software AG

Adabas

Planning for Adabas 8

Version 8.1.4

June 2014

Adabas

This document applies to Adabas Version 8.1.4.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: ADAMF-PLANNING-814-20140626

Table of Contents

Preface

This document provides information useful to customers intending to install and use Adabas 8 and to those planning a migration from previous versions of Adabas.

Caution: The material presented here is subject to change before product release. Software AG reserves the right to change the content of this product and this documentation prior to release and does not guarantee that all functionality mentioned will be implemented. However, this information is sufficiently accurate to allow you to successfully evaluate your upgrade to Version 8.

This document is intended for use by those persons responsible for Adabas 8 configuration and administration. A detailed knowledge of the current Adabas product set is assumed.

This document is organized as follows:

General Information	Provides information on the availability, prerequisites, and documentation of Adabas 8.
Migrating From Previous Adabas Versions	Lists considerations for migrating from prior versions of Adabas to Adabas 8.
Architectural Changes	Describes the architectural changes of Adabas 8.
Utility Changes	Describes the utility changes of Adabas 8.
ADARUN Changes	Describes the ADARUN changes for Adabas 8.
Command Changes	Describes the command changes of Adabas 8.
User Exit Changes	Describes the hyperexit changes of Adabas 8.
Add-On Product Support for Adabas 8	Describes the Adabas add-on product support for Adabas 8.

1 General Information

This chapter provides general information about Adabas 8.

Availability

Adabas 8 is scheduled to be available in the second or third quarter of 2006.

Compatibility

Adabas 8 is fully compatible with currently supported versions of Adabas.

Documentation

The documentation for this release is distributed entirely in HTML and PDF formats. This is a departure from previous versions of Adabas in that no hard copy documentation is provided for any mainframe Adabas documentation.

The Adabas documentation has been completely updated to reflect Adabas Version 8 technical updates. For a complete list of the enhancements and other aspects of this release, read *Adabas Release Notes* for Version 8.1.1.

2 Migrating From Previous Adabas Versions

When planning your migration from prior versions of Adabas to Adabas 8, please consider the following information. For information on Adabas add-on support, read ***Add-On Product Support for Adabas 8***, elsewhere in this chapter.

1. A new direct call interface has been added in Adabas 8 that is fully compatible with the existing Adabas ACB-based direct call interface. The new direct call interface is based on the new Adabas 8 ACBX and ABD structures.

   Existing application programs that use the ACB-based direct call interface can continue to run in the same way, without change. 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 segmented buffers (multiple format and record buffers) available with Adabas 8, you must use the ACBX-based direct call interface. For more information on these, read ***Command Changes***, elsewhere in this guide.
2. Existing files must either be reloaded or reorganized if you want to take advantage of the increased limit for MU and PE fields provided with Adabas 8. You may also need to:
   - increase the work pool (ADARUN LWP parameter)
   - increase the protection area on the Work data set (ADARUN LP parameter)
   - increase the internal format pool (ADARUN LFP parameter).
3. If a file has been established with the extended MU or PE limits available in Adabas 8 (for example, using the ADADBS MUPEX function described in ***Utility Changes*** elsewhere in this guide), and an application program reads the occurrence count of an MU field or PE group using an `xxC` element in the format buffer, verify that the program reads the occurrence count into a record buffer field with two or more bytes (for example, FB='MUC,2,B.') If the program reads the occurrence count into a one-byte field (for example, FB='MUC.' or FB='MUC,1,B.'), it

must be adjusted so that it can deal with two-byte occurrence count values. Adabas returns response code 55, subcode 9 if you only provide a one-byte field in the record buffer for the occurrence count of an MU field or PE group in a file with extended MU/PE limits.

4. If your hyperdescriptors make use of extended MU or PE fields, you will need to reassemble the corresponding hyperexits; with adjustments for the extended MU or PE fields, parameter list, and input parameters. For more information, read ***Hyperexit Changes***, elsewhere in this guide.
5. Large object fields -- that is, fields defined with the new LB option -- are generally stored in the LOB file associated with the file containing the LB fields (base file). Only small LB field values (up to 253 bytes) are stored directly in the base file. In the LOB file, LB field values are partitioned into segments and stored in one or more segment records, each of which (except the last) occupies a whole Data Storage block.

