SHARE SESSION REPORT

61	A764	The PL/I-VSAM Interface	78
SHARE NO.	SESSION NO.	SESSION TITLE	ATTENDANCE
PL/I		Dan Galender	TYM
PROJECT		SESSION CHAIRMAN	INST. CODE

Tymshare, 20705 Valley Green Dr., Cupertino, CA 95014, 408-446-6775

SESSION CHAIRMAN'S COMPANY, ADDRESS, AND PHONE NUMBER

TITLE: PL/I - VSAM Interface
SPEAKER: Thomas J. Kinzer (IBM)
IBM Corporation
610 Lincoln Street
Waltham, MA 02254
PROJECT: PL/I
SESSION: A764 (Wednesday, August 24, 1983, 4:30 p.m.)

4/C/LEJ/1

The OS/VS PL/I - VSAM Interface

I. Introduction

The IBM PL/I Program Products have provided an interface between PL/I RECORD I/O and VSAM since very early in the life of VSAM. The amount of VSAM function available to PL/I users was increased considerably by the support supplied by Release 3 of the OS PL/I Program Products (and for DOS/VS PL/I users by Release 5 of the DOS PL/I Program Products). This support became available in December 1976.

This paper attempts to describe this support and its implementation for OS PL/I users.

II. Topics

III. PL/I Record I/O Language for VSAM

The details of PL/I record I/O language are covered in the PL/I Language Reference Manual. No attempt will be made here to summarize them exhaustively in written form. Two figures are supplied, however, which do specify the compatible combinations of PL/I statement constructs, PL/I file and environment attributes, and VSAM dataset attributes. (Figures 1 and 2.)

Each statement type is associated with one and only one bit position in the bit strings. If that position is AND'ed together for all the file attributes and dataset attributes applicable to the open PL/I file, a "1" indicates the statement is valid; a "0" indicates it is not. AND'ing together the appropriate strings for a particular set of attributes gives a resultant string defining (by the presence of "1" bits) the statement types that are valid. Note that all record I/O statements are consistent with a VSAM RRDS!

These strings and this table come from the PL/I OPEN module, IBMBOPEA.

Some comments are in order, however, about some of these statements and options in cases where their meaning has apparently not been clear.

A. KEYED SEQUENTIAL Processing of an ESDS.

This support allows sequential processing with or without retrieval of the RBA for any record which VSAM reads from or writes to an ESDS. It also allows retrieval and update by key. PL/I regards RBA's as four-byte character strings which are obtainable from VSAM for use as keys. Using such an RBA as a key should always result in successful retrieval of a record and establishment of positioning for future successful sequential processing (if desired). Use of an invalid RBA as a key results in the KEY condition being raised without positioning being established. Since KEYED SEQUENTIAL processing allows access by RBA to an ESDS, there is no need to support the DIRECT attribute for an ESDS.

B. KEYED SEQUENTIAL Processing of a KSDS.

This support allows retrieval (and update if the UPDATE attribute applies) by key anywhere in the dataset. If a key is specified and a record exists to match this key, the record is retrieved and positioning is established for subsequent sequential reads. If no record is found to match the specified key, the KEY condition is raised, but PL/I establishes positioning in the dataset at the next record following where the missing record would have been, so that

sequential reads may be performed anyway. This provides a technique for "key-greater-than" processing. KEYED SEQUENTIAL processing also allows insertion of new records via WRITE KEYFROM statements either within or beyond the current key range of the dataset. A failing WRITE operation does not cause positioning to be established, and, in fact, causes positioning to be lost insofar as PL/I's definition of positioning is concerned. Under the current implementation of PL/I and VSAM, if an attempt to write to a PL/I KEYED SEQUENTIAL file (with or without ENV(SKIP)) fails due to a duplicate key condition, VSAM will establish positioning at the stored record with the duplicate key. A sequential read will actually retrieve it, but the PL/I language does not define this to be the case, and the PL/I programmer should not, therefore, depend on it. A PL/I program should use a read by key to retrieve this record.

If GENKEY processing is required, the short (generic) key should be supplied via a PL/I character string the length of which (or current length of which, if it has the VARYING attribute) is less than the length of the key. Otherwise, GENKEY has no effect.

C. KEYED SEQUENTIAL Processing of an RRDS.

This support allows a RRDS to be processed either sequentially or by key (relative slot number). Assuming that positioning already exists, a READ without a key causes the next non-empty slot's record to be read. A READ with key specified returns the record stored in the specified slot and establishes positioning for subsequent sequential retrieval. If the slot addressed by the READ with key is empty, PL/I raises KEY but establishes positioning at the empty slot so that a subsequent sequential read will return the record from the next non-empty slot. An attempt to write (not rewrite) into a slot that is already occupied causes positioning to be lost, with KEY (if KEYFROM was specified) or ERROR (if KEYFROM was not specified) being raised. GENKEY cannot be specified for an a RRDS.

