Architectural Changes

This chapter describes the architectural changes of Adabas 8. For a complete list of the enhancements and other aspects of this release, read *Adabas 8.1.1 Release Notes*.

- Lifted Limits
- Spanned Record Support
- Large Object (LB) Field Support
- Long Alpha (LA) Field Changes
- FDT Changes

Lifted Limits

- Logical Extent Limit
- Physical Extent Limit
- MU and PE Limit

Logical Extent Limit

The limit of five logical file extents for each Adabas file extent type has been lifted. The maximum number of logical file extents that you can now define is derived from the block size of the first Associator data set (DDASSOR1). The extent information is stored in a variable section of the FCB. New extents can be added now until the used FCB size reaches the block size of the Associator data set. For example, on a standard 3390 device type, a file could have more than 40 extents of each type (or there could be more of one type if there are less for another).

Physical Extent Limit

The Associator and Data Storage components of your Adabas database may now each contain more than five physical extents. A new maximum of 99 physical extents is now set for each, however, your actual real maximum could be less because, in the same manner as the logical file extent maximum, it is also derived from the block size of the first Associator data set (ASSOR1). For example, on a standard 3390 device type, there could be more than 75 Associator, Data Storage, and DSST extents each (or there could be more of one extent type if there are less for another).

MU and PE Limit

The number of occurrences of each MU field or each PE group in a record has been increased from 191 to 65,534. However, the actual limit is derived from the maximum Data Storage record length (the ADALOD MAXRECL parameter), which defaults to the size of the Data Storage block minus 4.

Note:
The use of more than 191 MU or PE fields in a record must be explicitly allowed for a file (it is not allowed by default). This is accomplished using the new ADADBS MUPEX function or the ADACMP

COMPRESS MUPEX and MUPECOUNT parameters.

All MU fields and PE groups and other fields must fit into one compressed record. If you are using spanned records (introduced with Adabas 8), more MU fields and PE groups can be stored.

If a file has been established with extended MU or PE limits, you should not read the occurrence count of an MU field or PE group into a one-byte field in the record buffer. If you try, Adabas returns response code 55 (ADARSP055), subcode 9. Therefore, any application program that reads the occurrence count using an `xxC` element in the format buffer (for example, FB=’MUC.’ or FB=’MUC,1,B.’) must be changed to read the occurrence count into a field with two or more bytes (for example, FB=’MUC,2,B.’ or FB=’MUC,4,B.’).

Spanned Record Support

This release of Adabas introduces the concept of spanned records. In the database, the logical record is split into a number of physical records, each part fitting into a single Data Storage (DS) block. The resulting physical records are each assigned individual ISNs. The first physical record is called the *primary record* and contains the beginning of the compressed record and is assigned a *primary ISN*. The remaining physical records are called *secondary records* and contain the rest of the data of the logical record. Secondary records are assigned *secondary ISNs*. These ISNs do not affect the user ISNs assigned when using the N2 command or the ISNs used when using the I option of the L1 command. If spanned records are used, a secondary address converter is used to map the secondary ISNs to the RABNs of the Data Storage blocks where the secondary records are stored.

A spanned record is comprised of one primary record and one or more secondary records. However, the number of segments in a spanned record is limited. The Adabas nucleus allows up to five physical records (one primary record and four secondary records) in a spanned record.

Spanned records are not directly visible to application programs. Applications always address spanned records via the ISNs of the primary records.

Spanned records are also supported in expanded Adabas files and in multi-client files.

Note:
Spanned record support must be explicitly allowed for a file. You can do this using the ADADBS RECORDSPANNING function or the SPAN parameter of ADACMP COMPRESS.

