Command Changes

This chapter describes the changes that have been made to commands, the ACB, and to the Adabas direct call interface for Adabas 8. It covers the following topics:

- Direct Call Enhancements
- Extended Adabas Control Block (ACBX) Support
- Adabas Buffer Description (ABD) Support
- Format Buffer Changes
- LF Command Changes

Direct Call Enhancements

A new direct call interface has been introduced with Adabas 8 that is based on new Adabas 8 extended Adabas control block (ACBX) and Adabas buffer description (ABD) structures.

Notes:

1. *None* of the enhancements are incompatible with the existing Adabas ACB-based direct call interface.
2. Existing application programs that use the ACB-based direct call interface can continue to run in the same way, without change.
3. In addition, you can decide whether you want to use the ACBX-based or ACB-based direct call interface in your application programs, on a call-by-call basis. The same program can use both interfaces.

Some of the new features of Adabas 8 require that your application program use the ACBX-based direct call interface. For example, if your application program makes use of the long buffers (larger than 32K) or the segmented buffers (multiple format and record buffers), available with Adabas 8, you must use the ACBX-based direct call interface.

For information about the new ACBX direct call, including its syntax, read *Calling Adabas*. For information about the ACBX structure and ABDs, read *Adabas Control Block Structures (ACB and ACBX)* and *Adabas Buffer Descriptions (ABDs)*.

Extended Adabas Control Block (ACBX) Support

A new extended Adabas control block, the ACBX, is now provided to support the increase in the buffer sizes in Adabas commands. The existing, non-extended Adabas control block (ACB) is still supported and your existing applications will still work, but if you want to take advantage of some of the extended features provided in Adabas 8, you must use the new ACBX. Specifically, you must use the ACBX if you are using the long buffer (buffers longer than 32K) or segmented buffer (multiple format/record buffer pairs or format/record/multifetch buffer triplets) features of Adabas 8.

Otherwise, your application programs may freely switch between Adabas calls using the existing direct call interface (ACB) and calls using the new interface (ACBX).

This section covers the following topics:

- How Adabas 8 Distinguishes Between ACB or ACBX Use
- Differences between the ACB and the ACBX

For detailed information about the control blocks supported by Adabas, including structure and DSECTs, read *Adabas Control Block Structures (ACB and ACBX)*.

How Adabas 8 Distinguishes Between ACB or ACBX Use

Any application program can make both ACB and ACBX direct calls. The control block (ACB or ACBX) is the first parameter in Adabas calls using either the ACB or ACBX interfaces. Adabas 8 determines which control block is used for a call by the presence of any value beginning with the letter "F" at offset 2 of the control block. Offset 2 in the ACB is the command code field (ACBCMD), but since there is no valid F* Adabas command, no valid direct call using the ACB will contain a value beginning with the letter "F" at offset 2. Offset 2 in the ACBX is a new version field (ACBXVER) identifying the new ACBX.

The presence or absence of an "F" at offset 2 determines how Adabas 8 interprets the direct call. If an "F" is specified in offset 2, Adabas interprets the control block and remaining direct call parameters as an ACBX call; if an "F" is *not* specified in offset 2, Adabas interprets the control block and remaining direct call parameters as an ACB call. If, for some reason, the remaining control block fields and direct call parameters are *not* specified correctly for the type of call indicated by the presence or absence of an "F" at offset 2 (for example, if ACB parameters are specified for an ACBX call), errors may result or the results of the call may not be as expected. For more information about how direct calls are specified using the new ACBX, read *Direct Call Changes*.

Differences between the ACB and the ACBX

The ACBX differs in many ways from the ACB. The ACBX includes some new fields that are not included in the ACB and the sizes of some ACBX fields are larger than their ACB equivalents. These expansions in the ACBX have been made to ensure that its structure can be flexible enough to handle potential future enhancements to Adabas, without altering its fundamental structure for many years.

This section describes the differences between the ACB and the ACBX:

- Control Block Length
- Buffer Length Fields
- Command Options, Additions, and Reserved Fields
- Unit Differences
- Field Length Differences
- New Fields in ACBX
- ACB Dual Purpose Field Changes
- Structure and Offset Differences

Control Block Length

The new ACBX is 192 (or X’C0’) bytes in length; the ACB is 80 bytes long.

Buffer Length Fields

The buffer length fields are not included in the ACBX as they are in the ACB. In Adabas 8, they are instead provided in the individual Adabas buffer descriptions (ABDs). So the ACBX contains no buffer fields corresponding to the ACBFBL, ACBIBL, ACBRBL, ACBSBL, and ACBVBL found in the ACB; the ABDs associated with the call are used instead. One ABD represents an individual Adabas buffer segment. They are described in *Adabas Buffer Descriptions*.