   If you plan to use large object fields, reevaluate your Adabas nucleus (ADARUN) parameters according to the following guidelines:

   - For insert, update, and delete operations involving LB field values, Adabas takes all LOB file segment records involved in hold. The hold operations are counted against the ADARUN NH and NISNHQ parameters. These parameters should therefore be increased depending on the expected maximum number of LOB file segment records that may be updated in parallel transactions.
   - Furthermore, the before and after images of LB field values being inserted, updated, or deleted are written to the protection area on the Work data set. The LP parameter should be increased by the number of Work blocks equivalent to twice the estimated size of LB field values that may be updated in parallel transactions.
   - The LOB file is defined with a unique descriptor (UQ option) that identifies its individual segment records. When updating or deleting LB field values, Adabas needs about 30 bytes of unique-descriptor-pool space per LOB segment record. The LDEUQ parameter should be increased by 30 times the maximum number segment records per large object times the number of LB field values that may be deleted or changed in parallel transactions.
   - The NAB parameter should be increased by the size of the largest LOB value that may be read from or written to the database, times the estimated maximum number of parallel Adabas calls that may read or write LOB values from or to the database.
   - The LU parameter should be increased by the size of the largest LOB value that may be read from or written to the database. Note that the LU parameter does not describe the size of a memory area allocated by Adabas but rather the maximum length of an area of the attached buffers (defined by the NAB parameter) that can be used by a single Adabas call.

   For complete information about ADARUN parameters, read *Adabas Initialization (ADARUN Statement)*, in *Adabas Operations Manual*.
6. If you intend to use either large object (LB) fields or spanned records, ensure that the ADARUN LWP parameter is at least 50,000 bytes times the number of threads (NT parameter). For complete

information about the ADARUN LWP parameter, read *LWP : Length of Adabas Work Pool*, in *Adabas Operations Manual*.

7. Some restrictions exist in the use of utilities under Adabas 8. Be sure to read about the utility enhancements and restrictions in ***Utility Changes*** (elsewhere in this guide) and *Limitations and Restrictions* (in *Adabas Release Notes*) for a description of how each Adabas utility has been updated for Adabas 8 and what the limitations and restrictions are for their use.

3 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 Release Notes* for Version 8.1.1.

Lifted Limits

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 re-

turns response code 55, 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*, in *Adabas Utilities Manual*.

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).

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. In Adabas 8, you can only store and retrieve entire LB fields, however, the ability to retrieve a portion of an LB field or to add and remove data at the end of an LB field is planned for a future Adabas version.

This section covers the following topics:

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***, elsewhere in this guide.

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

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) 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) 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 (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	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

Feature	LA Field Behavior	LB Field Behavior
	lengths of long values that can be stored. This can be a problem if multiple long values are contained in a record.	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.
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***, elsewhere in this guide.
- LA fields can now specify fixed field lengths greater than 253 in format buffers. For more information, read ***Format Buffer Changes***, elsewhere in this guide.
- 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***, elsewhere in this guide.
- 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***, elsewhere in this guide.

- 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***, elsewhere in this guide.

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***, elsewhere in this guide.
- 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***, elsewhere in this guide.
- 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***, elsewhere in this guide.

4 ADARUN Changes

The ADARUN CLOGLAYOUT parameter includes a new possible value, 8, that allows you to specify that the format of the Adabas command log (CLOG) should be Adabas 8 format. ADARUN CLOGLAYOUT=4 settings will no longer run, and setting 4 is no longer supported. Use CLOGLAYOUT=5 instead.

In prior versions of Adabas, you were not allowed to choose between the use of EXCP and EXCPVR when running from an APF-authorized load library. EXCP was always used when running non-APF-authorized; EXCPVR was always used when running APF-authorized. If you wanted to use EXCP when running APF-authorized, you were required to apply special A$- or AY- zaps.