This section covers the following topics:

- Spanned Record Structure
- Identifying Spanned Records
- Secondary Record Segmentation
- Padding Factors
- Spanned Record ISN Use
- ADARUN Parameters Affected

- Reporting on Spanned Records
- Securing Spanned Records

Spanned Record Structure

A spanned record is comprised of one primary record and one or more secondary records. The primary and secondary records in a spanned record are connected using their ISNs. The header of each physical record contains the ISN of the current record, the ISN of the primary record, as well as the ISN of the next secondary record. In addition, the header indicates whether the current record is the primary record or a secondary record.

The header of each physical record also provides the length of the record -- even if it is a segmented record (in which case, it is the length of the segment).

Identifying Spanned Records

Files can contain spanned records only if this has been explicitly requested via the SPAN parameter of ADACMP COMPRESS, the RECORDSPANNING function of ADADBS or the equivalent Adabas Online System function. The ADAREP database report and the Adabas Online System report functions indicate whether or not a file has been defined to allow spanned records.

The SPAN attribute of a file is retained in an ADAULD UNLOAD function. In other words, when a file is unloaded, deleted, and reloaded, its support for spanned records remains unchanged.

Similar rules hold for files that allow more than 191 MU or PE occurrences. For more information on identifying MU and PE occurrences greater than 191 in a compressed record, read *Identifying MU and PE Occurrences Greater Than 191 in Compressed Records*.

Secondary Record Segmentation

Secondary records are segmented either by field or by byte. For performance reasons, segmentation is done by field whenever possible. However, when any non-LB (large object) type field is larger than the data storage block size, the record is split at the byte level. If a field is larger than the remaining space in the data storage block, but smaller than the data storage block size, than the field is split at the field level and not at the byte level. The header of each secondary record indicates which type of segment record it is.

Padding Factors

Padding factors are generally ignored for spanned records, in an attempt to fully use the block. So it is frequently listed as zero on reports. The padding factor is only used in the last, short, segment of a spanned record.

Spanned Record ISN Use

Primary and secondary records are addressed by Adabas using address converters (AC). However, the primary address converter maps only the ISNs of primary records to the RABNs of their corresponding Data Storage blocks. If spanned records are used, a secondary address converter is used to map the secondary ISNs to the RABNs of the Data Storage blocks where the secondary records are stored. Therefore, spanned records have no affect on the index structure, since there is still only one index for each record.

Separate ISN ranges are maintained for primary and secondary ISNs. Wherever an ISN is stored or handled, it distinguishes between whether the action is for a primary or a secondary ISN.

All commands should be specified using the primary record's ISN; secondary record ISNs are kept hidden and cannot be used. Physical sequential commands will automatically skip the secondary records in Data Storage. Read commands that specify secondary ISNs will receive an error (response code 113, ADARSP113).

The ISN of the primary records are included in TOPISN and MAXISN values. Secondary record ISNs are not. Secondary ISNs are included in the MINSEC and MAXSEC values instead. A file containing spanned records can be loaded by specifying an MINISN value, but the MINISN must refer only to a primary record ISN (never a secondary record ISN).

ADARUN Parameters Affected

The following ADARUN parameters may need to be increased to support files with spanned records.

- The number of ISNs in the hold queue per user (NISNHQ parameter) may need to be increased as the number of spanned records to be updated also increases.
- The length of the Adabas work pool (LWP) may also need to be increased since space is needed to store both the before and after image of the spanned record and to support several update threads running in parallel. Space may also be needed to accommodate larger descriptor value tables (up to 65,534 occurrences of descriptors in PE groups are permitted).

Reporting on Spanned Records

Maximum record length statistics have no relevance with spanned files. Utilities that report on the maximum record length will now report that the statistics as "N/A" (not applicable). The FCB will contain high values in the maximum record length field for a file that is using spanned records.

Securing Spanned Records

Files containing spanned records can be ciphered and protected with security-by-value. If the primary record's ISN is referenced, all secondary segment records must be read, and therefore, processing is time-sensitive.

