General Programming Considerations

This section explains several concepts that are important to consider when programming calls.

Internal IDs can be specified to perform important functions during Adabas command execution. In the control block of an Adabas direct call command, you can specify a:

- Command ID (ACBCID or ACBXCID). This is a non-blank, non-zero value specified in the ACB or ACBX that acts as an internal ID for the command processing. It can be used to eliminate repeated interpretation and conversion by successive commands that use the same format buffer.
- Format ID. This is a separate four-byte internal ID for decoded format buffers defined either as user-specific or globally available (a *global format ID*) to other users running on the same Adabas nucleus.

The uses of these IDs is explored in detail in this section.

Also described in this section are the procedures used to retrieve ISNs from the Adabas Work and the multifetch and prefetch options. Multifetch and prefetch options are used to reduce execution time for programs that process large amounts of data in sequential order by reducing the number of system commands needed to the complete the Adabas call.

This chapter covers the following topics:

- Command, Format, and Global Format IDs
- ISN List Processing
- Using the Multifetch/Prefetch Feature

Command, Format, and Global Format IDs

The *command ID*, specified in the Adabas control block (ACBCID or ACBXCID), performs important functions during Adabas command execution. It is an automatically generated or user-specified nonblank, nonzero value that performs the following functions:

- It prevents repetitive format buffer decoding by acting as an internal ID for decoded record formats.
- It tags ISN lists generated by the S*x* command for later access, and saves ISN list overflow.
- It tags (identifies) sequential read processes through sets of records.

If desired, a separate internal *format ID* for decoded format buffers can be specified. This value is identified by a flag in the high-order (leftmost) two bits of the first byte of the Additions 5 field. Depending on the flag value, the format ID can be user-specific (an individual format ID) or available to other users running on the same Adabas nucleus (a global format ID).

This section covers the following topics:

Specifying Command, Format, and Global Format IDs

The following table summarizes the Adabas control block (ACB or ACBX) settings required to specify command IDs, format IDs, and global format IDs.

ID	ACB or ACBX Field Specifications	
	Command ID (ACBCID or ACBXCID)	**Additions 5 (ACBADD5 or ACBXADD5)**
Command ID	Set this field to any non-blank, non-zero value. Specifying a command ID value of X’FFFFFFFF’ causes Adabas to generate command IDs automatically, beginning with X’00000001’ and incrementing by 1 for each new command ID.	High-order, leftmost bits set to binary "00".
Format ID	Set this field to any non-blank, non-zero value. Specifying a command ID value of X’FFFFFFFF’ causes Adabas to generate command IDs automatically, beginning with X’00000001’ and incrementing by 1 for each new command ID.	Set the high-order, leftmost two bits of the first byte to binary "10"; set the fifth through eighth bytes to the format ID. The ID may not start with X’FE’ or X’FF’. In non-mainframe Adabas environments, set the first byte of the Additions 5 field to be any lowercase letter.
Global Format ID	Set this field to any non-blank, non-zero value. Specifying a command ID value of X’FFFFFFFF’ causes Adabas to generate command IDs automatically, beginning with X’00000001’ and incrementing by 1 for each new command ID.	Set the high-order, leftmost two bits of the first byte binary "11"; set the remaining bytes to the ID. All eight bytes of the Additions 5 field are used as a global format ID. In mainframe environments, the first byte may be assigned in the hexadecimal range E2 through E9 (characters S-Z); all other ranges are reserved for use by Software AG. In non-mainframe Adabas environments, the first byte may be assigned in the hexadecimal range 51-5A (characters S-Z).

Notes:

1. When B’00’ is set in the high-order (leftmost) two bits of the first byte of the Additions 5 field, the command ID is automatically used as the format ID.
2. When B’10’ is set in the high-order (leftmost) two bits of the first byte of the Additions 5 field, separate values are used for the command ID and the format ID, and the fifth through eighth bytes of the Additions 5 field are used as the format ID.
3. When B’11’ is set in the high-order (leftmost) two bits of the first byte of the Additions 5 field,

separate values are used for the command ID and the format ID, the format ID is treated as a global format ID and all eight bytes of the field are used as the global format ID.