Command Options, Additions, and Reserved Fields

The number of command option, additions, and reserved control block fields are increased in the ACBX:

- The ACBX contains eight command option fields, up from the two command option fields available in the ACB.
- The ACBX contains six additions fields, up from the five additions fields available in the ACB.
- The ACBX contains four reserved fields, up from one reserved field available in the ACB.

  Reserved ACBX fields must be set to binary zeros; the reserved 4 field (ACBXRSV4) should be initialized to binary zeros and then left unchanged.

Unit Differences

The units used to measure command time (thread time) differ between the ACB and the ACBX. The ACB measures command time (ACBCMDT) in 16 microsecond units; the ACBX measures command time (ACBXCMDT) in 1/4096 microsecond units.

Field Length Differences

The lengths of many control block fields are increased in the ACBX. The following table summarizes these changes:

Field Title	Length	
	ACB	ACBX
File Number	2	4
Database ID	2	4
ISN	4	4
ISN Lower Limit	4	4
ISN Quantity	4	4
Compressed Record Length	4	8
Decompressed Record Length	4	8
Command Time	4	8
User Area	4	16
Format Buffer Length	2	4 (in the ABD)
Record Buffer Length	2	4 (in the ABD)
Search Buffer Length	2	4 (in the ABD)
Value Buffer Length	2	4 (in the ABD)

New Fields in ACBX

The following new fields have been introduced in the ACBX:

ACBX DSECT Name	Description
ACBXADD6	Additions 6
ACBXCOP3	Command options 3
ACBXCOP4	Command options 4
ACBXCOP5	Command options 5
ACBXCOP6	Command options 6
ACBXCOP7	Command options 7
ACBXCOP8	Command options 8
ACBXDBID	The database ID. In the ACB, the database ID is stored in the response code field (ACBRSP) for X’30’ calls and in the first byte of ACBFNR for other logical calls.
ACBXERRA	Error offset into the buffer (32-bit).
ACBXERRB	Error character field (field name).
ACBXERRC	Error subcode.
ACBXERRD	Error buffer ID, if multiple buffers are involved.

ACBX DSECT Name	Description
ACBXERRE	Error buffer sequence number, if multiple buffers are involved.
ACBXERRG	Error offset into the buffer (64-bit) - this field is not yet supported.
ACBXLCMP	Compressed record length (or portion of record if the entire record is not read). In the ACB, the compressed record length is stored in the Additions 2 field (ACBADD2).
ACBXLDEC	Decompressed record length. In the ACB, the decompressed record length is stored in the Additions 2 field (ACBADD2).
ACBXLEN	The length of the ACBX, currently 192
ACBXRSV2	Reserved. The value of this field must be set to zero.
ACBXRSV3	Reserved. The value of this field must be set to zero.
ACBXRSV4	Reserved for use by Adabas.
ACBXSUBR	Subcomponent response code, used by Adabas add-on products.
ACBXSUBS	Subcomponent response subcode, used by Adabas add-on products.
ACBXSUBT	Subcomponent error text, used by Adabas add-on products.
ACBXVER	When set to C’F2’, this field indicates to Adabas that the new extended ACB (ACBX) is used.

ACB Dual Purpose Field Changes

There are a number of cases where an ACB field that has multiple purposes has been split out into new fields in the ACBX:

- In the ACB, the Response code field (ACBRSP) is used to store the database ID for X’30’ calls. For the other logical calls the one-byte database ID was stored in the first byte of the file number field, ACBFNR. The ACBX provides a Database ID field (ACBXDBID) for this purpose.
- In the ACB, the ACBADD2 field is used to retain error information for certain Adabas response codes. In the ACBX, new error information fields (ACBXERR* series) are provided for this purpose.
- In the ACB, the ACBADD2 field is used to return, for a successful call, the compressed and decompressed record lengths of the processed data. In the ACBX, for a successful call, the Compressed Record field (ACBXLCMP) contains the length of the compressed data processed by Adabas and the Decompressed Record field (ACBXLDEC) contains the length of the decompressed data.

Structure and Offset Differences

The offset and sequence of ACBX fields is generally different from the corresponding ACB fields, as depicted in the following table. For detailed information about the ACBX structure, read *Adabas Control Block Structures (ACB and ACBX)*.