D. DIRECT Processing

The DIRECT attribute is not supported when one processes an ESDS (and is not needed since KEYED SEQUENTIAL processing provides the function). DIRECT processing requires the use of exact full-length keys on a KSDS: GENKEY is not supported. For an RRDS, DIRECT processing can be performed using slot numbers.

For AIX processing, the AIX must have the UNIQUE attribute (i.e., no duplicate alternate keys). If the alternate keys are non-unique, KEYED SEQUENTIAL processing must be used.

E. The DELETE Statement

If no key is specified, the record just read will be deleted. If a key is specified and the user has not just read the records with that key, PL/I will read it and then delete it. Such a request by key is not desirable if the file being processed is associated with a PATH used for processing a non-unique alternate index, since the record read and deleted will be the base cluster record associated with the first entry in the alternate index record. Which record this might be is a matter of luck.

F. The REWRITE Statement

REWRITE is handled just like DELETE (above) except that some additional length checking must be performed. Record lengths in an ESDS cannot be changed when the record is rewritten. The warning about alternate index processing (above, under DELETE) is appropriate for REWRITE as well.

IV. PL/I RECORD I/O Implementation Overview.

The OS PL/I Optimizing Compiler and Checkout Compiler do not build detailed control block structures for PL/I files. They build a simple control block called the Declaration Control Block (DCLCB), which contains the information which can be specified on a PL/I file declaration. At execution time, PL/I OPEN builds the PL/I and OS control blocks required to process the dataset, loads the appropriate PL/I I/O routine ("transmitter"), and identifies (and perhaps loads) the appropriate error handling and end-of-file modules.

This process can be seen on the following chart, taken from the OS PL/I Transient Library PLM. (Figure 3.)

PL/I OPEN may be invoked explicitly via an OPEN statement, in which case one or more files can be opened, or implicitly by an I/O request to a file which is not open. In this case, either compiled code or the link-edited record I/O interface module IBMBRIOA branches to an address obtained from PL/I's "Dummy File Control Block" (built during PL/I initialization).This address does not point to OS Data Management or a PL/I transmitter (as it would if the file were open) but to the implicit open entry point of PL/I OPEN/CLOSE. The link-edited OPEN/CLOSE bootstrap module IBMBOCLA loads the first of PL/I's OPEN transients (module IBMBOPAA) from the PL/I Transient Library.

IBMBOPAA is the "driver" of PL/I OPEN. It gets a parameter list from the OPEN statement (or builds one if an implicit open request is being processed), builds a control block called an Open Control Block, and then loads IBMBOPBA to process non-VSAM

STREAM or RECORD files (if there are any) followed by IBMBOPEA to process VSAM files (if there are any). After those modules have completed execution, IBMBOPAA scanns the parameter list and OCB to see if UNDEFINEDFILE should be raised for any files or if PL/I CLOSE should be invoked to close PL/I files whose associated datasets were opened with some sort of error.

The non-VSAM PL/I OPEN module, IBMBOPBA, builds a PL/I File Control Block (FCB) and the appropriate DCB for each non-VSAM dataset in the list and then XCTL's to modules to issue the OS OPEN macro, validate information derived from OS control blocks after the dataset is open, select and load the appropriate non-VSAM transmitters, etc. The details of this are not germane to the present discussion, except for one special bit of processing that IBMBOPBA performs; as it scans the OPEN parameter list, skipping files declared with "ENVIRONMENT (VSAM)", it issues the RDJFCB macro for each remaining dataset's JFCB and looks to see if the dataset is really a VSAM dataset. If it is, IBMBOPBA flags it as VSAM so PL/I VSAM OPEN can process it later. (Actually, if the file declaration specified "ENVIRONMENT (INDEXED)" and the dataset is a VSAM dataset, IBMBOPBA does a little extra checking for certain special cases that require an ISAM DCB and VSAM's FIP. See the PL/I Programmer's Guide.)

In this way, most files originally declared as using ISAM ("ENVIRONMENT (INDEXED)") end up using PL/I native VSAM support if the dataset is actually VSAM. In addition, in many cases a VSAM ESDS, KSDS, or RRDS can be processed via a PL/I file declaration originally coded for QSAM. In particular, if no environment option for dataset organization was coded in the declaration and no information incompatible with VSAM was coded, the PL/I file can process an ESDS, KSDS, or RRDS. If "ENVIRONMENT (CONSECUTIVE)" was explicitly coded, an ESDS can be processed. VSAM datasets cannot be processed via PL/I STREAM files. Under MVS JFCB's look the same whether built by scheduler allocation or dynamic allocation, and PL/I can detect VSAM datasets to be associated with PL/I files not declared with "ENV(VSAM)" under either TSO or batch. In the TSO foreground under SVS (not MVS) this JFCB information is not available to PL/I.

When non-VSAM OPEN is complete, control returns to IBMBOPAA which looks to see if there is any VSAM dataset to open. If there is it calls the PL/I VSAM open module, IBMBOPEA.