4. Adabas does not verify the assignment of global format IDs. It is your responsibility to ensure that only global format IDs in the allowable range are assigned. Software AG can neither enlarge the range of global format IDs available to users nor make any changes to its products to resolve a global format ID assignment problem.

Command IDs for Read Sequential Commands

The read sequential commands (L2/L3, L5/L6, L9) require that a command ID be specified. The command ID is needed by Adabas to return the records to the user in the proper sequence. These command IDs are maintained by Adabas in the table of sequential commands.

The command ID value provided with these commands is also entered and maintained in the internal format buffer pool unless a separate format ID is provided, as described in the section *Using Separate Command and Format IDs*. The command ID is released by Adabas when an end-of-file condition is detected during read sequential processing.

Command and Format IDs for Read, Update, and Find Commands

The read commands (L1-L6, L9) and update commands (A1/A4, N1/N2) require a format buffer that identifies the fields to be read or updated. This format buffer must be interpreted and converted into an internal format buffer by Adabas. Using a valid command ID avoids repeated interpretation and conversion by successive commands that use the same format buffer.

A read or update command with a valid command ID causes Adabas to check whether the command ID is in the internal format buffer pool. If the command ID is present, its internal format buffer is used, and no format buffer reinterpretation is required.

Note:
When reading or updating a series of records that use the same format buffer, processing time can be significantly reduced if you use a command ID.

Internal format buffers (and the format IDs) resulting from L9 commands can only be used by other L9 commands. Moreover, L9 commands cannot use non-L9 internal format buffers / format IDs. This is also true of global format buffers (and global format IDs) and L9 commands.

When reading and updating the same fields (for example, L5 followed by A1), Software AG also recommends that the same command ID be used for both commands (see the A1/A4 and N1/N2 commands for restrictions on using the same format buffer for reading and updating).

If the read-first-record option is used with an S1/S2/S4 command and a command ID is specified, the command ID and the resulting internal format buffer are also stored in the internal format buffer pool.

If the internal format buffer pool is full and a command is received with a command ID without an internal format in the pool, Adabas overwrites the longest unused entry in the pool with the new interpreted format ID. If a command is subsequently received that uses the deleted command ID, the reinterpreted format buffer for that command ID replaces the next-longest unused entry in the pool. For this reason, programs must not change the format buffer between successive read or update commands with the same command ID. Note, however, that use of a command ID does not guarantee that the format buffer is not reinterpreted.

Using Separate Command ID and Format IDs

It is possible to use separate values for command IDs and format IDs. As long as the high-order (leftmost) two bits of the Additions 5 field are set to binary ’00’, the command ID is automatically used as the format ID. If, however, the Additions 5 field’s high-order two bits are binary ’10’, the fifth through eighth bytes (Additions 5 + 4(4)) of the field are used as the format ID. Note that the ID may not start with X’FE’ or X’FF’.

Note:
To identify the format ID as separate from the command ID, non-mainframe Adabas environments expect the first byte of the Additions 5 field to be any lowercase letter. When using separate format IDs in a heterogeneous environment, it is important to identify them alike across all platforms used in the system.

Using a Global Format ID

Particularly in an online environment, multiple users of the same program often read or update the same fields of a file and therefore use identical format buffers.

- When you use the *individual* format ID option, Adabas must store the same internal format buffer for each user.
- When you use the *global* format ID option, a single internal format buffer is shared by many users and the need for Adabas to overwrite internal format buffer pool entries is reduced. This option identifies the format buffer to each user by format ID only, rather than by both format ID and terminal ID. A command ID cannot be designated as a global format ID; in addition, the restriction of L9 formats and their IDs being valid only for use by other L9 commands also applies to global formats and IDs.

The global format ID option is activated by setting the high-order (leftmost) two bits of the first byte of the Additions 5 field to binary ’11’ (see *Specifying Command, Format, and Global Format IDs*). This causes all eight bytes of Additions 5 to be recognized as the global format ID.

Note:
To identify the format ID as global in non-mainframe Adabas environments, the first byte of the Additions 5 field must be set to be any digit or uppercase letter. When using global format IDs in a heterogeneous environment, it is important to identify them alike across all platforms used in the system.