Large Object (LB) Field Support

Your Adabas files can now include large fields (known as *large object (LB)* fields) that can contain up to 2,147,483,643 bytes (about 2 GB) of data. I

This section covers the following topics:

- Adabas 8 Handling of LB Fields
- Defining Large Object (LB) Fields in the FDT
- Format Buffer Support of LB Fields

- Deciding Between Long Alpha and Large Object Fields

Adabas 8 Handling of LB Fields

Both binary and character LB fields are supported in Adabas 8. Adabas does not modify binary LB fields in any way. The identical LB field binary byte string that was stored is what is retrieved when the LB field is read. For more information about defining binary LB fields, read *Defining Binary LB Fields*.

Adabas performs character code conversion on character large objects according to Universal Encoding Support (UES)-related definitions for the database, file, and user. The presence of the new field option, NB (no blank compression) in the LB field definition indicates whether on not Adabas removes trailing blanks in character LB fields.

Defining Large Object (LB) Fields in the FDT

LB fields must be defined in the Field Definition Table (FDT) using a new LB field option. The field format must be "A" (alphanumeric).

The default field length of an LB field must currently be defined as zero. Future releases of Adabas may consider supporting nonzero default field lengths greater than 253 for long alpha (LA) and large object (LB) fields.

An LB field *cannot* be:

- A descriptor or parent of a special (phonetic, sub-, super-, or hyper-) descriptor.
- Defined with the FI or LA options.
- Specified in a search buffer or in its format selection criteria.

An LB field may be:

- Defined with any of the following options: MU, NB (a new no blank compression option), NC, NN, NU, or NV
- Part of a simple group or a PE group.

This section covers the following topics:

- NV Option with LB Fields
- New NB Option
- Defining Binary LB Fields
- LB Field Examples

NV Option with LB Fields

When specified, the NV (no conversion) option indicates that no conversion should occur when a field value is provided by or returned to a machine with a different architecture than the Adabas server.

New NB Option

A new NB option can be used with LA and LB fields to control blank compression. When specified, the NB option indicates that Adabas should not remove trailing blanks for the field. If you specify the NB option for a field, you must also specify the NU or NC option for the field; NB processing requires the use of NC or NU as well. Future releases of Adabas may consider allowing the NB option for regular alphanumeric or wide-character fields.

Defining Binary LB Fields

A binary LB field is defined by specifying both the NV and the NB options, indicating that Adabas will not modify the field values in any way during storage.

Note:
Binary LB fields are not defined using format B, because format B can imply byte swapping in some environments with different byte orders. Byte swapping does not apply to binary LB fields.

LB Field Examples

The following table provides some valid example of FDT definitions for LB fields:

FDT Specification	Description
`1,L1,0,A,LB,NU`	Field L1 is a null-suppressed character LB field
`1,L2,0,A,LB,NV,NB,NU,MU`	Field L2 is a multiple-value, null-suppressed, binary LB field.

Format Buffer Support of LB Fields

In general, LB fields can be specified in format buffers in much the same way as regular fields are specified. This section describes the exceptions and special features of specifying LB fields in format buffers:

- Range Notation
- Occurrence Index
- Specifying LB Field Formats
- Specifying the Lengths of LB Fields

Range Notation

For multiple-value LB fields and LB fields within periodic groups, you can use the range notification to specify a fixed number of occurrences of a field. For example, the following format buffer specification will select the first 10 values of LB field L2:

```
FB='L21-10.'
```

However, you cannot specify the open-ended 1-N notation to select all occurrences of the field. For example, the following format buffer is *not* valid:

```
FB='L21-N.'
```

The 1-N notation is not supported for LB fields.

Occurrence Index

For multiple value LB fields and LB fields within periodic groups, you must specify a specific occurrence of the field. In the following example, the first value of multiple-value LB field L2 is selected:

```
FB='L21.'
```

Likewise, in the following example, the fifth value of LB field L3 in its second PE-group instance is selected:

```
FB='L32(5).'
```

However, you cannot specify the base field without an occurrence index. For example, the following format buffer specification is *not* valid if L2 is a multiple-value field:

```
FB='L2.'
```

Specifying LB Field Formats

You can specify A (alphanumeric) in the format buffer of an LB field. If no format is specified, the format defined for the LB field in the FDT is used.