This release introduces a new ADARUN parameter, EXCPVR. Using this parameter, you can specify whether EXCP or EXCPVR is used when running APF-authorized. For complete information on the use of this parameter, read *EXCPVR : Control EXCP or EXCPVR Use*, in *Adabas Operations Manual*.

Finally, updates to the old A$- or AY-zaps will no longer be provided, as the zaps are no longer necessary.

5 Utility Changes

All Adabas utilities have been updated to support the new extended features of Adabas 8. Some of this support appears in the form of new or modified utility parameters. In other cases, support was added internally that does not affect your use of the utility at all.

Note: Be sure to read about the utility restrictions in *Limitations and Restrictions*, in *Adabas Release Notes* for a brief description of what the limitations and restrictions are for utility use in Adabas 8.

This chapter describes the Adabas utilities whose user interface has changed in this release. For complete information about the functions of any Adabas utility, read *Utilities*in *Adabas Utilities Manual*.

ADAACK Utility Changes

Spanned record support is provided in the ADAACK utility. However, ADAACK assumes that an ISN passed to it is a primary ISN or is the only ISN for a record. If an ISN is the primary ISN for a spanned record, all of the associated segment records of the spanned record are automatically checked in the secondary address converter.

When printing error information about a particular ISN, the ADAACK utility will now indicate whether the problem is with a primary or secondary ISN, if the record is spanned.

ADACDC Utility Changes

Spanned records are not supported by the ADACDC utility at this time. However, when the `IGNORESPANNED` parameter is specified in an ADACDC run, ADACDC processing ignores spanned records, issues a warning message, and continues its processing. A return code of "4" is returned.

ADACMP Utility Changes

- General Changes
- ADACMP COMPRESS Changes

General Changes

Traditionally, the DD/FEHL error data set produced for ADACMP errors has truncated rejected records that exceeded the FEHL physical record length. In Version 8, the rejected records are segmented instead of truncated. Because of this change, the DD/FEHL LRECL setting must be at least 500 bytes.

ADACMP COMPRESS Changes

The following *new* parameters have been added to the ADACMP COMPRESS utility to support the MU/PE extension, spanned records, and LB fields:

- The `DATADEVICE` parameter specifies the Data Storage device type to be used for the segmentation of spanned records. If the SPAN parameter is specified, ADACMP will break long, spanned, compressed records into segments that are just a bit smaller than the Data Storage block size implied by the DATADEVICE parameter.

  If `DATADEVICE` is specified without the SPAN parameter, it is used to derive the maximum compressed record length that will be accepted. Longer compressed records will be considered in error and will be written to the DD/FEHL data set.
- The `HEADER` parameter indicates whether or not the ADACMP compression logic should expect segmented ADACMP record headers in the uncompressed input records. The default is NO. (Contrast this with the HEADER parameter introduced in this release for ADACMP DECOMPRESS.)
- The `LOBDEVICE` parameter identifies the device type that will be used for loading the *LOB file* produced by the COMPRESS function.
- The `LOBVALUES` parameter indicates whether the uncompressed input data may contain long LB values (larger than 253 bytes).
- The `MAXLOGRECLEN` parameter can be used to specify the size, in bytes, of a buffer used by ADACMP to assemble the physical uncompressed segmented records into logical compressed records. This buffer is allocated and used only if HEADER=YES is also specified.
- The `MUPEX` parameter indicates whether the extended MU/PE limits are allowed for the file. If this option is not specified, the maximum number of MU fields and the maximum number of PE fields that can be specified is 191.
- The `MUPECOUNT` parameter specifies the size of the value count field in the input record for the COMPRESS function . Valid values are "1" or "2". If "1" is specified, each value count field preceding the MU or PE values in the input data must be one byte with a value of no more than "191". If "2" is specified, each value count field preceding the MU or PE values in the input data must be two bytes. A value count may exceed 191 only if the `MUPEX` parameter is also specified.
- The `SPAN` parameter allows the record to be spanned after it is compressed if the compressed record exceeds the data storage block size of the device.