The first byte of the global format ID may be assigned in the hexadecimal range E2-E9; characters S-Z. All other ranges are reserved for use by Software AG.

The allowable range of values for global format IDs in non-mainframe Adabas environments is hexadecimal 51-5A; characters S-Z.

Warning:
Adabas does not verify the assignment of global format IDs. It is the user’s responsibility to ensure that only global format IDs in the allowable range are assigned. Software AG can neither enlarge the range of global format IDs available to users nor make any changes to its products to resolve a global format ID assignment problem.

A global format ID can be deleted using the RC command and specifying the global format ID in the Additions 5 field, as described.

Command IDs Used with ISN Lists

If a command ID is specified for any command which results in an ISN list (S1,S2,S4,S5,S8,S9), the command ID value may be used to identify the list at a later time.

- If the save-ISN-list option is used for an S*x* command, a command ID must be provided. The save-ISN-list option causes the entire ISN list to be stored on the Adabas Work. ISNs from the list may subsequently be retrieved by an S*x* command or by using the GET NEXT option of the L1/L4 command.
- If the save-ISN-list option is not used and an ISN buffer overflow condition occurs (the entire ISN list cannot be inserted in the ISN buffer), the overflow ISNs will be stored on the Adabas Work only if a command ID value was used. In this case, the command ID and the ISN list it identifies will be released by Adabas when all the ISNs have been returned to the user.

Automatic Command ID Generation

Automatic command ID generation may be invoked by specifying a command ID value of X’FFFFFFFF’. This causes the Adabas nucleus to generate command IDs automatically, beginning with X’00000001’ and incrementing by 1 for each new command ID. Automatic command ID generation may not be desirable in all cases; refer to the section *Command and Format IDs for Read, Update, and Find Commands*.

Releasing Command IDs

You can release a command ID and its associated entries (or ISN list) with an RC command, a CL command, or by using the release-CID option of any S*x* command (S1, S2, S4, S5, S8, S9).

The RC command contains options that allow you to release only those command IDs contained in the internal format buffer pool, the table of sequential commands, or the table of ISN lists.

The CL command causes all the command IDs currently active for you to be released.

The release-CID option of an S*x* command causes the CID specified to be released as the first action taken by the command.

Internal Identification of Command IDs

Each command ID entry is identified by Adabas using an internal user ID together with the command ID value. As a result, one user need not be concerned with the command ID values in use by another. However, a user should avoid using the same command ID value for different commands, particularly if the command ID is used for sequential read (L2/L5, L3/L6, L9) commands and S*x* commands.

Examples of Command ID Use

This section covers the following topics:

- Example 1 : Find / Read Processing
- Example 2 : Find / Read Using the GET NEXT Option
- Example 3 : Read / Update Processing
- Example 4 : Read / Find Processing

Example 1 : Find / Read Processing

A set of records is to be selected and read. The same format buffer is to be used for each record being read.

```
FIND (S1)           CID=EX1A
READ (L1)           CID=EX1B
READ (L1)           CID=EX1B
```

Example 2 : Find / Read Using the GET NEXT Option

A set of records is to be selected and read using the GET NEXT option of the L1/L4 command.

```
FIND (S1)           CID=EX2A
READ (L1)           CID=EX2A
READ (L1)           CID=EX2A
READ (L1)           CID=EX2A
```

Example 3 : Read / Update Processing

A file is to be read and updated in sequential order. The same format buffer is to be used for reading and updating.

```
READ PHYS SEQ (L5)  CID=EX3A
UPDATE (A4)         CID=EX3A
READ PHYS SEQ (L5)  CID=EX3A
UPDATE (A4)         CID=EX3A
```

Example 4 : Read / Find Processing

A file is to be read in logical sequence. A find command is to be issued to a second file using the value of a field read from the first file, and the records that result from the find command are then to be read using the GET NEXT option.

```
READ LOG SEQ (L3)   CID=EX4A
FIND (S1)           CID=EX4B
READ (L1)           CID=EX4B
READ (L1)           CID=EX4B
READ LOG SEQ (L3)   CID=EX4A
FIND (S1)           CID=EX4B
READ (L1)           CID=EX4B
```

ISN List Processing

This section discusses the procedures used to retrieve ISNs from the Adabas Work data set. If the GET NEXT option of the L1/L4 command is used to read the records that correspond to the ISNs contained in the ISN list, ISN handling as discussed in this section is performed automatically by Adabas, and the user need not make use of these procedures.

