Large Object (LB) Files and Fields

Adabas 8 introduces large object fields (LB fields). These are fields that may contain much more data than the 253 bytes of normal alphanumeric fields or the 16,381 bytes of LA fields. Fields containing large objects are defined with the new LB option. The theoretical maximum size of the value of an LB field is just short of 2 GB; practical usable sizes are smaller.

Adabas stores LB field values in a separate file, called a *LOB file*, that is tightly associated with the file containing the LB fields, called the *base file*.

Getting Started with Large Object (LB) Fields

To use LB fields, complete the following steps:

1. Define an FDT with one or more fields assigned the LB option. You can use the ADACMP COMPRESS function to create LB fields. For example:

```
ADACMP COMPRESS
ADACMP FNDEF='1,AA,8,A,DE'                          Key field
ADACMP FNDEF=...
ADACMP FNDEF='1,AZ,250,A,NU'               Data field
ADACMP FNDEF='1,L1,0,A,LB,NV,NU,NB'                 Binary LOB field
```

   This defines field L1 as a large object field. The NV and NB options specify that this field is not subject to character conversion, nor to the compression of trailing blanks. The NU option specifies that this field is null-suppressed (fields with the NB option must also have either the NU or NC option defined). In effect, Adabas will not modify the provided field values in any way. For further information about the LB field option and all field options (including the NV and NB field options), read *Field Options*.

   If you provide input data for ADACMP to compress, the input can contain either short LB field values (up to 253 bytes) or large LB field values (greater than 253 bytes). In the uncompressed input record, at the place that corresponds to the LB field (in the example, L1), you may provide an empty LB value by specifying an inclusive length of X'00000004' that is not followed by LB data (since the LB value is empty). For further information about the structure of the COMPRESS input data, read *Input Data Requirements*

   Or:
   You can also update an existing FDT using the ADADBS NEWFIELD function with an FNDEF specification defining an LB field. (Adabas Online System also provides equivalent functionality.) If you elect to do this, you do not need to define and load a new *base file* and you can skip the next step and proceed with Step 3 of these instructions.

2. Use the ADALOD LOAD function to load the (possibly empty) base file with LB fields into the database.

   You will load an empty LOB file separately in the next step, although the LOB and base files can be loaded in either order, or even in parallel. For example:

   ```
   ADALOD LOAD FILE=11,NAME='BASE-FILE'
   ADALOD      LOBFILE=12
   ADALOD      MAXISN=100000,DSSIZE=5000B
   ADALOD      ...
   ADALOD      SORTSIZE=100,TEMPSIZE=100
   ```

   The LOBFILE parameter specifies the file number of the LOB file associated with this base file. For complete information about the LOB-related ADALOD parameters, read *LOAD: Load a File*.

   Note:
   A base file-LOB file pair can also be established by just loading a *LOB file*, if a file with LB fields (*base file*) is already present.

   Specify the //DDAUSBA output data set from the ADACMP COMPRESS step as //DDEBAND input data set for the ADALOD LOAD function.

3. Use the ADALOD LOAD function to load an empty *LOB file* and associate it with the *base file* you loaded in the previous step or that you updated in Step 1. (The LOB and base files can be loaded in either order, or even, in parallel.)

   ```
   ADALOD LOAD FILE=12,LOB,NAME='LOB-FILE'
   ADALOD      BASEFILE=11
   ADALOD      MAXISN=500000,DSSIZE=500000B,NISIZE=4000B,UISIZE=40B
   ADALOD      ISNREUSE=YES,DSREUSE=YES,INDEXCOMPRESSION=YES
   ADALOD      ...
   ADALOD      SORTSIZE=100,TEMPSIZE=100
   ```

   The LOB file type indicates that ADALOD should load a LOB file with a predefined FDT. No //DDEBAND input data set need be supplied if you are loading an empty LOB file.

   The BASEFILE parameter specifies the file number of the *base file* associated with this LOB file. For complete information about the LOB-related ADALOD parameters, read *LOAD: Load a File*.

   Note:
   A base file-LOB file pair can also be established by just loading a *LOB file*, if a file with LB fields (*base file*) is already present.