IBMBOPEA issues a SHOWCB macro to obtain the size of an ACB and an RPL, gets some PL/I non-LIFO storage, in which it builds PL/I FCB's and VSAM RPL's and ACB's for the VSAM datasets being opened, opens them, issues SHOWCB macros for dataset information, validates dataset information against PL/I file information previously known, gets non-LIFO storage for control

blocks and perhaps a dummy buffer, selects and loads the proper VSAM transmitter, issues any required POINT macros, and returns to IBMBOPAA.

V. VSAM I/O as Used by PL/I

To communicate with VSAM, the assembler language programmer sets fields in a VSAM Request Parameter List (RPL). These fields specify which dataset is to be accessed by addressing its ACB, the address and length of the user's I/O area (if there is one), the length of a short key, and processing options in a field called "OPTCD". After setting these fields in the RPL, the programmer issues GET, PUT, POINT, and perhaps READ, WRITE, CHECK, and ENDREQ macros. The way the macros function is conditioned by the various fields in the RPL.

The PL/I programmer codes PL/I READ, WRITE, REWRITE, UNLOCK, and perhaps WAIT statements. The transmitter must establish the correct values of the RPL fields, issue the appropriate macros, and interpret VSAM error information (if there is any) in PL/I terms. It must see to it that the PL/I record I/O language is implemented, and this means, for example, that PL/I file positioning must be provided even though it sometimes differs from VSAM's own positioning conventions.

VSAM OPTCD settings important for PL/I are reviewed below:

A. ADR/KEY (Addressed vs. Keyed).

"ADR" implies use of a Relative Byte Address, and thus applies only to ESDS processing in PL/I. "KEY" implies use of a logical key. This key is embedded within each record if the dataset is a KSDS. For an RRDS, this key is not within a record, but is a relative slot number.

This distinction represents a VSAM distinction, not a PL/I distinction. If the PL/I programmer is to process via a key, the PL/I file must have the KEYED attribute. This key will be a character string. The transmitter will set ADR/KEY in the RPL based on type of request and type of dataset. In the PL/I program an ESDS RBA, a KSDS logical key, or an RRDS slot number are all keys.

B. FWD/BWD (Forward vs. Backward).

PL/I always uses FWD unless the user coded ENV(BKWD) in the file declaration. In this case PL/I OPEN issues a POINT macro to establish positioning at the end of the dataset.

Backwards retrieval by key (full key, not generic) may be done anywhere in the dataset, but any sequential retrieval (retrieval without a key specified) will retrieve the record

prior to the point of current positioning. If a backwards read by key results in a key-not-found condition, PL/I does not provide positioning for subsequent sequential read requests. VSAM permits the user to switch back and forth between FWB and BWD, but PL/I does not support this capability.

C. ASY/SYN (Asynchronous vs. Synchronous)

Normal synchronous processing returns control to the user only after the request has been "completed", meaning after the RPL and user I/O area are available for reuse. Asynchronous processing returns control to the user before the user can safely reuse the RPL and I/O area; the user must issue a CHECK macro to get error information and re-synchronize with VSAM, thus being assured of being able to safely reuse the I/O area and RPL. Note that the question of when VSAM actually performs the physical I/O operation is an entirely different question.

PL/I normally uses SYN, but if the user requests the EVENT option on the I/O statement, PL/I will issue the I/O requests in ASY mode and issue the CHECK macro at the user's WAIT statement.

D. KEQ/KGE (Key Equal vs. Key Greater Than or Equal)

This distinction is meaningful only for keyed retrieval from a KSDS or RRDS. It specifies for a KSDS whether the user insists on an exact match on key or will take the next higher record in the dataset. This distinction applies also to an RRDS, but only for a POINT macro, not GET.

PL/I always uses KEQ, except that after a keyed retrieval attempt fails due to "key not found", PL/I reestablishes positioning by issuing POINT using the same key but KGE.

E. FKS/GEN (Full Key Search vs. Generic)

This distinction is meaningful only for keyed retrieval from a KSDS. It specifies whether the KEQ or KGE comparison in (D.) above is to be made on the whole key or only on a leading substring of it.

PL/I always uses FKS unless the "GENKEY" environment option applies. If GENKEY was specified a short key will be compared only against a leading substring of a key in the dataset, the length compared being determined by the length of the short key. If a full length key is specified, GENKEY has no effect, that is, the search simply becomes a full key search.

For example, suppose a dataset has seven-byte keys. If GENKEY were not specified and the program supplied a five byte key field, a full key comparison would be performed using the supplied five bytes padded out to seven bytes with blanks. If GENKEY had been specified, only the first five bytes would have been compared. If GENKEY had been specified and a seven-byte key provided, the search would have become a full key search.

F. LOC/MVE (Locate vs. Move)

Conventionally, this distinction differentiates between processing in the buffers and moving the data to or from a workarea. In VSAM, LOC may only be used for input-only requests. This is of such limited usefulness that PL/I only uses LOC for READ IGNORE processing of datasets without spanned records.