The following *changes* have been made to existing ADACMP COMPRESS parameters:

- The report produced by the DEVICE parameter now includes an indication of whether or not the MUPEX parameter has been set for a file.
- The syntax of the FNDEF parameter has been changed, to allow you to specify the number of occurrences of the MU and PE options. You can also specify the NB and LB field options.
- The MAXPE191 parameter is no longer supported for the ADACMP COMPRESS function in Adabas 8. When specified, a warning message is issued and processing continues.
- If USERISN is specified with HEADER=YES, the ISN immediately follows the ADAH header as part of the logical record.

When running ADACMP COMPRESS for a file that includes LB fields, a second sequential output data set (identified by the DDAUSB1 JCL DD control statement) is produced for the *LOB file* containing the LB field values.

ADACMP DECOMPRESS Changes

ADACMP DECOMPRESS processing supports the extended MU and PE limits and spanned records as input. The size of the value count preceding MU or PE values in each decompressed output record depends upon the extended MU and PE support for a file. A two-byte value count is given for files with extended MU and PE support. A one-byte value count is given for files without extended MU and PE support. In addition, the following functionality is supported.

- The decompression of LB fields. A new LOBVALUES parameter has been added for this support.
- A new HEADER parameter has been added to the ADACMP DECOMPRESS utility to indicate whether or not the ADACMP decompression logic should produce the ADACMP segmented record headers (ADAH and ADAC) as part of the decompressed output. The default is NO.
- A new MAXLOGRECLEN parameter can be used to specify the size, in bytes, of a buffer used by ADACMP to assemble logical records that span one or more physical records of uncompressed output data. This buffer is allocated and used only if HEADER=YES is also specified.
- The ISN parameter has been altered so that if it is specified with HEADER=YES, the ISN immediately follows the ADAH header as part of the logical record.

ADADBS Utility Changes

Three new database services have been added to the ADADBS utility to support the extended MU/PE field counts and record spanning:

- The MUPEX function allows you to specify the maximum number of occurrences allowed for MU or PE fields in a file.
- The RECORDSPANNING function activates the use of spanned records in a file.

- The SPANCOUNT function counts the spanned records in a file.

In addition, a new RESETPPT function resets the PPT blocks on the ASSO data set.

ADADCK Utility Changes

The ADADCK utility checks the header of a spanned record for plausibility, as follows:

- The ISNs listed in the header are verified. The header contains the ISN of the primary record in the spanned record chain, the ISN of the previous spanned record in the chain, and the ISN of the next spanned record in the chain.
- The spanned record identifier bits are checked. The header of a spanned record includes bits that indicate whether the record is a primary or secondary spanned record. Only one of these bits can be turned on in the header of any spanned record.

A new MAXPISN parameter has also been introduced that allows you to set the maximum number of primary ISNs that will be checked for a spanned Data Storage file. The default is 1000.

ADAFRM Utility Changes

The ADAFRM utility can now be used to clear multiple PLOG headers from the PLOG, without requiring that you reformat the entire PLOG. To do this, you should specify the NUMBER parameter in conjunction with the FROMRABN parameter and set the SIZE parameter to "1".

In addition, this utility can now handle the increased number of physical Associator and Data Storage extents (99) allowed in Adabas 8.

ADAICK Utility Changes

If you run the ADAICK DSCHECK function, the primary and secondary ISNs are now identified in the output.

ADALOD Utility Changes

The following new parameters are introduced for the ADALOD LOAD and UPDATE functions in support of spanned records and their associated secondary address converter:

- The AC2RABN parameter allows you to specify or update space allocation for the secondary address converter. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.
- The optional MAXISN2 parameter allows you to specify the desired size of the secondary address converter (AC2) in ISNs. The secondary address converter is used to map secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

The following new parameters and parameter values are introduced for the ADALOD LOAD functions in support of large object (LB) fields and their associated *LOB files*:

- The optional LOBFILE parameter allows you to specify the file number of the *LOB file* associated with a *base file*. This parameter is used when loading base files.
- The optional BASEFILE parameter allows you to specify the file number of the *base file* associated with a *LOB file*. This parameter is used when loading LOB files.
- A new file type, LOB, specified on the FILE parameter, allows you to indicate that you are loading an Adabas LOB file with a predefined FDT.