Specifying the Lengths of LB Fields

There are three methods you can use in a format buffer specification to set the length of an LB field in the corresponding record buffer:

- Explicit Length Specification
- Zero Length Specification
- Asterisk (*) Length Notation

Note:
If no length is specified for an LB field in its format buffer specification, the default length (zero) from the FDT is used. In this case the rules for zero length specification are used, as described elsewhere in this section.

Explicit Length Specification

You can explicitly specify the length in the format buffer. If a nonzero length is specified in the format element for the LB field, the length specifies the amount of space allotted for the LB value in the record buffer. The maximum valid length that can be specified is 2,147,483,647.

In the following example, 50,000 bytes are allotted for LB field L1 in the record buffer and 10 bytes are allotted for field AA:

```
FB='L1,50000,AA,10,A.'
```

The record buffer must provide sufficient space for the entire field if its format element includes an explicit length setting. If sufficient space is not provided, errors (response code 53, ADARSP053) will result.

Zero Length Specification

If a zero length is specified in the format element, the amount of space available for the LB field values in the record buffer is variable and depends on the actual LB value. In this case, the first four bytes of the LB value in the record buffer are used to store the actual length of the LB field, including the four-byte length itself (the LB value length plus four). The maximum valid inclusive length is 2,147,483,647. In the case of LA fields, only a two-byte length is stored in the record buffer.

In the following example, the record buffer for LB field L1 will first contain the four-byte length of the L1 field, followed by the actual value of the L1 field. In addition, 10 bytes are allotted for field AA, the value of which immediately follows the value of the L1 field.

```
FB='L1,0,AA,10,A.'
```

The record buffer must provide sufficient space for the entire field if its format element includes a zero length setting. If sufficient space is not provided, errors (response code 53, ADARSP053) will result.

Asterisk (*) Length Notation

For LA and LB fields only, you can specify an asterisk (*) instead of a length in the format element. This indicates that the amount of space available for the LB field value in the record buffer is variable and depends on the actual value of the LB field. However, unlike the zero length specification setting, *no* four-byte length field precedes the LB field value in the record buffer; the record buffer area corresponding to the LB format element only contains the value of the LB field. The actual LB field value length *should be* retrieved for read commands and *must be* specified for update commands using the new format buffer length indicator, *L*. For more information about the length indicator, read *Length Indicator (L)*, elsewhere in this guide.

In the following example, the record buffer for LB field L1 contains only the value of the L1 field, followed by the value of the AA field for which 10 bytes have been allotted.

```
FB='L1,*,AA,10,A.'
```

In the following example, the record buffer for LB multi-value field L2 contains the first ten values of L2.

```
FB='L21-10,*.'
```

The record buffer is not necessarily required to provide sufficient space for the entire field if its format element includes an asterisk length setting. However, in read command processing, the field value can be truncated if *both* of the following conditions are met:

- The record buffer space available is insufficient for the field value.
- A field with asterisk notation is specified at the *end* of the format buffer.

In these conditions, no error is returned. If this were the case in the second example above (`FB='L21-10,*.'`), Adabas would truncate the ten values to be read down to the length of the corresponding record buffer segment. (The truncation occurs from right to left; that is, the last value is truncated first; if the remaining space is still insufficient, the second-to-last value is truncated, and so on.) In extreme cases, if no space is available at all for the field value, the value is truncated down to zero bytes.

In the first example above (`FB='L1,*,AA,10,A.'`), if the record buffer segment is too short, no truncation occurs because this is not allowed for fields specified with a fixed length or length of zero (0). Rather, the nucleus returns response code 53 (ADARSP053 - record buffer too small).