4. Run a database report using ADAREP and see how the base file and LOB file are shown in the report.

5. Reevaluate and adjust the Adabas nucleus (ADARUN) parameters NISNHQ, NH, LP, LDEUQ, LWP, NAB, and LU to make sure the nucleus will have sufficient resources for processing LB fields. For more information about these parameters, read *Adabas Initialization (ADARUN Statement)*. For information on how these parameters might need adjusting, read *Migrating From Previous Adabas Versions*

6. Write or adjust an application program to load data, including LB field values, into the base file. Behind the scenes, Adabas will store LB field values (except for very short ones) in the LOB file, but this is transparent to your application. Commands in application programs should always be directed

against the base file; application programs need neither know nor care about the existence of a LOB file.

- One way to work with LB fields is using Natural 4.2. The LB field (for example, L1) of the LOB file is mapped to a dynamic variable in Natural. Otherwise, your application program can be written in much the normal way. Natural will automatically issue the Adabas calls in the format necessary to deal with large LB field values.

- Alternatively, your application program can issue direct calls against Adabas. For calls with *large* LB field values (more than 32 KB of data), you must use the ACBX direct call interface of Adabas 8. (Calls using the ACB direct call interface can be used if all specified LB field values fit into a single 32 KB buffer.) For more information, read *Specifying an ACBX Interface Direct Call*. For example, you could specify the following information in the ACBX direct call:

Direct Call Component	**Specification**

ACBX	`ACBXCMD = N1 (insert record)`
	`ACBXFNR = 11 (base file number)`
	`...`

First format/record buffer segment pair	`FB1 = 'AA,8,A,`**`L1L,4,B`**`,...,AZ,250,A.'`
	`RB1 = 'KEY-1   ' `**`X'000186A0'`**` ... 'Some arbitrary data...'`

Second format/record buffer segment pair	`FB2 = '`**`L1,*`**`.'`
	`RB2 = X'100,000 bytes of binary data'`

The locations and lengths of the two buffer segment pairs must be given in four Adabas buffer descriptions (ABDs), which are not shown here. For more information, read *Adabas Buffer Descriptions (ABDs)*

The length indicator (L) for the LB field in the first format buffer segment (`L1L,4,B`) specifies that the length of the L1 field value proper is stored at the corresponding place in the first record buffer segment, which indicates an LB field value length of 100,000 bytes (in hex, 186A0). For more information about the length indicator, read *Length Indicator (L)*.

The asterisk (*) length notation in the second format buffer segment (`L1,*`) specifies that the LB field value proper (without any further length information) is stored at the corresponding location in the second record buffer segment. For more information about asterisk length notation, read *Asterisk (*) Length Notation* .

LOB File Management

LOB files must be managed independently from their associated base files. For example, using ADADBS REFRESH to refresh the base file will not automatically refresh the LOB file; a separate, independent ADADBS REFRESH job for the LOB file must be run independently. Likewise, Adabas Online System (AOS) and Adabas Manager file functions must be applied separately to the LOB file from the base file.

Processing LOB Field Segments

LOB fields (LB-type fields) can be processed in segments (parts) using two different mechanisms:

- You can use a special *LOB segment notation* in the format buffer of an Adabas direct call to select part of a LOB field (a LOB segment) that should be processed by the call.

- You can read or write a LOB value in parts (LOB segments) from left to right using multiple A1 or L1/L4 calls.

This section describes each of these processing mechanisms:

- Format Buffer LOB Segment Notation Processing

- Multiple Call LOB Segment Processing

Format Buffer LOB Segment Notation Processing

You can select an entire LOB field or part of a LOB field (a LOB segment) in the format buffer of a direct call using the following syntax:

```
field-name [index-or-range] [(bytenum,LOBlength[,LOBlength2])]
```

The last part of this syntax `(bytenum,LOBlength[,LOBlength2])` is used to identify the segment of a LOB field that should be processed. This part of the syntax is therefore called *LOB segment notation*. For complete information on this syntax, read *Selecting LOB Values or LOB Value Segments* .

Using LOB segments you can:

- Read only a part of a LOB value;

- Delete or replace the rightmost part of a LOB value;

- Replace a LOB segment with a new segment of equal length.

This section describes the rules for each of these uses of a LOB segment.

- Reading a LOB Segment
- Maintaining the Rightmost Part of a LOB
- Replacing a LOB Segment

Reading a LOB Segment

To read a segment of a LOB value, the `(bytenum,LOBlength)` section of LOB segment notation is used. The following rules apply:

1. The number of bytes specified by *LOBlength* at the corresponding position in the record buffer are filled with data from the database. If *LOBlength* is zero, no data is put into the record buffer.