ADAORD Utility Changes

To support spanned records, the following two new parameters have been added to the REORASSO, REORDB, REORFASSO, REORFILE, and STORE functions of ADAORD.

- The AC2RABN parameter allows you to specify the beginning RABN for the file's secondary address converter extent.
- The optional MAXISN2 parameter allows you to specify the desired size of the secondary address converter (AC2) in ISNs.

The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

You can restructure databases and files from Adabas 5.1 - 5.3, 6.1, 6.2, 7.1, 7.2, or 7.4 and store them in Version 8 using ADAORD STORE. However, you cannot store the restructure output of an Adabas 8 database or file in a database running with any prior Adabas version (for example, Adabas 7).

You can restructure databases and files from an Adabas version prior to Adabas 8 and store them in an Adabas 8 database using ADAORD STORE. However, you cannot store the restructured output of an Adabas 8 database or file in a database running with any prior Adabas version (for example, Adabas 7). If you attempt this, the following warning will be generated and ADAORD will end with a CC=4:

```
*** Warning: The input dataset is from V8 and will not be processed
```

ADAREP Utility Changes

The report produced by ADAREP now lists whether the MUPEX and spanned record options have been set for the database.

In the "Contents of Database" section of the report, the following changes have been made:

- The padding factor has been removed from this table to make room for the larger extent values. However, it is still shown in the individual file details also provided in the report and it appears again if LAYOUT=1 is specified.
- When LAYOUT=1 is specified during report creation, the larger extent values, supported by Adabas 8, appear for each file.
- If a file is unable to build at least ten further file extents, ADAREP marks the file with an asterisk (*) on the right.

In the "File Options" section of the report, a "T" indicates that two-byte MU/PE indexes are active for the file and an "S" indicates that use of spanned records has been activated for the file. In addition, the "Contains LOB Fields" column indicates whether the file contains one or more LB fields (an "L" appears if it does) and the "LOB File" column (the last column) indicates whether or not the file is a LOB file (an "L" appears if it is). Note that these two LB field columns are mutually exclusive; only one of them will be marked.

In the "Physical Layout of the Database" section of the report, secondary address converter extents (for spanned records) are identified as "AC2" in the "Table File Type" column.

In the "File Information" section of the report, a new field called "Two Byte MU/PE" indicates whether two-byte MU/PE indexes are active for the file. In the same section, the highest, maximum expected, and minimum secondary ISN are given as well as a new field called "Spanned Rec Supp", which indicates whether or not spanned records are activated for the file. In addition, the "Contain LOB Fields" field indicates whether or not the file contains one or more LB fields and the "LOB File" field indicates whether or not the file is a LOB file.

In the Space Allocation section of the report, secondary address converter extents (for spanned records) are identified as "AC2" in the "List Type" column.

Finally, three new checkpoints may be written by the Adabas nucleus: 75, 76, and 77.

ADASAV Utility Changes

The following new parameters are introduced for the ADASAV RESTONL FMOVE and ADASAV RESTORE FMOVE functions in support of spanned records and their associated secondary address converter:

- The AC2RABN parameter allows you to specify the starting secondary address converter RABN for each file specified by FMOVE.
- The MAXISN2 parameter allows you to specify the new number of secondary ISNs to be allocated for each file specified by FMOVE.

ADASEL Utility Changes

While there are no new parameters, ADASEL has been updated to support the MU/PE extension. In particular, when specifying indexes for MU or PE fields in an ADASEL SELECT IF statement, the indexes can now range from "1" through "65,534". Prior releases of Adabas restricted these indexes to values ranging from "1" through "191".

ADASEL recognizes spanned records in its processing, but it cannot process files containing spanned records.

ADAULD Utility Changes

While there are no new parameters, ADAULD has been updated to support the MU/PE extension and spanned records. In particular, two new statistics listing the number of record *segments* that were read and written during the run are produced by the ADAULD utility.

6 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.

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* in the *Adabas Command Reference Guide*. For information about the ACBX structure and ABDs, read *Adabas Control Block Structures (ACB and ACBX)* and *Adabas Buffer Descriptions (ABDs)* in the *Adabas Command Reference Guide*.