Only read commands executed by the Adabas nucleus may truncate values specified with the asterisk notation; no truncation occurs in update commands. In addition, the ADACMP utility does not truncate values specified with the asterisk notation.

Deciding Between Long Alpha and Large Object Fields

The following table comparing pertinent LA and LB field features may help you decide which to use when defining fields for your database.

Feature	LA Field Behavior	LB Field Behavior
Zero field length specification in format buffers	Two bytes in the corresponding record buffer area are used to store the actual length of the LA field.	Four bytes in the corresponding record buffer area are used to store the actual length of the LB field.
Data record storage	Alphanumeric and wide-character fields are stored within the compressed record. All long values must fit into the same compressed record. The maximum length of simple or spanned data records limits the number and lengths of long values that can be stored. This can be a problem if multiple long values are contained in a record.	Some LB field values (those larger than 253 bytes) are stored offline in a separate large object file (the LOB file) and only references to the LB field values in the LOB file are included in the data record. This allows for storing more long objects for a single data record than using normal or LA fields. However, the performance overhead at runtime and for file maintenance is increased for LB fields because of this behavior. Smaller LB field values (up to 253 bytes) are stored directly in the compressed record. This improves performance for small values, but also limits the number of small LB field occurrences that can be stored in the same compressed record.
Asterisk (*) field length notation in format buffers	Supported for LA fields of any length.	Supported for LB fields of any length.

Feature	LA Field Behavior	LB Field Behavior
Maximum length of any stored object does *not* exceed 16,381 bytes	Alphanumeric or wide-character LA field can be used. This avoids the overhead of LB fields, but limits the number of such fields that can be stored in a single record.	Alphanumeric LB field can be used.
Maximum length of any stored object exceeds 16,381 bytes	Not supported.	Supports objects with sizes larger than 16,381 bytes.
So many large objects that they will not fit in a single simple or spanned data record	Not supported.	Supports multiple large objects.

Long Alpha (LA) Field Changes

The following updates have been made to long alpha (LA) fields in this release:

- FDT definitions of LA fields can now specify the new NB option to control blank suppression for the field. For more information about the NB option, read *FDT Changes*.
- LA fields can now specify fixed field lengths greater than 253 in format buffers. For more information, read *Format Buffer Changes*.
- The new asterisk (*) field length specification is supported for LA fields in format buffers. For more information about the asterisk field length specification, read *Asterisk (*) Length Notation*.
- The new format buffer indicator (*L*), referred to as the *length indicator*, can now be used to retrieve or specify the actual length of an LA field value. For more information, read *Format Buffer Changes*.
- For multiple value LA fields and LA fields within periodic groups, you cannot specify the base field without an occurrence index or with a "1-N" index. You must use a specific index or index range. For example, if L2 is an LA field with the MU option, the following format buffer specifications are *not* valid:

  ```
  FB='L21-N.'

  FB='L2.'
  ```

  However, the following format buffer specification is valid, requesting the first three values of field L2:

  ```
  FB='L21-3.'
  ```

For an analysis of the differences between LA and LB fields, read *Deciding Between LA and LB Fields*.

FDT Changes

The following changes have been made for field definitions in the FDT:

- The internal FDT structure in version 8 has increased and these larger FDTs may use more than four Associator blocks. The additional blocks required for a larger FDT are automatically allocated from the Associator free space. The fixed space for FDTs in the Associator will remain reserved to accommodate backward compatibility and conversion.
- A new LB field option (large object field option) allows you to identify a field as an LB field. For more information, read *Defining Large Object (LB) Fields in the FDT*.
- A new NB option (no blank compression option) allows you to stop Adabas from removing trailing blanks from LA and LB fields. For more information, read *Defining Large Object (LB) Fields in the FDT*.
- The existing NV option can also be specified for LB fields. If specified, the LB field is not subject to character code conversion, and it cannot be converted from A-format to W-format and vice versa. For more information, read *Defining Large Object (LB) Fields in the FDT*.