with INSERT IXFRVRSE).

4. IXFSNAPL, if you want a record of the before-images applied. IXFSNAPL may be inserted in the same overlay as LOGPROC and IXFRVRSE.

5. RESTORE3, the routine that retrieves checkpoint information from CHEKPTFL (see Job A, step 19); also insert RESTORE (the Csect name), in the startup overlay.

6. REQONDDQ (and USRSEREX), if serial restart of messages which update files is desired. See the Operating Reference Manual.

## 4.4.3 Execution JCL

7. Code PARM='RESTART,...' on the EXEC statement.

8. If logging to disk, see Job A, step 17-item f for the JCL and other considerations, and the Operating Reference Manual for definition of the RESTRTLG DD statement.

9. If logging to tape, restart may use one log tape for both input during restart and output after Intercomm goes live; if DISP=MOD is specified for INTERLOG, log entries for the run that failed are left intact; the tape is repositioned after the last entry, and receives entries reflecting the requeuing of messages and application of before-images during restart, as well as fresh log entries when Intercomm goes live. If DISP=NEW is specified, the entries for the run that failed will be destroyed, including file after-images.

   In the following examples, asterisks refer to the discussion of the INTERLOG DD statement in Job A. The LABEL parameters in the illustrations assume standard labels. If unlabeled tapes are used, code LABEL=(,BLP) or LABEL=(,NL) in both DD statements. Make the volume serial numbers identical, so that both data sets are assigned to the same drive. LABEL=(,SUL) is recommended. This will cause the EOV exit to be taken in the File Handler and will prevent time-outs caused during mounting of a new log volume.

   Input Log (note that RECFM=U is required):

```
//RESTRTLG    DD      DSN=anyname,DISP=(OLD,PASS),
//                    UNIT=unit,VOL=SER=abc,LABEL=(2,BLP),
//                    DCB=(DSORG=PS,RECFM=U,BLKSIZE=blksize*)
```

   Output Log:

```
//INTERLOG    DD      DSN=anyname,DISP=({NEW},KEEP),
                                        {MOD}
//                    UNIT=unit,VOL=SER=abc,LABEL=(,SUL),
//                    DCB=(DSORG=PS,RECFM=VB,BLKSIZE=blksize*,
//                    LRECL=blksize-4,OPTCD=C,NCP=lgnum*)
```

10. LOGPROC uses a temporary disk data set that holds messages to be restarted. The data set is variable-format, with one message per block, so its block size must be equal to the length of the longest message that can be queued for a restartable subsystem. File recovery log entries are not sent to this data set. It is created by LOGPROC; no preformatting is necessary. In the following, m stands for the maximum message size.

```
//LOGDISK      DD          UNIT=direct-access-device,
//                         SPACE=(m,(primary,secondary),RLSE),
//                         DCB=(DSORG=DA,BLKSIZE=m)
```

11. If IXFSNAPL has been included (see step 4, above), insert the following DD statement:

```
//FRLOG  DD  SYSOUT=A,
//           DCB=(DSORG=PS,BLKSIZE=120,RECFM=FA)
```

Error messages issued by IXFRVRSE will also be printed on FRLOG, so you will be able to associate errors with the log entries that caused them (see Figure 13).

## 4.5 JOB C--RECREATING FILES OFF-LINE

1. Define a FAR for each file you want to recreate, containing the ddname and one parameter, RECREATE.

2. The linkedit step of a recreation run is as follows:

```
//LINK         EXEC      PGM=IEWL,PARM='LIST,LET,XREF,NCAL'
//SYSUT1       DD        UNIT=sysda,SPACE=(CYL,(2,1))
//SYSPRINT     DD        SYSOUT=A
//SYSLIB       DD        DSN=module-library,DISP=SHR
//SYSLMOD      DD        DSN=&&TEMP,DISP=(NEW,PASS),UNIT=sysda,
//                       SPACE=(CYL,(1,1,1))
//SYSIN        DD        *
               INCLUDE   SYSLIB(IXFCREAT)
               INCLUDE   SYSLIB(IXFHND00)
               INCLUDE   SYSLIB(IXFFAR)
               INCLUDE   SYSLIB(IXFHND01)
               INCLUDE   SYSLIB(IXFSNAPL)        optional
               INCLUDE   SYSLIB(BATCHPAK)
               NAME      FCREATE
```

3. Assuming the linkedit step illustrated in step 2 above, the execution step will start with these two statements:

```
//GO            EXEC         PGM=FCREATE[,PARM='....']
//STEPLIB       DD           DSN=&&TEMP,DISP=(OLD,DELETE)
```

   See the description of IXFCREAT in this section for an explanation of the PARM field on the EXEC statement.