Offset	ACB DSECT Field Name	ACBX DSECT Field Name
00	ACBTYPE (Call type)	ACBXTYPE (Call type)
01	reserved	ACBXRSV1 (reserved 1)
02	ACBCMD (Command code)	ACBXVER (ACBX version indicator)
04	ACBCID (Command ID)	ACBXLEN (ACBX length)
06	*(ACBCID continued)*	ACBXCMD (Command code)
08	ACBFNR (File number)	ACBXRSV2 (reserved 2)
0A	ACBRSP (Response code -- used for the database ID with X'30' calls)	ACBXRSP (Response code)
0C	ACBISN (ISN)	ACBXCID (Command ID)
10	ACBISL (ISN lower limit)	ACBXDBID (Database ID)
14	ACBISQ (ISN quantity)	ACBXFNR (File number)
18	ACBFBL (Format buffer length)	ACBXISNG (8-Byte ISN)
1A	ACBRBL (Record buffer length)	*(ACBXISNG continued)*
1C	ACBSBL (Search buffer length)	ACBXISN (ISN -- *included in ACBXISNG*)
1E	ACBVBL (Value buffer length)	*(ACBXISN and ACBXISNG continued)*
20	ACBIBL (ISN buffer length)	ACBXISLG (8-Byte ISN Lower Limit)
22	ACBCOP1 (Command option 1)	*(ACBXISLG continued)*
23	ACBCOP2 (Command option 2)	*(ACBXISLG continued)*
24	ACBADD1 (Additions 1)	ACBXISL (ISN lower limit -- *included in ACBXISLG*)
28	*(ACBADD1 continued)*	ACBXISQG (8-Byte ISN Quantity)
2C	ACBADD2 (Additions 2)	ACBXISQ (ISN quantity -- *included in ACBXISQG*)
30	ACBADD3 (Additions 3)	ACBXCOP1 (Command option 1)
31	*(ACBADD3 continued)*	ACBXCOP2 (Command option 2)

Offset	ACB DSECT Field Name	ACBX DSECT Field Name
32	*(ACBADD3 continued)*	ACBXCOP3 (Command option 3)
33	*(ACBADD3 continued)*	ACBXCOP4 (Command option 4)
34	*(ACBADD3 continued)*	ACBXCOP5 (Command option 5)
35	*(ACBADD3 continued)*	ACBXCOP6 (Command option 6)
36	*(ACBADD3 continued)*	ACBXCOP7 (Command option 7)
37	*(ACBADD3 continued)*	ACBXCOP8 (Command option 8)
38	ACBADD4 (Additions 4)	ACBXADD1 (Additions 1)
40	ACBADD5 (Additions 5)	ACBXADD2 (Additions 2)
44	*(ACBADD5 continued)*	ACBXADD3 (Additions 3)
48	ACBCMDT (Command time)	*(ACBXADD3 continued)*
4C	ACBUSER (User area)	ACBXADD4 (Additions 4)
54	---	ACBXADD5 (Additions 5)
5C	---	ACBXADD6 (Additions 6)
64	---	ACBXRSV3 (reserved 3)
68	---	ACBXERRG (Error offset in buffer, 64-bit -- this is not yet supported).
6C	---	ACBXERRA (Error offset in buffer, 32-bit)
70	---	ACBXERRB (Error character field)
72	---	ACBXERRC (Error subcode)
74	---	ACBXERRD (Error buffer ID)
75	---	ACBXERRE (Error buffer sequence number)
78	---	ACBXSUBR (Subcomponent response code)
7A	---	ACBXSUBS (Subcomponent response subcode)
7C	---	ACBXSUBT (Subcomponent error text)

Offset	ACB DSECT Field Name	ACBX DSECT Field Name
80	---	ACBXLCMP (Compressed record length)
88	---	ACBXLDEC (Decompressed record length)
90	---	ACBXCMDT (Command time)
98	---	ACBXUSER (User area)
A8	---	ACBXRSV4 (reserved 4)

Adabas Buffer Description (ABD) Support

Since Adabas 8, through its ACBX interface, supports segmented buffers (multiple pairs of format and record buffers, or multiple triplets of format, record, and multifetch buffers), the total number of buffers is no longer fixed and limited. The individual buffers are no longer described by fields in the ACBX itself (in the way the buffer lengths are defined in the ACB); instead, each buffer has its own Adabas buffer description (ABD) structure that describes what kind of buffer it is, where it is located, what size it is, and other pertinent information.

The addresses of ABDs can be specified in direct calls to Adabas using the ACBX. This section describes the structure of an ABD.