PL/I READ INTO and WRITE FROM statements are implemented using MVE, with VSAM moving the data to or from the PL/I record variable. If the PL/I file has the BUFFERED attribute, so that READ SET and LOCATE SET statements may be used, PL/I OPEN allocates a buffer big enough to hold the maximum-sized logical record. READ SET and LOCATE SET statements are then implemented using this buffer and VSAM MVE processing.

G. NSP/NUP/UPD (Note String Position vs. No Update vs. Update)

These distinctions apply to the retrieval, rewriting, and insertion of records. Their meaning with GET and PUT macros is as follows:

GET UPD - Retrieve for update and establish positioning.
PUT UPD - Rewrite and establish positioning.
GET NSP - Retrieve for input only, but establish positioning.
PUT NSP - Write new record and establish positioning.
GET NUP - Retrieve for input and don't establish positioning.
PUT NUP - Write new record and don't establish positioning.

For PL/I files that have the DIRECT attribute, positioning is irrelevant, and PL/I uses either UPD or NUP. For PL/I KEYED SEQUENTIAL files, positioning must be maintained, and PL/I uses either UPD or NSP.

H. DIR/SEQ/SKP (Direct vs. Sequential vs. Skip)

These three critical distinctions represent the three VSAM processing modes: direct, sequential, and skip sequential.

1. DIR implies that the user must supply a key (RBA for an ESDS, logical key for a KSDS, or relative slot number for an RRDS). PL/I uses DIR to process a KSDS or RRDS if the PL/I file declaration has the DIRECT attribute. In this case, PL/I requires a "full key" (FKS) and a "key equal" (KEQ) comparison, although VSAM itself permits KGE and/or GEN for GET requests in DIR mode.

   If a PUT macro is issued against a KSDS using DIR, VSAM finds the appropriate control interval by means of a top-down index search, and places the record in it if there is room for it. If there is not room, VSAM "splits" the control interval at the record boundary nearest the mid-point of the control interval. If a control area must be split, it is split roughly in half. Freespace percentages play no role in determining whether there is room for a new record or a new CI when an insert or update is performed in DIR mode.

   A PUT in DIR mode causes an immediate write to DASD. This immediate write, other DIR processing logic, but record insertion as though SEQ were in effect can be obtained by coding "SIS" - Sequential Insert Strategy.

2. SEQ implies that position is held in the dataset and the user wishes to process forward (if FWD is in effect) or backward (if BWD) from that position.

   For a GET request, there is no way to specify a key. Positioning in the dataset determines which record will be retrieved. Thus retrieval by key using SEQ implies that a POINT/GET macro sequence is required. If a record was retrieved using GET UPD, a subsequent PUT UPD will rewrite (and thus update) it.

   The PUT macro in SEQ mode for a new record must be clearly understood. PUT SEQ conjures up the intuitive image of writing the next tape record, and indeed, this image is appropriate if the VSAM dataset is an ESDS or an RRDS; a new record is always written at the end of an ESDS, and a PUT SEQ to an RRDS causes the record to be written into the slot at which VSAM is positioned. In a KSDS, however, each record, including the new one, has a key embedded within it. The logical sequence of the dataset is defined by the sequence of these keys; VSAM will not write a new record in the wrong place. For a PUT SEQ, VSAM presumes that the new key is higher than

the key of the current positioning and moves forward through the dataset to the proper location in the dataset at which to write the new record. If positioning is not established or if the new key is lower than the key of current positioning, VSAM returns an error indication and does not write the record. If, to implement a PUT SEQ, VSAM must move forward past the end of the current CI, it does so by searching the index top down - a sort of implicit POINT.

PUT SEQ uses a special technique for inserting records into a KSDS so that in case a number of records are to be inserted at one spot in the dataset, VSAM will be able to insert them efficiently. If the new record would have neither the highest nor lowest key in the target CI, VSAM inserts it if it will fit and splits the CI at the point of record insertion if it will not fit. If the new record would have the highest or lowest key in the control interval, VSAM may place it in a new CI by itself so there will be plenty of space after it for the "mass sequential insertion" that may be about to occur. If the new record would have the lowest key in the CI, but there is not room for it, VSAM will always put it in a new CI unless there is not only room for it to fit, but room for it to fit while preserving the free space percentage specified in the catalog. If the record goes into a new CI and the mass sequential insertion does occur, the new CI's will contain free space as specified in the catalog. In effect, PUT SEQ into the middle of a dataset is executed as though the operation were in initial load of the dataset which happens to be occurring in the middle of it, not at the end.

PUT SEQ works similarly insofar as control area splits are concerned. If a CI split would produce a new CI that would not be the last CI in the CA, VSAM will split the CA, but at the point of insertion, not in the middle. If a new highest CI in the CA would cause the CA's freespace to fall below the freespace percentage specified in the catalog, VSAM will not split the CA but will put the new CI at the beginning of a new CA.