The notation S*x* command as used in this section refers to any command that may result in an ISN list (S1/S2/S4/S5/S8/S9).

This section covers the following topics:

- Storage of ISN Lists
- Retrieval of ISN Lists
- ISN List Processing Examples

Storage of ISN Lists

Adabas stores ISNs on the Work data set under either of the following conditions:

- An S*x* command is issued, a nonblank nonzero command ID is specified, and the save-ISN-list option is specified. The entire resulting ISN list is stored.
- An S*x* command is issued, a non-blank non-zero command ID is specified, the save-ISN-list option is not specified, and the resulting ISN list contains more ISNs than can be inserted in the ISN buffer. Only the overflow ISNs are stored in this case.

If an S*x* command is issued with blanks or binary zeros in the command ID field, Adabas does not store any ISNs on the Adabas Work.

Retrieval of ISN Lists

You can retrieve ISNs stored on the Adabas Work data set by issuing an S*x* command in which the same command ID value is used as was used for the initial S*x* command. When an S*x* command with an active command ID value is issued, Adabas uses this as an indicator that you are requesting ISNs from an existing ISN list. Adabas locates the ISN list identified by the specified command ID and inserts the next group of ISNs in the ISN buffer. As many ISNs are returned as can fit in the ISN buffer.

This section covers the following topics:

- Save-ISN-List Option Specified
- Save-ISN-List Option Not Specified
- Using the ISN Quantity Field of the Control Block

Save-ISN-List Option Specified

If the save-ISN-list option was specified with the S*x* command used to create the ISN list, Adabas uses the ISN specified in the ISN lower limit field to determine the next group of ISNs to be returned.

The next group begins with the first ISN that is greater than the ISN specified in ISN lower limit.

- If binary zeros are specified, the next group begins with the first ISN in the list.
- If a value is specified which is greater than any ISN in the list, response code 25 is returned.

If the ISN list was created using an S2 command, the ISN specified must be present in the ISN list. Use of the save-ISN-list option thus permits the user to skip forward and backward within an ISN list. This is useful for programs that must perform forward and backward screen paging.

Save-ISN-List Option Not Specified

If the save-ISN-list option was not specified with the S*x* command used to create the ISN list, Adabas returns the ISNs in the order in which they are positioned in the list, and deletes each group from the Work when it has been inserted in the user's ISN buffer. The command ID used to identify the list is released when the last group of ISNs has been returned to the user. The ISN lower limit field is not used in this case, unless processing is to begin above a specified ISN range.

Using the ISN Quantity Field of the Control Block

The user can determine when all of the ISNs in a list have been retrieved using the ISN quantity field of the control block.

- The first S*x* command returns the total number of records that satisfy the search criteria in this field.
- In addition, each subsequent S*x* command used to retrieve ISNs from the Adabas Work uses this field to store the number of ISNs that were inserted in the ISN buffer.

ISN List Processing Examples

This section covers the following topics:

- Example 1 : Using Sx Command with L1/L4 Commands with GET NEXT Option
- Example 2 : Using the Save-ISN-List Option
- Example 3 : With ISN Overflow Handling
- Example 4 : Without ISN Overflow Handling

Example 1 : Using S*x* Command with L1/L4 Commands with GET NEXT Option

```
Sx command
L1/L4 command with 'GET NEXT' option
L1/L4 command with 'GET NEXT' option
L1/L4 command with 'GET NEXT' option
```

The following examples show various results according to the size of the ISN buffer. Positioning for ISN list is defined by ISN lower limit.