With Adabas 8, you can define ABDs for eight different types of buffers:

- Format buffers
- Record buffers
- Multifetch buffers
- Search buffers
- Value buffers
- ISN buffers
- Performance buffers (used by Adabas Review)
- User buffers

Each Adabas buffer segment is represented by a single ABD, although you can define multiple ABDs of a given type in the same program. Offset 4 (ABDID) in each ABD identifies the type of buffer defined by the ABD.

For detailed information about ABDs, including their structure, read *Adabas Buffer Descriptions (ABDs)*.

Format Buffer Changes

The following changes have been made to format buffers and format buffer specifications in Adabas 8:

- A new format buffer indicator, *L*, can now be used to retrieve or specify the actual length of an LB or long alpha (LA) field value. This format buffer element is referred to as the *length indicator*.
- More than 32 K of data per buffer can now be specified in Adabas commands.
- The field length specifications in a format buffer for alphanumeric and wide-character fields has been extended from 253 bytes to 2,147,483,647 bytes.
- Adabas format, record and multi-fetch buffers specified in Adabas direct calls using the ACBX interface can be split into multiple segments, which need not be contiguous in storage.
- Support for large object (LB) fields has been added to format buffers. This includes the ability to specify lengths using zero field lengths and asterisk (*) field length notation. For complete information on format buffer support for LB fields, read *Format Buffer Support of LB Fields*.
- For files using the Adabas 8 extended MU/PE limits, format buffer elements for occurrence counts (for example, FB=’MUC.’) must be adjusted so they can handle two-byte occurrence count values (for example, FB=’MUC,2,B.’).

This section covers the following topics:

- Length Indicator (L)
- Rules for Specifying Multiple Adabas Buffer Segments

Length Indicator (L)

A new format buffer indicator, *L*, can now be used to retrieve or specify the actual length of any LA or LB field value. This format buffer element is referred to as the *length indicator*.

Note:
At this time, the length indicator can only be used in format buffer specifications for LA or LB fields. Support for use of the length indicator in other fields as well will be considered in a future release of Adabas.

The length indicator is specified using the field name followed by the character *L* (for example, `FB=’ACL,4,B.’` would return the length of the AC field). If the field is a multiple-value field or is located in a periodic group, the field name and character L are followed by the related occurrence indices (for example, `FB=’ACL2,4,B.’` would return the length of the second value of multiple-value field AC). The compressed field length is returned in four-byte binary format. A different length and format cannot be specified.

This section covers the following topics:

- Using the Length Indicator with MU/PE Fields
- Using the Length Indicator in Read Commands
- Using the Length Indicator in Update Commands

Using the Length Indicator with MU/PE Fields

When used with MU or PE fields, the length indicator must specify occurrence indices, including the range of occurrence index. However, it cannot specify the 1-N occurrence index. Consider the following examples.

1. In the following example, the length of the fifth value of the second occurrence of periodic group field AC would be returned:

   ```
   FB='ACL2(5).'
   ```

2. In the following example, the lengths of the first ten values of multiple value field AC would be returned:

   ```
   FB='ACL1-10.'
   ```

3. The following example is illegal as the 1-N notation is notation is not allowed with the length indicator in MU/PE fields:

   ```
   FB='ACL1-N.'
   ```

4. The following example is also illegal as the length indicator does not support MU fields with out an occurrence index:

   ```
   FB='MCL.'
   ```

In addition, if you elect to combine the length indicator and an asterisk length notation value request in the same format buffer for an MU or PE field, the value requests must use corresponding ranges as the length requests. It does not matter whether the length requests and value requests are specified in the same or different format buffer segments. Consider the following examples, where XX is an LA or LB field with the MU option:

1. The following valid examples request the length of the first two values of the XX field as well as their actual values.

   ```
   FB='XXL1-2,XX1-2,*.'
   ```

   ```
   FB='XXL1,XXL2,XX1,*,XX2,*.'
   ```

2. The following *invalid* examples are attempts to request the length of the first two values of the XX field as well as their actual values. However, these examples are invalid because the ranges specified for the MU field in the length and value requests are not specified in a corresponding manner.

   ```
   FB='XXL1,XXL2,XX1-2,*.'
   ```

   ```
   FB='XXL1-2,XX1,*,XX2,*.'
   ```

3. The following two format buffers request the length of the third and fourth values of the XX field, as well as their actual values.

   ```
   FB='XXL3,XXL4.'
   ```

   ```
   FB='XX3,*,XX4,*.'
   ```

4. The following invalid format buffers attempt to request the length of the third and fourth values of the XX field, as well as their actual values, but fail because the ranges specified for the length and value requests are not specified in a corresponding manner.