Extension into a new CI or CA does not count as a CI or CA split in the VSAM statistics.

It follows that if all insertions into a KSDS are done via PUT SEQ, a lot of freespace will be created, preserved, and propagated, but little of it will be used to hold new records. One might be well-advised to create the dataset with freespace and then alter the freespace percentages downwards after loading the

dataset, but before inserting new records via sequential (PUT SEQ) processing.

PL/I primarily uses SEQ mode to access a VSAM dataset associated with a PL/I SEQUENTIAL file. Records are read sequentially by means of GET SEQ macros. If the file has the KEYED attribute, then records are retrieved by key by using a POINT/GET macro sequence in SEQ mode.

WRITE KEYFROM (or LOCATE KEYFROM) is implemented by PUT SEQ if the key of current positioning is known and the key of the new records is higher than the key of current positioning. If either of these conditions is not met, the record is written using PUT DIR,NSP.

It should be remembered that VSAM reads ahead when one retrieves in SEQ mode and defers writing to disk when one adds or updates records in SEQ mode.

3. SKP implies that skip sequential mode will be used, and is meaningful only for processing a KSDS. GET and PUT with SKP function like GET and PUT with DIR in that the user must supply a key, but differently in that SKP requires that keys be supplied in ascending sequence. SKP processing for ascending keys establishes positioning by reading the sequence set rather than searching the index top down. SKP functions like DIR for inserting new records in that no attempt is made to preserve feespace percentages, and CI splits and CA splits occur at midpoint, not point of insertion. That is, insertions done in SKP mode use "direct insert strategy", not "sequential insert strategy".

   Since VSAM will not insert a record in the wrong place in a KSDS, it may have to position itself forward in the dataset beyond the control interval associated with current positioning. Whereas VSAM uses a top-down index search (implicit POINT) to move forward to implement a PUT SEQ request, it reads forward via the sequence set for a PUT SKP request.

   If the PL/I file declaration specifies "KEYED SEQUENTIAL ENV(SKP)", PL/I sets the default processing mode to SEQ and uses SEQ for reading without a key specified, but attempts to use SKP for keyed requests. For a keyed read PL/I issues GET SKP if the key is greater than the key of current positioning and POINT followed by GET SEQ otherwise.

   For a write by key, PL/I issues a PUT SKP if the key is greater than the key of current positioning and PUT DIR,NSP otherwise.

VI. Implementation Techniques Used in PL/I Processing of VSAM

After the dataset has been successfully opened, there are a PL/I File Control Block (FCB), a VSAM Access Method Control Block (ACB), a dummy record buffer (if the PL/I file has the BUFFERED attribute), a PL/I I/O Control Block (IOCB), and a pair of areas in which to save keys (if the PL/I file has the KEYED attribute). A VSAM Request Parameter List has been built within the PL/I IOCB.

The PL/I control blocks listed here are not unique to PL/I's VSAM support; they might be built (with some differences in content) for datasets requiring other access methods. The VSAM ACB and RPL are not built for any other access method that PL/I supports.

The FCB contains various fields relating to PL/I I/O, and for VSAM it includes the default settings of such fields in the RPL as OPTCD, RECLEN, AREA, AREALEN, and KEYLEN. These default values were placed in the RPL when IBMBOPEA issued the GENCB macro to generate the RPL.

The IOCB contains in addition to the RPL, fields to hold backup copies of the fields in the RPL. The OPTCD, AREA, AREALEN, and KEYLEN do not change unless PL/I changes them, so the PL/I transmitter can generate in workareas in the IOCB the settings required for the current request, compare them to the current settings in the RPL (without actually referencing the RPL) and avoid issuing a MODCB macro unless it is really necessary. In this way, PL/I can maintain the discipline of the MODCB, SHOWCB, TESTCB, GENCB interface without actually issuing any more of these macros than absolutely necessary. Record length represents a special case, since after a READ operation,the length of the record just read is stored in the RECLEN field of the RPL. The PL/I transmitter does not retrieve it unless it has to, (i.e., unless the file has the SCALARVARYING attribute, the record variable has the VARYING attribute, or a record from an ESDS is to be rewritten). Thus, if the most recent operation was READ, PL/I may have to execute an otherwise unneeded MODCB before executing a WRITE or REWRITE because it doesn't know the value currently set for RECLEN in the RPL.

At the beginning of each I/O request, the default setting of OPTCD is copied from the FCB into a workarea in the IOCB. The options are then modified as necessary for this request, and required values of other RPL fields are determined. Finally, a common routine is invoked to compare the requisite fields to the backup copies of the current RPL fields and a MODCB parameter list is progressively built in the IOCB as mismatches are found. At the end of this process, if a MODCB parameter list has been started, it is completed and a single MODCB macro is issued. If MODCB is not needed, it is not issued.