4. Include a DD statement for each file to be recreated; use the same DCB/AMP parameters as coded for Intercomm execution.

5. Define the FAR data set:

```
//ICOMIN        DD           *
ddname1,RECREATE
                 .
                 .
                 .
ddnamen,RECREATE
/*
```

6. Include a DD statement for FRLOG if IXFSNAPL was linkedited in the load module. (See step 11 under Job B.)

7. Include a DD statement with ddname AFTERIM for the log file containing the after-images. This file will normally consist of one or more log tapes or disk data sets produced by Intercomm runs (note that RECFM must be VB):

```
//AFTERIM       DD           DSN=anyname,DISP=OLD,UNIT=unit,
//                           LABEL=(2,BLP),VOLUME=SER=vvvvvv,
//                           DCB=(DSORG=PS,BLKSIZE=bbbb,        Note 1
//                           LRECL=blksize-4,RECFM=VB)
```

Note 1: See Job A, step 17-item d, for BLKSIZE value.

If the file takes up more than one reel, you can specify a list of volume serial numbers or, if on disk, concatenate DD statements. Concatenation of intermixed tape and disk files is not supported. Concatention must be in ascending date/time sequence.

# Chapter 5

# BACKOUT-ON-THE-FLY

## 5.1 INTRODUCTION

Backout-on-the-Fly is a dynamic facility whereby chosen files will be restored to the condition they were in prior to the failure of a message-processing thread. Only those records altered by the failing thread are restored. Any updates concurrently accomplished by other threads, to other records, remain untouched. Failure of a thread includes time-outs, program checks, and a specific request for Backout-on-the-Fly by a subsystem return code with value 912.

The failure of a thread which has updated a file leaves the integrity of the file in doubt. The value of the integrity of the file may be such that the user does not wish to continue processing until it is restored. To prevent future processing of the file in question, the user may stop the failing subsystem from new work via the CANC=STOP SYCTTBL macro specification. The user may, in this case, provide his own USRCANC program to respond to messages destined for the failing subsystem. However, other subsystems might still try to access that file. Therefore, a special procedure to close and lock the file to prevent all future accesses, should be initiated, utilizing a facility such as the FILE command of the General Purpose Subsystem (GPSS). However, such drastic measures may significantly degrade the goals of running the on-line system. One is then left with the final option of closing the on-line system and performing file reversal or recovery and restarting the system.

Implementation of Backout-on-the-Fly greatly reduces the probability of necessitating a hard restart or recovery of the on-line system. The facility should be used to forestall the necessity of performing file recovery or reversal on those files whose integrity is vital. Backout-on-the-Fly performs its operation concurrently with all other Intercomm processing: even other threads accessing the file in question continue to process. The CANC=STOP parameter may be omitted from SYCTTBLs requesting Backout-on-the-Fly. Also, Intercomm may be brought up using the STARTUP parameter.

Before- and after-images of updated records, after-images of added records, and before-images of deleted records are maintained on an Intercomm DDQ during thread processing. The only limit to the number of updates that may be done by a thread, or concurrently within the system, or to the number of files each thread may access, is the amount of space available on the DDQ file used by Backout-on-the-Fly.

Any user file may utilize Backout-on-the-Fly provided it is specified as REVERSIBLE via a File Attribute Record (FAR) specification. Files so specified will be dynamically backed-out upon thread failure for all subsystems specifying the Backout-on-the-Fly feature via the SYCTTBL parameter BACKOUT=YES.

In the event of a thread failure, the console operator is notified that a reversal has begun. Each entry on Backout-on-the-Fly's DDQ is read in reverse chronological order. For updated records, the record in the on-line file is read and compared character for character with the image of the update done by the thread. If they match, the before-image is applied. For added records, a similar comparison is performed and, if identical, the record is deleted. For deleted records the file is read to insure a new record with the same key (RBN) had not been added. If no such record exists, the record is added back into the file. The entire process will continue until the associated Backout-on-the-Fly DDQ becomes exhausted, at which time the operator will be so informed. If the process was successful, the file is as it was before thread processing began.

Backout failure can be caused by one of the following:

- A record mismatch while comparing the after-image to the current file. This indicates that another thread has updated the same record in the interval following its update by the failing thread and the BOF attempted recovery.
- A record deleted by the failing thread is found in the on-line file.
- A record added is not found or is found to be altered.
- Running out of space in the thread-associated DDQ.
- A low core condition preventing Backout-on-the-Fly from acquiring work areas.
- An I/O error on the DDQ or user file.