1.

```
ISN Buffer Length = 0
read the ISN list to the work area
```

2.

```
L1/L4 with GET NEXT
result: first ISN’s record
```

1.

```
ISN Buffer Length = 4
read first ISN with S1
```

2.

```
L1/L4 with GET NEXT
result: second ISN’s record
```

1.

```
ISN Buffer Length = 12
read first ISN with S1
first 3 ISNs are returned in ISN buffer
```

2.

```
L1/L4 with GET NEXT
result: read fourth/fifth/sixth ISNs’ records
```

Example 2 : Using the Save-ISN-List Option

Initial S*x* call using save-ISN-list option:

```
Command = Sx
Command ID = SX01               (save-ISN-list option)
Command Option 1 = H
ISN Lower Limit = 0
ISN Buffer Length = 20
CALL ADABAS ...
```

Resulting ISN quantity = 7 (total matching ISNs in stored list)

Resulting ISN list: (all ISNs are stored on Work):

```
8    12  14  15  24  31  33
```

Resulting ISN buffer:

```
8    12  14  15  24
```

Subsequent S*x* call:

```
Command = Sx
Command ID = SX01
ISN Lower Limit = 24     (limit ISN choice to 24, +)
ISN Buffer Length = 20   (space for 5 ISNs from ISN list)
CALL ADABAS ...
```

Resulting ISN quantity = 2 (total ISNs returned in ISN buffer)

Resulting ISN buffer:

```
31   33   14  15  24.... remainder of ISN buffer unchanged....
```

Subsequent S*x* call:

```
Command = Sx
Command ID = SX01
ISN Lower Limit = 0
ISN Buffer Length = 20
CALL ADABAS ...
```

Resulting ISN quantity = 7

Resulting ISN buffer:

```
8    12  14  15  24
```

Example 3 : With ISN Overflow Handling

Initial S*x* call (save-ISN-list option not used):

```
Command = Sx
Command ID = SX02
Command Option 1 = blank     (no option)
ISN Lower Limit = 0
ISN Buffer Length = 20
CALL ADABAS ...
```

Resulting ISN quantity = 7 (total ISNs returned in ISN buffer and stored on Work)

Resulting ISN list: (only ISNs 31 and 33 are stored on Work)

```
8    12  14  15  24  31  33
```

Resulting ISN buffer:

```
8    12  14  15  24
```

Subsequent S*x* call:

```
Command = Sx
Command ID = SX02
ISN Lower Limit     (not used)
ISN Buffer Length = 20
CALL ADABAS ...
```

Resulting ISN quantity = 2

Resulting ISN buffer:

```
31   33   14   15  24
```

ISNs 31 and 33 are deleted from the Adabas Work, and command ID SX02 is released. A subsequent S*x* call with command ID ’SX02’ will be processed as an initial S*x* call since ’SX02’ was released after the last ISNs were returned to the user.

Although the ISN lower limit is not specified in this example, a non-zero value would also return only those ISNs greater than the specified value in the ISN buffer, just as in example 2.

Example 4 : Without ISN Overflow Handling

Initial S*x* call with blank or zero command ID:

```
Command = Sx
Command ID = blanks or binary zeros
Command Option 1 = blank     (no option)
ISN Lower Limit = 0          (no lower limit specified)
ISN Buffer Length = 20
CALL ADABAS ...
```

Resulting ISN quantity = 7 (total matching ISNs)

Resulting ISN list: none are stored on Work

```
8  12  14  15  24  31  33
```

Resulting ISN buffer:

```
8  12  14  15  24
```

A subsequent S*x* call with command ID equal to blanks or binary zeros and ISN lower limit equal to 0 will result in a reexecution of the same find command with the same result as the initial call. A subsequent call with command ID equal to blanks or binary zeros and ISN lower limit = 24 causes reexecution of the S*x* command. The result will be ISN quantity of 2 with ISNs 31 and 33 in the ISN buffer.

Using the Multifetch/Prefetch Feature

Programs that process large amounts of data in sequential order require frequent storage access, causing long execution times. The Adabas multifetch and prefetch options significantly reduce the execution times of such programs by reducing the number of system commands needed to complete Adabas calls.

The multifetch and prefetch options reduce execution time in almost all normal applications; however, the specific advantage depends on the type of application program.

Note:
In Adabas 8, ACBX interface calls support the multifetch but not the prefetch feature. However, the prefetch feature is still supported for ACB interface direct calls; so, if your application uses ACB interface direct calls, you can continue to use the prefetch feature in those calls only.