   ```
   FB='XXL3,XXL4.'
   ```

   ```
   FB='XX3-4,*.'
   ```

Using the Length Indicator in Read Commands

When the length indicator is specified for a field in the format buffer of a read command, the number of bytes required for the field value in the record buffer (without padding and with no further length indication) is returned at the corresponding field position in the record buffer. The amount of space required in the record buffer is based on the field format and the UES-related definitions for the database, file, and user.

You cannot, in the same format buffer, specify the length indicator to retrieve the length of an LA or LB field in combination with a request for the actual field value in a different format (converted) from the field's base format. For example, if character LB field L1 is stored in format A, `FB='L1L,4,B,L1,*,W.'` is an illegal format buffer specification because it requests the length of the the L1 field in addition to the value of the L1 field converted to Unicode (format W). The reason for this restriction is that the length element gives the length of the field in its native format, but the length of the value returned in the requested format (Unicode) would be different due to the conversion.

If character LB field L1 (format A) contains a 40,000-byte EBCDIC value, consider the following examples:

1. Suppose the format buffer specification for L1 is:

   ```
   FB='L1L,4,B.'
   ```

   The record buffer will contain the four-byte binary length of the value of field L1:

   ```
   X'00009C40'
   ```

2. Suppose the format buffer specification for L1 is:

   ```
   FB='L1L,4,B,L1,*,A.'
   ```

   The record buffer will contain the four-byte binary length of the value of field L1 at the beginning of the record buffer area , followed by 40,000 characters of the actual L1 data.

Read Operations, Length Indicators, and the NB (No Blank Compression) Option

If the length indicator of a field is specified in the format buffer (for example, `FB='L1L,4,B.'`), and if the field is *not* subject to blank compression (the NB option is specified for the field in the FDT), the length returned is the number of bytes specified when the value was stored (which can be zero). However, if the field is subject to blank compression, the length returned is the number of significant left-most bytes, beyond which the value is padded with blanks. For an all blank value, the returned length is the length of one blank (one byte for alphanumeric fields, two bytes for wide-character fields).

Using the Length Indicator in Update Commands

When a length indicator is specified in the format buffer for an update command, the corresponding value in the record buffer specifies the actual value length of the field in the record buffer. Only one length indicator for the base field can be specified and it must be accompanied by the asterisk (*) length notation in the format buffer.

The length indicator must occur in its format buffer segment prior to any format element that implies a variable length in the record buffer (due to the use of asterisk notation or zero length notation). In other words, the length indicator is located in a constant position, independent of the values of any fields mentioned in the format.

Rules for Specifying Multiple Adabas Buffer Segments

Adabas format, record and multifetch buffers specified in Adabas direct calls using the ACBX interface can be split into multiple segments, which need not be contiguous in storage. The search, value, ISN, monitor, and user information buffers must each be provided in a single segment, as in prior releases.

The following rules apply to multiple segment format buffers in ACBX Adabas direct calls:

- Multiple format and record buffers usually come in pairs. For commands where data in the record buffer is *not* described by a format specification in the format buffer, no format buffer segments need be specified; if any are specified, they are ignored.
- Multiple records can be read in one Adabas call using the multifetch feature. If multifetch is used, the format, record, and multifetch buffer segments must be specified in triplets.
- Each format buffer segment must end with a period, as in prior releases, and must be a complete and valid format buffer on its own.

  If a format ID is specified for a call, Adabas associates it with the set of all specified format buffer segments. If a subsequent call specifies an almost -- but not exactly -- identical format buffer (including the sequence of buffer segments), it must use a different format ID or release and redefine a previously used format ID.
- The specification sequence of buffers or buffer segments in an Adabas direct call depends on the type of command:

  1. An ACBX is followed by zero or more format buffer segments, usually the same number of record buffer segments, and, for calls with multifetch, the same number of multifetch buffer segments.

     For commands where data in the record buffer is *not* described by a format specification in the format buffer, no format buffer segments need be specified; if any are specified, they are ignored.
  2. Depending on the type of Adabas command, one search, value, or ISN buffer segment may follow.
  3. At the end of the call, zero or one performance buffer segment may be appended and zero or one user information buffer segment may be appended.

LF Command Changes

If an LF command with Command Option 2 set to "S" is run and large object (LB) fields are encountered, the LB field description is returned in an F-type field element. Bit 6 in the second format byte (at offset 7 or byte 8 in the element) is set to indicate that the LB (large object) option is set for the field. In addition, bit 1 of the second format byte indicates whether the LB field is defined with the NB option. For more information, read about the LF command in the *Adabas Command Reference Guide*.