2. The LOB value segment to be read starts at the position in the LOB value specified by *bytenum* and is *LOBlength* bytes long. If an asterisk (*) is specified for *bytenum*, the segment to be read starts at the current position in the LOB value:

- For an L1 or L4 command with L specified in the Command Option 2 field (ACBCOP2 or ACBXCOP2) of the Adabas control block, the current position is derived from the ISN Lower Limit field (ACBISL or ACBXISL) of the Adabas control block, which specifies the number of bytes preceding the LOB value segment to be read. If the current position points past the end of the value, response code 3 (ADARSP003) is issued, indicating the end of the LOB read sequence. An ISN Lower Limit value of zero corresponds to byte number 1. Response code 3 (ADARSP003) is given if the ISN Lower Limit value is greater than or equal to the length of the LOB value.

  For an L1 or L4 command with the L option, the new current position after the read of the segment is returned in the ISN Lower Limit field. It equals the old current position plus the *LOBlength*. The new current position may point past the end of the LOB value.

- For a read or search command without the L option, the current position is defined to be the leftmost byte of the LOB value (byte number 1). No response code 3 (ADARSP003) is given if the LOB value is empty (i.e., has length zero).

3. If the ending position points past the end of the LOB value, the LOB value is considered to be padded with a sufficient number of blanks to make the padded value include the entire segment requested.
4. The record buffer space associated with the format element is filled with the requested segment of the LOB value, including any necessary padding with blanks.

Maintaining the Rightmost Part of a LOB

To delete the rightmost part of a LOB value, or to replace the rightmost part of a LOB value with new data, the LOB segment notation `(bytenum,LOBlength)` is used. The following rules apply:

1. The data to be deleted from the LOB value starts at the insertion point identified by *bytenum* and extends to the end of the LOB value. New data, if any, is inserted at the same insertion point. If an asterisk (*) is specified for *bytenum*, the insertion point is the current position in the LOB value:

   - For an L1 or L4 command with L specified in the Command Option 2 field (ACBCOP2 or ACBXCOP2) of the Adabas control block, the current position is derived from the ISN Lower Limit field (ACBISL or ACBXISL) field of the Adabas control block, which specifies the number of bytes preceding the LOB value segment to be replaced. An ISN Lower Limit value of zero corresponds to byte number 1. For an A1 command with the L option, the new current position after the replacement of the segment is returned in the ISN Lower Limit field. It equals the old current position plus *LOBlength*. If no trailing blanks were removed after the segment was replaced, the new current position points to the end of the LOB value. If trailing blanks were removed, the new current position points past the end of the LOB value stored in the database.

   - For a store or update command without the L option, the current position is defined to be the leftmost byte of the LOB value (byte number 1).

2. Any existing data in the LOB value at and to the right of the insertion point is deleted or replaced. If the insertion point is at the end of the LOB value, no data is deleted and the new data is appended to the LOB value.

3. If the insertion point lies past the end of the LOB value, the value is padded with a sufficient number of blanks to extend to the insertion point.

4. The number of bytes specified by *LOBlength* from the corresponding position in the record buffer, constituting the segment to be written, are stored at the insertion point. If *LOBlength* is zero, no bytes are stored.

5. Unless the LOB field has the NB option (no blank compression), any blanks at the end of the resulting LOB value are removed. If no data or only blanks were inserted at the end of the value, the compression operation removes any trailing blanks to the left of the insertion point.

   Note:
   For a LOB field with the NB option (no blank compression), the length of the LOB value after the write operation always equals the position of the insertion point (minus 1) plus *LOBlength*. If the insertion point lies past the end of the LOB value and no data is to be inserted (*LOBlength* is zero), the LOB value is padded with blanks up to the insertion point. (For a LOB field without the NB-option, this padding does not make any difference, as the inserted blanks are removed by the compression operation.)

Replacing a LOB Segment

To replace a segment of a LOB value with a new segment of the same length, the extended LOB segment notation `(`*`bytenum`*`,`*`LOBlength`*`,`*`LOBlength2`*`)` is used (where the value of *LOBlength2* must equal the value of *LOBlength*). The following rules apply:

1. The LOB value segment to be replaced starts at the insertion point identified by *bytenum* and is *LOBlength* bytes long. If an asterisk (*) is specified for byte-number, the insertion point is the current position in the LOB value:

   - For an A1 command with L specified in the Command Option 2 field (ACBCOP2 or ACBXCOP2) of the Adabas control block, the current position is derived from the ISN Lower Limit field (ACBISL or ACBXISL) field of the Adabas control block, which specifies the number of bytes preceding the LOB value segment to be replaced. An ISN Lower Limit value of zero corresponds to byte number 1. For an A1 command with the L-option, the new current position after the replacement of the segment is returned in the ISN Lower Limit field. It equals the old current position plus *LOBlength*. The new current position may point past the end of the LOB value stored in the database.