This section covers the following topics:

- Multifetching Versus Prefetching
- Invoking Multifetch/Prefetch
- Multifetch Operation Processing
- Prefetch Operation Processing

- The Effects of ISN Changes on Prefetched or Multifetched Records

Multifetching Versus Prefetching

The multifetch and prefetch features reduce the communication overhead between the application program and the Adabas nucleus. Multifetch operations store multiple records that result from a single call and then transfer the records to the user in the record buffer. Without multifetch operations, multiple Adabas calls would be necessary to obtain the same result. When an ACB interface direct call is performed, the record descriptor elements (RDEs) of multifetched records are stored in the ISN buffer; when an ACBX interface direct call is performed, the record descriptor elements (RDEs) of multifetched records are stored in multifetch buffers.

Multifetch operation is similar to prefetch; multifetch comprises prefetch functions and more. Releases of Adabas prior to Adabas 8 support *both* prefetch and multifetch operations; however, new programs should use the multifetch (M) option, which is common across all Adabas platforms. In addition, the prefetch option is no longer supported for releases of Adabas 8 or later.

Invoking Multifetch/Prefetch

If you are using *only* ACB interface direct calls in your application, you can invoke prefetch or multifetch processing using one of two methods. How they are invoked determines where the preread records are held for processing and therefore what buffer space must be allocated.

- The first method, specifying the PREFETCH=YES (for multifetch processing) or OLD (for prefetch processing) parameter on the ADARUN statement, is the most efficient and requires no application programming changes. These parameters provide control for batch jobs.

  When PREFETCH=YES or OLD, Adabas uses a double-buffering technique that allows processing of one group of records while the following group is being fetched.

  For more information about the prefetch/multifetch ADARUN parameters, which include PREFETCH, PREFICMD, PREFIFIL, PREFNREC, PREFSBL, PREFTBL, PREFXCMD, and PREFXFIL, read *Adabas Initialization (ADARUN Statement)*.

  Note:
  PREFETCH=OLD is available for the purpose of upwards compatibility. Some newer Adabas features are no longer supported. For example, attempts to pass APLX multiple buffers with this access will return a response code 22, subcode 51. We recommend you use PREFETCH=YES if such new features must be supported.

- The second method is to specify the M or O option (for multifetch processing) in the L1/L4, L2/L5, L3/L6, L9, BT or ET commands or the P option (for prefetch processing) in the L1/L4, L2/L5, L3/L6 or L9 commands. Use of the command-level options M, O, and P are provided in the documentation for individual commands, as well as in *Multifetch Operation Processing* and *Prefetch Operation Processing*.

If you use ACBX interface direct calls in your application or if you *mix* ACB and ACBX interface direct calls in your application, you can invoke multifetch processing only by specifying the M or O option in the L1/L4, L2/L5, L3/L6, L9, BT or ET commands. Use of the command-level options M and O are provided in the documentation for individual commands, as well as in *Multifetch Operation Processing*.

Note:
Prefetch processing is not supported for applications that use ACBX interface direct calls.

Multifetch Operation Processing

Multifetch operations are compatible with the corresponding operations on non-mainframe platforms, and can be used across platforms in heterogeneous environments.

Multifetching can be used with the following Adabas commands:

- L1/L4 with I or N option (read by ISN, find with GET NEXT)
- L2/L5 (read physical)
- L3/L6 (read logical by descriptor)
- L9 (histogram)
- BT (backout transaction)
- ET (end of transaction)

For all read calls (L*x*), multifetch returns a group of records in the record buffer and a description of these records in either the caller's ISN buffer (for ACB interface direct calls) or the caller's multifetch buffer (for ACBX interface direct calls). The maximum number of records is limited by the following values, which are specified in one of Adabas control blocks (ACB or ACBX) or the ABD, as appropriate:

- User-defined maximum as input to the call
- Record buffer length
- ISN buffer length (ACB interface direct calls)
- Multifetch buffer length (ACBX interface direct calls)

This section covers the following topics:

- READ (Lx) Multifetch Processing
- BT / ET Multifetch Processing

READ (L*x*) Multifetch Processing

If you are making a direct call for a read command and you want to use multifetch processing, certain fields must be set prior to the call, as follows:

What to Set	Where to Set It: ACB Interface Direct Call	ACBX Interface Direct Call	
	ACB	**ACBX**	**ABD**
Supported command type and options (see the command list in section *Multifetch Operation Processing*)	Command code (ACBCMD)	Command code (ACBXCMD)	---
Maximum number of values to return, or *0* to multifetch all values.	ISN lower limit (ACBISL)	ISN lower limit (ACBXISL)	---
Set to "M" or *O* (see notes below)	Command Option 1 (ACBCOP1)	Command Option 1 (ACBXCOP1)	---
Length of the record buffer	Record buffer length (ACBRBL)	---	Buffer size (ABDXSIZE) in the record buffer ABD
Length of the ISN buffer	ISN buffer length (ACBIBL)	---	---
Length of the multifetch buffer	---	---	Buffer size (ABDXSIZE) in the multifetch buffer ABD

Notes:

1. *Command option "M" indicates that the multifetch option is to be used. Command option "O" selects both the multifetch option ("M") and the existing command option "R" (returns Adabas response code 145 for a requested ISN that is already being held by another user) for the L4/L5/L6 commands.*
2. *For an L1 command, either the command option "I" (ISN sequence) or option "N" (GET NEXT option) must be specified with the multifetch command option "M" or "O"; otherwise, Adabas response code 22 occurs.*

The contents of the returned record buffer and ISN or multifetch buffer are as follows:

```
Record Buffer: record1,record2, ... ,recordn
```

Records are returned in the record buffer as usual. If more than one record is returned, all records are placed adjoining in the record buffer.

Descriptive elements for these records are returned in the ISN buffer. The first (leftmost) fullword of the ISN buffer contains the number of elements that follow (signed integer, four bytes). Following this count are the record descriptor elements, each 16 bytes long:

```
ISN or Multifetch Buffer: RDE count{RDE1 }...
```

A record descriptor element (RDE) has the structure shown in the following table.

Format	Length	Content
All fields unsigned integer, right aligned	4 bytes	Length of this record in record buffer. Records may have different lengths.
	4 bytes	Adabas response for this record. If a nonzero response is given, no record is stored in the record buffer.
	4 bytes	ISN for this record.
	4 bytes	(L9 only) ISN quantity: value count for this descriptor.

If an error is detected while the first record is being processed, the error response is returned in the response code field of the appropriate Adabas control block (ACBRSP or ACBXRSP).

If an error is detected while a record other than the first is being processed, the response code is returned in the corresponding record descriptor element in the ISN or multifetch buffer.

BT / ET Multifetch Processing

By default, Adabas releases all currently held ISNs for the user issuing a BT/ET command. With the multifetch option, only a subset of the records held by the current transaction is released. The records to be released from hold status are specified in the ISN buffer if you are using the ACB direct call interface; if you are using the ACBX direct call interface, the records to be released from hold status are specified in the multifetch buffer. The first fullword in the ISN or multifetch buffer specifies the number of 8-byte elements following.

You can activate the command-level multifetch feature for the ET/BT command call by setting the following fields of the Adabas control block as indicated:

If you are making a direct call for an ET or BT command and you want to use multifetch processing, certain fields must be set prior to the call, as follows:

What to Set	Where to Set It		
	ACB Interface Direct Call	ACBX Interface Direct Call	
	ACB	ACBX	ABD
"BT" or "ET"	Command code (ACBCMD)	Command code (ACBXCMD)	---
"M"	Command Option 1 (ACBCOP1)	Command Option 1 (ACBXCOP1)	---
Length of the ISN buffer	ISN buffer length (ACBIBL)	---	---
Length of the multifetch buffer	---	---	Buffer size (ABDXSIZE) in the multifetch buffer ABD

Note:
If multifetch is set with ADARUN PREFETCH=YES, the "P" option (prefetch) is automatically used for ET and BT commands; the "M" option (multifetch) is automatically used for all other commands.

The ISN or multifetch buffer must contain the following values:

```
ISN  or Multifetch Buffer: ISN descriptor count {ISN descriptor element (See table below)} ...
```

An ISN descriptor element has the structure shown in the following table.