VII. PL/I Record I/O Modules for VSAM

PL/I OPEN loads the appropriate transmitter to provide the interface to data management. There are currently some fifty-four of these modules in the PL/I Transient Library. They all have names of the form:

IBMBRxyA

There are four VSAM transmitters, recognizable by the "V" as the sixth letter of their names. They are:

IBMBRVAA - ESDS Transmitter
IBMBRVGA - KSDS Sequential Output Transmitter
IBMBRVHA - KSDS and PATH Transmitter
IBMBRVIA - RRDS Transmitter

PL/I OPEN also at least identifies an error module and an EOF module, but does not actually load them into storage unless the program is running under PL/I multitasking. For VSAM there is a single error module, and it handles EOF as well. Record I/O error modules all have names of the form:

IBMBREyA

PL/I's error module for VSAM is IBMBREEA.

The transmitters receive each request, validate it, handle WAIT statements and event variables if necessary, build the parameters needed by VSAM as described earlier, issue the MODCB macro if necessary, issue the appropriate VSAM macros (GET, PUT, POINT, plus CHECK for WAIT statements), and set flags for the error module if any exceptional conditions arise. The transmitter either returns control to the user, or loads (if necessary) and calls the error module.

The error module sets up the paremeters (ONKEY, ONFILE, ONCODE, etc.) needed by the error handler and identifies the condition to be raised based on information passed to it by the transmitter. This does not include VSAM feedback (FDBK) fields, since the transmitter has already translated VSAM information into PL/I terms. When an attempt to read a KSDS or RRDS declared with the KEYED SEQUENTIAL attributes causes key-not-found, it is the error module which, just before raising the PL/I KEY condition, issues a POINT macro with OPTCD KGE to establish positioning for the user.

VIII. ESDS Processing - IBMBRVAA

This transmitter executes using the SEQ and ADR options. Any WRITE request places the new record at the end of the dataset.

A READ KEY statement becomes a VSAM POINT/GET macro sequence. The KEYTO option allows RBA's to be retrieved.

IX. KSDS Sequential Output Processing - IBMBRVGA

This transmitter performs only the functions of initial load and resume load of an KSDS. It only implements WRITE KEYFROM, WRITE KEYFROM EVENT, and LOCATE KEYFROM statements. Each new record must have a higher key than any record already in the dataset.

X. KSDS and Path Processing - IBMBRVHA

This transmitter performs all functions associated with KSDS processing except dataset loading. It processes alternate index paths. It implements, in one context or another, every record I/O statement defined by the PL/I language except WRITE or LOCATE without KEYFROM.

If the PL/I file has the KEYED and SEQUENTIAL attributes, then the default processing mode established by PL/I OPEN is SEQ. If ENV(SKIP) was not specified, SEQ is reinstated for every I/O operation except an out-of-sequence WRITE, in which case PUT DIR,NSP is issued. If ENV(SKIP) was specified, SEQ is tentatively reinstated for every I/O request (and used in a READ request without a key specified), but PUT DIR,NSP is used for an out-of-sequence WRITE, POINT followed by GET SEQ is used for an out-of-sequence keyed READ, and SKP mode is used for all other keyed requests.

PL/I DIRECT files are processed in VSAM DIR mode.

XI. RRDS Processing - IBMBRVIA

This transmitter performs all functions associated with RRDS processing. In one context or another it implements every possible syntactic variant of every record I/O statement defined by the PL/I language!

The key used with an RRDS is a relative slot number, which from VSAM's point of view is a binary integer. To PL/I, dataset keys are character strings, and the relative slot number is a character string eight bytes long. When the user supplies the key, the key field must be a decimal number that either fills the whole eight-byte string or is bounded before, after, or both by blanks. When PL/I returns such a key to the user (e.g., KEYTO) the numeric digits will be right-justified and the leading zeroes will be suppressed. One should code rather carefully to minimize character-to-arithmetic conversions, since they are performed by library call. It is best to use "PIC '(7)Z9'" as the data type of the key.

XII. Alternate Indices

VSAM permits the user to define alternate indices over a KSDS or an ESDS using any contiguous field within the first 4096 bytes of a record (and within the first segment of a spanned record) as the alternate key. The maximum length of such a key is 256 bytes. As the base cluster records are changed, VSAM "upgrades" the alternate indices as necessary to reflect these changes.

To process a dataset by alternate key, one declares a PL/I file just as one would for a KSDS. The execution time DD statement points to a dataset name which is the name of a PATH linking the alternate index and base cluster. If the keys in the AIX are UNIQUE, the PL/I file can have either the DIRECT or KEYED SEQUENTIAL attributes. If they are non-unique, the PL/I file must have the KEYED SEQUENTIAL attributes. For non-unique alternate keys, the SAMEKEY BIF tells the user whether or not the set of primary keys for an alternate key has been exhausted as the user processes them sequentially.

It is possible to declare two PL/I files, point one to an AIX path and the other to the base cluster, and process either or both at the same time. There are no problems at all with doing this for input-only processing. There are no problems doing this with updates to either or both as long as both are PL/I DIRECT files (in which case the AIX must have the UNIQUE attribute).