   - For a store or update command without the L option, the current position is defined to be the leftmost byte of the LOB value (byte number 1).

2. If the ending position points past the end of the LOB value, the LOB value is considered to be padded with a sufficient number of blanks to make the padded value include the entire segment.

3. The number of bytes specified by *LOBlength* from the corresponding position in the record buffer, constituting the segment to be written, are stored at the insertion point, replacing any existing data of the same length stored at and past this position. If *LOBlength* is zero, no data is stored.

4. Unless the LOB field has the NB option (no blank compression), any blanks at the end of the resulting LOB value are removed. If only blanks were inserted at the end of the value, the compression operation removes any trailing blanks to the left of the insertion point.

Note:
For a LOB field with the NB option (no blank compression), if the insertion point lies past the end of the LOB value and no data is to be inserted (length is zero), the LOB value is padded with blanks up to the insertion point.

Multiple Call LOB Segment Processing

The command option L in the Command Option 2 field of L1/L4 and A1 commands allows you to read and write a LOB value from left to right with multiple calls using a constant format. This section describes this processing.

- Reading a LOB Value using Multiple Calls
- Writing a LOB Value using Multiple Calls
- L Option Rules
- Current (Asterisk) Positioning in LOB Values

Reading a LOB Value using Multiple Calls

To read a LOB value from left to right with multiple calls using a constant format, an application should use a sequence of L1 (or L4) commands on the record containing the LOB value. The format buffer should use the LOB segment notation `(`*`bytenum,LOBlength`*`)` to position each read call. It can reference the current position in the LOB field using an asterisk (*) in the *bytenum* specification (current positioning notation). The L option should be specified in the Command Option 2 field of the Adabas control block to request tracking of the current position in the LOB value using the ISN Lower Limit field of the Adabas control block. Here is a recommended command sequence for sequential LOB reading:

1. The application uses any read-type command to locate and read the target record, excluding the LOB value. It may put the record in shared or exclusive hold status to prevent parallel updates to the current record while the following read sequence on the LOB field is in progress.

   Note:
   We strongly recommend that you protect the integrity of a LOB value while it is being read piecemeal in multiple calls. This may be done using application rules that prevent possible concurrent LOB updates (for example, the application might ensure that LOB values are never changed and become visible to readers only once they have been fully stored), or by putting the record containing the LOB value in shared or exclusive hold status. If you are reading a LOB value piecemeal and you update the LOB value during the read sequence, it is your responsibility, too, to ensure that the value being read is consistent in terms of the application logic.

2. The application issues an L1/L4 command to read the first segment of the LOB value. The format buffer references the LOB field using an asterisk in *bytenum* (this is current positioning notation). The L option is set in the Command Option 2 field of the control block to indicate that the current position in the LOB value should be tracked using the ISN Lower Limit field of the control block. Initially, the application sets the ISN Lower Limit field to zero, which is interpreted as position 1 of the LOB field. Adabas increments the ISN Lower Limit field by the number of bytes read (specified in *LOBlength*).

3. Until response code 3 (ADARSP003) is issued to indicate that the end of the LOB value has been reached, the application issues further L1/L4 commands with the same format buffer as in step 2. The L option is set in the Command Option 2 field of each call to indicate that LOB field reading should continue at the current position in the LOB value, which is identified by the ISN Lower Limit field. When the application is only reading the LOB value from left to right, it should not modify the ISN

Lower Limit field; Adabas will increment the ISN Lower Limit setting each time by the number of bytes read.

4. When done with reading the LOB value, the application may continue to work on the current record (for example, update it) or go on to the next record.

Writing a LOB Value using Multiple Calls

To write a LOB value from left to right with multiple calls using a constant format, an application should use a sequence of A1 commands on the record containing the LOB value. The format buffer should use the LOB segment notation `(`*`bytenum`*`,`*`LOBlength`*`)` to position each write call. It can reference the current position in the LOB field using an asterisk (*) in the *bytenum* specification (current positioning notation). The L option should be specified in the Command Option 2 field of the Adabas control block to request tracking of the current position in the LOB value using the ISN Lower Limit field of the Adabas control block. Here is a recommended command sequence for sequential LOB writing:

1. The application uses an N1/N2 or A1 command to store or update the target record, initially with an empty LOB value. The mandatory exclusive hold status of the record prevents parallel updates to the current record while the following write sequence on the LOB field is in progress.