Format	Length	Content
Binary, right aligned	4 bytes	Adabas file number
	4 bytes	ISN

Prefetch Operation Processing

Prefetch is effective for programs that call sequential commands (L1/L4 with GET NEXT, L2/L5, L3/L6, L9) using the ACB direct call interface. It is not available for use with ACBX direct calls.

When using prefetch, a series of sequential read commands requires only one Adabas call. This single call causes several records to be read at a time from the database. This results in a significant reduction in interregion communication overhead and also permits the overlapped operation of the user program and the Adabas nucleus.

Note:
If the hold option is used (L4/5/6 commands), Adabas places records in hold status when they are read into the prefetch buffer area. This means that if an ET command is issued before all records have been processed, all records (including those not yet processed) are released. The hold ISN option of the ET or HI command can be used to place any such records back into hold status.

Specific commands or files can be excluded from prefetch option processing by specifying the files or commands to be excluded with the respective ADARUN PREFXFIL or PREFXCMD parameters.

This section covers the following topics:

- Invoking Prefetch Operation with Command Option P
- Additional Prefetch Programming Considerations

Invoking Prefetch Operation with Command Option "P"

When enabling prefetch with the command-specific P option, Adabas uses the ISN buffer defined within the user program as the intermediate storage area for the pre-read records. Each record in the ISN buffer is preceded by a 16-byte header:

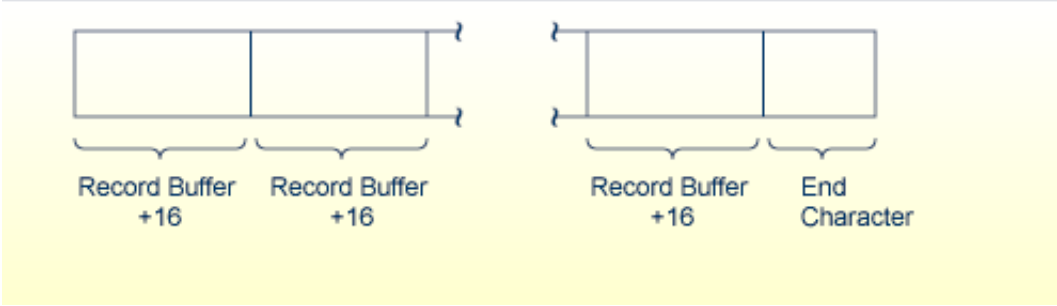
Byte	Use
1-2	Length of record (including length definition). A length of zero indicates the end of data.
3-4	Nucleus response code
5-8	Nucleus internal ID (if the response code is neither zero nor 3, a subcode is returned in the rightmost 2 bytes)
9-12	ISN of the record
13-16	ISN quantity (L9 command only)

The first record is provided by Adabas in the record buffer (without the 16-byte header). The user must then process additional records from the ISN buffer. When end-of-file occurs, the header of the last record in the ISN buffer contains Adabas response code 3, and the two-byte end character contains binary zeros.

Additional Prefetch Programming Considerations

The following are points to consider when using the prefetch option:

- The record buffer size should be set just large enough to contain the largest expected decompressed record.
- If the sequential pass of a file is not to be continued until end-of-file condition is detected, be sure to issue an RC command to release the command ID used whenever file processing has been completed.
- The command ID should not be changed during file processing.
- When using a command option "P" to invoke prefetch operation, the ISN buffer size must be a multiple of the total of the record buffer length plus 16, and a final two bytes for an end character:



ISN Buffer Size for Prefetch Programming

The Effects of ISN Changes on Prefetched or Multifetched Records

If a prefetched or multifetched record is updated, the following processing is applied:

Important:
You should not modify a file (especially the descriptor being accessed) while reading or locating (finding) records in the same file.

- Any update-protection only applies to the active sessio;, anything done by other sessions is not included.
- If an ISN is modified for a record that has been preferchted or multifetched, it is re-fetched (via the L1 command) when the program finally gets to it.
- If an ISN is deleted for a record that has been preferchted or multifetched, it is skipped.
- If an ISN is inserted for a record that has been preferchted or multifetched, it is skipped. For this reason, we do not recommend that you modify files while reading or locating records in the same file.