Problems can arise when updating with both a path and base cluster open if the AIX is non-unique (and thus must be processed via a PL/I KEYED SEQUENTIAL file) and/or the base cluster is processed as a PL/I KEYED SEQUENTIAL file.

Users should be very careful with application programs that do such processing. There is no way for the PL/I programmer to deal with the VSAM multiple-string processing issues that arise in such a program.

XIII. Buffering

OS/VS PL/I-VSAM users can control buffering by specifying BUFNI and BUFND either in the JCL or in the PL/I file declaration. For KSDS processing that does anything except read the KSDS sequentially as though it were a reel of tape, it is essential to good performance to provide enough index buffers (via BUFNI) to enable VSAM to do top-down index searches efficiently. Thus a PL/I file declaration that specifies KEYED SEQUENTIAL (with or without ENV(SKIP)) or DIRECT should specify a number for BUFNI larger than VSAM's default of one. The number specified should exceed the number of index levels shown in the VSAM catalog for the KSDS to be processed.

Extra data buffers are helpful for doing CA splits and for reading ahead sequentially. They are not helpful if one's sequential processing is really direct processing by key with occasional "runs" of sequential processing interspersed with lots of repositioning in the dataset. This latter situation is likely not uncommon for PL/I KEYED SEQUENTIAL processing.

A special case of interest involves an application program which reads an input transaction file and either updates an existing master file record (if such a record exists) or writes a new record into the master file. One can imagine two scenarios for such a program:

-Try to read the master file record. If it exists, update it. If KEY is raised (due to key-not-found), build a new record and write it.

-Build and try to write a new master file record. If KEY is raised (due to duplicate key), read the record and update it.

Most programmers instinctively implement the first approach. If any significant percentage (over five or ten percent) of the transactions will actually result in adding a new record, the second approach is more efficient because of VSAM buffering and the fact that a failed read-for-update attempt causes loss of positioning.

XIV. Bibliography

A. SHARE/GUIDE Papers

1. "VSAM Concepts and User Experiences", Tina Martin, SHARE 45, August 1975.

2. "VSAM Application Design and Implementation", T. J. Kinzer, SHARE 45, August 1975.

3. "New Releases of PL/I Products - A Step Forward", T. J. Kinzer, GUIDE 43, November 1976.

4. "Recent Improvements in PL/I", T. J. Kinzer, SHAPE 48, March 1977.

5. "VSAM Performance Improvements for Random or Alternate Index Processing", G. H. Royer, GUIDE 45, November 1977.

6. "PL/I and VSAM: Forwards, Backwards, and Inside-Out, Bob Stearns, SHARE 51, August 1978.

B. IBM PL/I Manuals

1. OS PL/I Checkout and Optimizing Compilers: Language Reference Manual, GC33-0009.

2. OS PL/I Checkout Compiler: Programmer's Guide, SC33-0007.

3. OS PL/I Optimizing Compiler: Programmer's Guide, SC33-0006.

4. OS PL/I Checkout Compiler: Messages, SC33-0034.

5. OS PL/I Optimizing Compiler: Messages, SC33-0027.

6. OS PL/I Checkout Compiler: Execution Logic, SC33-0032.

7. OS PL/I Optimizing Compiler: Execution Logic, SC33-0025.

8. OS PL/I Transient Library: Program Logic, LY33-6009.

C. IBM VSAM Manuals

1. Planning for Enhanced VSAM Under OS/VS, GC26-3842.

2. OS/VS Virtual Storage Access Method (VSAM) Options For Advanced Applications, GC26-3819.

3. OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide, (For VS1 R6 and MVS R3, 7), GC26-3838.

4. OS/VS2 Access Method Services, (For MVS), GC26-3841.

5. Data Facility Extended Function: Access Method Services Reference, SC26-3967.

6. OS/VS1 Access Method Services, (For VS1), GC26-3840.

7. OS/VS2 Virtual Storage Access Method (VSAM) Logic, (For MVS), SY26-3825.

8. OS/VS1 Virtual Storage Access Method (VSAM) Logic, (For VS1), sy26-3841.

9. OS/VS2 SVS Independent Component: Access Method Services, (For SVS), GC26-3867.

10. OS/VS2 SVS Independent Component: Planning For Enhanced VSAM, (For SVS), GC26-3869.

11. OS/VS2 SVS Independent Component: VSAM Options For Advanced Applications, (For SVS), GC26-3870.

12. OS/VS2 SVS Independent Component: Virtual Storage Access Method (VSAM) Programmer's Guide, (For SVS), GC26-3868.

13. OS/VS2 SVS Independent Component: Virtual Storage Access Method (VSAM) Logic, (For SVS), SY26-3857.

D. Other

1. OS/VS VSAM Sharing - A Technical Discussion (Part I), Edward H. Daray, IBM Corporation, Palo Alto Systems Center, G320-6015.