   Note:
   The exclusive hold status of the record does not prevent parallel reads of the current record, including the LOB value, unless those reads request shared or exclusive hold status for the record, too. If the commands used to write or update the LOB value are spread over more than one transaction, it is the user's responsibility to ensure that either the LOB value cannot be updated between transactions or the next transaction will detect and adapt to a change of the value.

2. The application issues an A1 command to write the first segment of the LOB value. The format buffer references the LOB field using an asterisk in *bytenum* (this is current positioning notation). The L option is set in the Command Option 2 field of the control block to indicate that the current position in the LOB value should be tracked using the ISN Lower Limit field of the control block. Initially, the application sets the ISN Lower Limit field to zero, which is interpreted as position 1 of the LOB field. Adabas increments the ISN Lower Limit field by the number of bytes written (specified in *LOBlength*).

3. Until the entire LOB value has been written, the application issues further A1 commands with the same format buffer as in step 2. The L option is set in the Command Option 2 field of each call to indicate that LOB field writing should continue at the current position in the LOB value (at the end of the value), which is indicated by the ISN Lower Limit field. When the application is only writing the LOB value from left to right, it should not modify the ISN Lower Limit field; Adabas will increment the ISN Lower Limit setting each time by the number of bytes written.

4. When done with writing the LOB value, the application may continue to work on the current record (for example, read or update it again) or go on to the next record.

L Option Rules

The following rules apply to the use of the L option in Command Option 2 of the Adabas control block in conjunction with the use of current (asterisk) positioning in the format buffer:

1. The Command Option 2 field can only be set to L in the control blocks for L1, L4 and A1 commands. If L is specified, the format buffer must contain exactly one format element for a LOB field segment using current (asterisk) positioning.

2. When the L option is specified, the ISN Lower Limit field in the Adabas control block must be a number in the range from 0 to 2,147,483,647 (i.e., 2^{31-1}). This number gives the current position in the LOB value, specified as the number of bytes preceding the target segment. Adabas increments the ISN Lower Limit field by the *LOBlength* specified in the format buffer. (The returned ISL value lies in the range from 0 to 4,294,967,294.)

3. When the L option is specified in a read command (L1 or L4), response code 3 (ADARSP003) with a new subcode is given if the current position in the LOB value points past the end of the value (if the value in the ISN Lower Limit field is greater than or equal to the length of the LOB value).

4. The L option cannot be specified with the I or N options of an L1/L4 command, nor with the V option of an A1 command. (The V option is defined in Adabas on open systems only and is related to ADAM files.) All of these options are specified in the Command Option 2 field and they also imply the processing of a new record, whereas the L option is used for the reading or writing of (different parts of) the same record with multiple calls.

5. The L option cannot be specified with the M or P options. Instead of reading multiple consecutive LOB value segments in a single command using multifetch or prefetch, the application should rather read a single larger segment.

Current (Asterisk) Positioning in LOB Values

When an L1/L4 and A1 command sequence is performed with the L option (using the current (asterisk) position notation in the format buffer element), the current position in the LOB value being read or written must be tracked. Multiple LOB read/write sequences may be in progress in parallel, with a different position being current in each LOB value.

The ISN Lower Limit (ISL) field in the Adabas Control Block is used to keep track of the current position in the LOB value being read or written. Prior to the call, the application program sets the ISN Lower Limit field to the number of bytes preceding the LOB value segment to be read, replaced, or deleted. After the call, Adabas increments ISN Lower Limit by the length specified for the LOB field segment in the format buffer.

To read an existing (or write a new) LOB value from left to right using a format element with current (asterisk) positioning, your application should initially set the ISL field to zero for the first L1/L4 or A1 command of the read (or write) sequence. For the subsequent calls, it should leave the ISN Lower Limit field unchanged or it must reproduce the value previously returned by Adabas.

The ISN Lower Limit field is used to determine the current position in a LOB value only if the L option is specified, too. For a call without the L option, the current position is defined to be byte number 1 of the value and the ISL field is neither used nor updated in the course of the operation on the LOB value segment. Only L1, L4, and A1 commands can be issued with the L option.