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:

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

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***, elsewhere in this guide.

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:

- 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***, elsewhere in this guide.

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.
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).

ACBX DSECT Name	Description
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)* in the *Adabas Command Reference Guide*.

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)
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)

Offset	ACB DSECT Field Name	ACBX DSECT Field Name
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)
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)* in the *Adabas Command Reference Guide*.

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***, elsewhere in this guide.
- 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:

- 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*.

7 User Exit Changes

This chapter describes the user exit changes for Adabas 8.

User Exit 1

With the introduction of user exit 11, support for user exit 1 will be dropped. However, to ease the migration, a sample user exit, UEX11UX1, is supplied that you can insert in front of your existing user exit 1 to have it invoked as user exit 11. This sample will only work for direct calls made using the ACB direct call interface; it will not work for direct calls made using the Adabas 8 ACBX direct call interface. The exit is still subject to exit 11 constraints, as described in ***User Exit 11***. In particular, changes are allowed only to the file number (CQEFNR), Additions 2 (ACBADD2), Additions 3 (ACBADD3), and user area (ACBUSER) fields. The nucleus will ignore changes in any other ACB field and all other changes to the CQE. Please refer to comments in the sample user exit for more details, including how to link it with an existing user exit 1. Adabas nucleus support for this transition aid will be withdrawn in a future Adabas release.

User Exit 11

This new user exit is given control by Adabas immediately after a command is received by the Adabas nucleus. The command itself has yet to be processed except for the determination of the type of command (simple access, complex access, update).

One of the most common applications of this user exit is to insert a security password or a cipher code into the ACBX.

This user exit functionality largely matches that of the classic user exit 1, except for the fact that edited copies of the CQE and ACBX data structures are used during user exit 11 processing, rather than the actual structures used by user exit 1. Only certain fields in the ACBX may be changed by the exit: ACBXFNR (file number), ACBXADD2 (Additions 2), ACBXADD3 (Additions 3), and ACBXUSER (user area). The nucleus will ignore changes in any other ACBX fields and all changes to the CQE. DSECT EX11PARM maps the user exit 11 parameter list.

Hyperexit Changes

Updated hyperexit logic is supplied in this release that allows for the extended MU/PE fields in hyperdescriptor specifications. The Adabas 8 nucleus detects if an old hyperexit has been provided and returns a response code if it detects an old parameter list.

In addition, Adabas 8 includes a Hyperexit Stub that allows your existing hyperexits to use the Adabas 8 parameter list without change. The Hyperexit Stub is intended as a temporary solution for those customers who do not wish to immediately update their hyperexits to use the new

Adabas 8 parameter areas. For more information about all hyperexit support in Adabas 8, read *Hyperdescriptor Exits 01 - 31* in *Adabas User, Hyperdescriptor, and Collation Descriptor Exits Manual*.

8 Add-On Product Support for Adabas 8

This chapter describes Version 8 plans for the following add-on products.

Caution: The material presented here is subject to change before product release. Software AG reserves the right to change the content of this product and this documentation prior to release and does not guarantee that all functionality mentioned will be implemented. However, this information is sufficiently accurate to allow you to successfully evaluate your upgrade to Version 8.

General Compatibility Information of Client-Based Add-On Products

The client-based add-on products include:

- Adabas Fastpath
- Adabas SAF Security
- Adabas Transaction Manager
- Adabas Vista
- Adabas System Coordinator

Many of the base Adabas 8 enhancements do not have a direct effect upon these add-on products. This means that the Adabas 8 enhancements provide no obstacle to upgrading. For example, these products are not affected by internal Adabas 8 characteristics such as spanned records or internal changes to utilities.

These products will support the new and changed Adabas features in the following ways (where relevant):

Specific Adabas 8 Feature Support	When a specific Adabas 8 feature directly affects current add-on product functionality, the appropriate add-on products are enhanced to support the Adabas 8 feature enhancements.
Adabas 8 Feature Tolerance	When a specific Adabas 8 feature does *not* directly affect current product functionality, the Adabas 8 functionality can be freely used, but the add-on product might not take advantage of it.