2. IMS/VS and OS/VS VSAM Buffer Options (Shared vs. on-Shared Resources), Wayne Weikel & Edward H. Daray, IBM Corporation, Palo Alto Systems Center, G320-6035.

3. VSAM Primer and Reference, G320-5774.

Masks which indicate valid statements. This table shows which bits represent which statements.

```
1  BYTE 0 80 READ SET
2         40      SET KEYTO
3         20      SET KEY
4         10      INTO
5         08      INTO KEYTO
6         04      INTO KEY
7         02      INTO KEY NOLOCK
8         01      IGNORE
9  BYTE 1 80      INTO EVENT
10        40      INTO KEYTO EVENT
11        20      INTO KEY EVENT
12        10      INTO KEY NOLOCK EVENT
13        08      IGNORE EVENT
14        04 WRITE FROM
15        02       FROM KEYFROM
16        01       FROM EVENT
17 BYTE 2 80       FROM KEYFROM EVENT
18        40 REWRITE
19        20         FROM
20        10         FROM KEY
21        08         FROM EVENT
22        04         FROM KEY EVENT
23        02 LOCATE
24        01        KEYFROM
25 BYTE 3 80 DELETE
26        40        KEY
27        20        EVENT
28        10        KEY EVENT
29        08 UNLOCK
30        04 WRITE FROM KEYTO
31        02       FROM KEYTO EVENT
```

Figure 1

Bit Strings For Valid Combinations

```
DC    X'26329458'        DIRECT
DC    X'FDEFFFF6'        SEQUENTIAL
DC    X'FFFFFDFE'        KEYED
DC    X'918D6AA0'        NON-KEYED
DC    X'FFF80008'        INPUT
DC    X'FFFFFCFE'        UPDATE
DC    X'00078306'        OUTPUT
DC    X'FFFD7E0E'        ESDS
DC    X'FFFAFDF8'        KSDS
DC    X'1FFFBCFE'        UNBUFFERED
DC    X'FF0673CC'        BUFFERED
DC    X'FDEFFFF6'        NON-EXCLUSIVE
DC    X'FFF87CF8'        BKWD
DC    X'FFFFFFFF'        RRDS
```

Figure 2

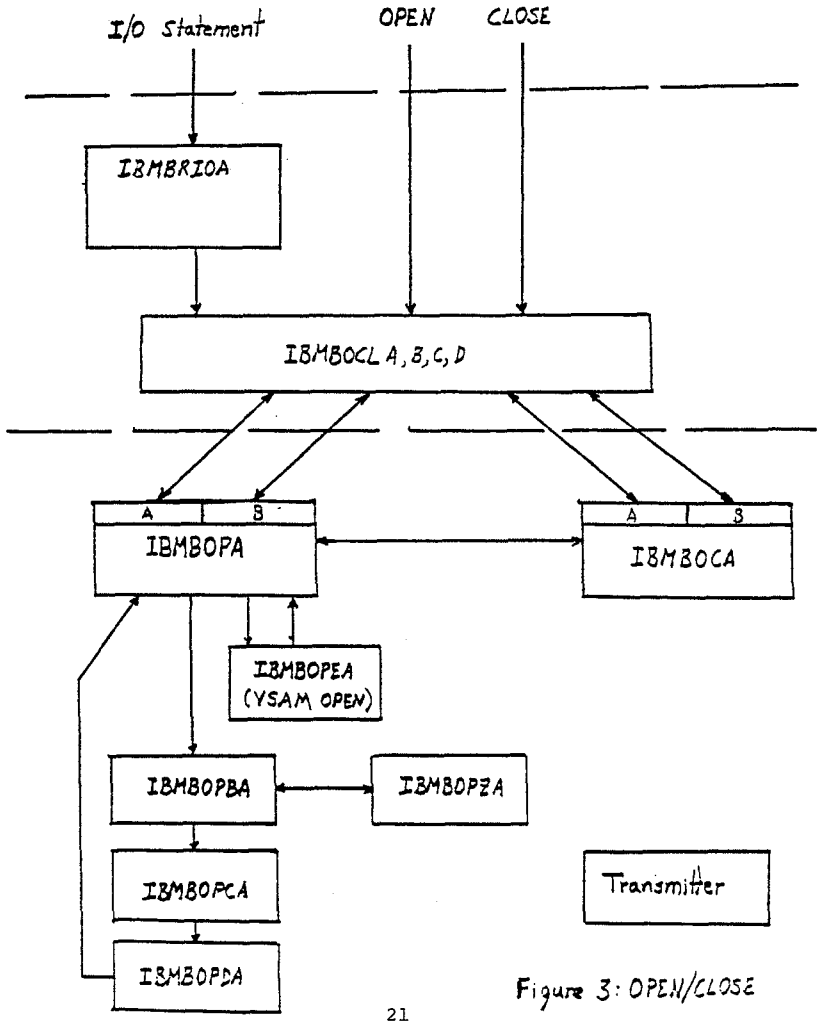

Figure 3: OPEN/CLOSE