Failure for any reason will be broadcast to the operator who will then be asked for information on how to proceed. As this is a potentially critical condition, with file integrity lost, the entire network will wait for this operator-prompted direction.

Backout-on-the-Fly maintains the before- and after-images on a blocked, single-retrieval transient DDQ. If thread completion is unsuccessful, the queue is transferred to an unblocked semipermanent queue which is then read backwards. Blocking not only reduces I/O overhead, but if file access is small and/or DDQ block size is large, a block may be kept in core and never written to auxiliary storage. For example, if a thread updates a record of 100 bytes and the blocksize of the DDQ is 300 bytes, both the before- and after-images can remain in the 300-byte buffer. Assuming the thread completes successfully, the buffer will be freed and no I/O access will have occurred.

## 5.2 IMPLEMENTATION PROCEDURES

The following preparatory steps are required to implement Backout- on-the-Fly:

- Set up a DDQ environment
- SYCTTBL specifications
- FAR specifications
- Linkedit
- JCL

### 5.2.1 The DDQ Environment

Set up the DDQ environment as described in the Dynamic Data Queuing Facility. A separate DDQ data set for Backout-on-the-Fly is not required, except if using the Multiregion Facility, the file must be unique to the region (cannot be shared across regions). If not specified, the DDQ default data set will be used. If so, insure that PERMS=YES is coded on its DDQDS macro. To define a backout DDQ, code a DDQDS macro with the following parameters:

```
DDNAME=THREDLOG,
BLOCKNG=YES,
PERMS=YES,
SHARED=NO,
RESTART=NO, (if DDQ dedicated to Backout-on-the-Fly or will have
             only transient queues created by other users.)
BLOCKS=n
```

The default number of blocks is 8. However, the acquired core size is, of course, a function of the physical block size. The user is urged to consider the size of the files monitored by Backout-on-the-Fly and written to the DDQ and to consider varying either BLOCKS and/or the physical block size to reduce overhead of either acquiring extents or I/O accesses.

If the default data set is used as the backout DDQ, the DDQDS macro defining it should be coded with the same considerations for the BLOCKS parameter.

NOTE: under Release 10, the THREDLOG (or default) DDQ data set used for Backout-on-the-Fly may be shared across regions.

### 5.2.2 SYCTTBL Requirements

Code BACKOUT=YES for each subsystem subject to Backout-on-the-Fly.

BACKOUT=YES is the default on the SYCTTBL macro. Therefore, if the backout modules are included in the linkedit, all subsystems will be utilizing this feature against those files specified as reversible. For those subsystems for which Backout-on-the-Fly processing is not desired, recode the SYCTTBL with BACKOUT=NO. Also code RVFILE=NO (the default) on all SYCTTBLs, except those for which file or data base recovery is desired instead of Backout-on-the-Fly (code BACKOUT=NO).

### 5.2.3 FAR Specifications

Code REVERSE=YES or CRITICAL for each file subject to Backout-on-the-Fly. Note that the FAR parameters RECREATE and CHECKPOINT apply only to the File Recovery facility and are not used for Backout-on-the-Fly user files.

### 5.2.4 Linkedit Requirements

Include DDQMOD, DDQSTART, RMPURGE, IXFFAR, IXFRVRSE, IXFLOG, INTCRQ, INTPRQ, IXFVERF1, TRVRSE.

The Backout-on-the-Fly modules are:

- INTCRQ

  Record before- and after-images to blocked single retrieval transient DDQ. Called by IXFLOG.

- INTPRQ

  Initiates recovery attempt. Closes and releases backout DDQs. Called by RMPURGE.

- TRVRSE

  Creates unblocked semipermanent queue and then reads it backwards inspecting for before/after-images and calling IXFRVRSE or IXFVERF1 respectively. Called by INTPRQ.

- IXFVERF1

  Reads relevant record from user file and checks to see if it is the same as the recovery after-image. Called by TRVRSE.

## 5.2.5 Execution JCL

JCL requirements for the THREDLOG DDQ data set are as follows:

```
//THREDLOG    DD          DSN=name,DISP=OLD,DCB=(DSORG=DA,OPTCD=RF)
```

A QCF file is also required since semipermanent queues are created for unsuccessful thread completion (see the Dynamic Data Queuing Facility).

## INDEX