This section covers the following topics:

Specific Adabas 8 Feature Support

The following Adabas 8 features are supported in the client-based add-on products:

- The new ACBX calling protocol
- The new FDT layout
- Large object (LB) fields, where relevant
- MU/PE limit increases, where relevant

Adabas 8 Feature Tolerance

All other Adabas 8 features are tolerated by the client-based add-on products, since they have no specific effect upon the functionality of these products.

Required Actions to Support Adabas 8

There are no specific actions needed to be taken in the client-based add-on products to make them support Adabas 8 because detection of the Adabas version level is automatic, where relevant. However, there are some conversion requirements.

Specifically, the client-based add-on products share the same *system configuration file*. The system configuration file is an Adabas file container for all the information needed to enable the software to operate effectively at runtime. Conversion of the system configuration file from previous releases is mandatory, as follows:

- A new file must be used for the Version 8 level
- A conversion utility will be available to copy Version 7.4 file contents and convert them into the new Version 8 file format.
- If the conversion step fails, the new file must be reset to an empty state before starting the conversion again.

Adabas 7.4 Compatibility

The Version 8.1 client-based add-on products can be used with Adabas 7.4 and Adabas 8 databases simultaneously under the following conditions:

- Adabas 8 functionality is *not* used when targeting Adabas 7.4 databases
- Fixes (to be clarified when the client-based add-on products are released) may be necessary to enable Version 8.1 client-based add-ons to operate with Adabas 7.4 and other Version 7.4-compatible Adabas add-ons.

- All of the client-based add-on products must be at the same level. For example, you cannot mix Adabas Fastpath 7.4 functionality with Adabas Vista 8.1 functionality.

Adabas System Coordinator Version 8 Support

Adabas System Coordinator is a client-based add-on product for Adabas. Please read ***General Compatibility Information of Client-Based Add-On Products***, earlier in this section, for general information on migrating Adabas System Coordinator and the other client-based add-on products to Version 8.

The Adabas System Coordinator provides base technology for Adabas Fastpath, Adabas SAF Security, Adabas Transaction Manager, and Adabas Vista. Consequently, some enhancements have been implemented in the Adabas System Coordinator to provide benefit for these other products in a highly integrated way.

This section covers the following topics.

More Focused Client Runtime Configuration

There is increasing need for Adabas client sessions to operate differently within the same job. For example:

- Client "ABC" in CICSXYZ needs special tracing controls to be in use, all other clients do not.
- Transaction "D412" in CICSXYZ must be able to operate with a lower timeout limit than other transactions.
- Stepname "S0010" in job PRODA032 must be excluded from using the Adabas System Coordinator.

This level of runtime control is becoming extremely important. For example, tracing options can be directed at a very few sessions, rather than globally. This can mean overall memory consumption can be kept to a minimum, while at the same time aggressively pursuing a problem by investigating only the sessions to be scrutinized.

Adabas System Coordinator Version 8.1 allows these configuration controls to be prescribed in advance by adding optional override controls to the original base job level controls. With Version 8.1 it is possible to preconfigure overrides as follows:

Job Type	Override Controls
Batch job	1. Stepname 2. LOGIN (e.g. RACF LOGIN userid) 3. Special API
TSO, CMS, TIAM, etc	Special API
COM-PLETE, CICS, IMS, UTM	1. Special API 2. LOGIN 3. Transaction code

As a terminal operator moves from one transaction to another, the runtime behaviors can be altered dynamically according to what is prescribed in the configuration file.

Fully Dynamic Client Runtime Configuration for Experts

In addition to being able to preset the different configurations to be adopted at runtime, you can now dynamically change the runtime controls for your current session. So, you may decide to switch tracing on or off, for example, regardless of what is prescribed in the configuration file.

These truly dynamic changes can be performed in Natural by logging onto one of the product libraries (SYSMP812, SYSMT812, SYSMV812, SYSMW812 or SYSMX812) and entering `CORENV xxx` on the command line, where *xxx* is the product code AFP, AVI, ATM or COR.

Alternate System Configuration File

The system configuration file is now a vital part of the runtime operation. As such, it can become a single point of failure. Adabas System Coordinator 8.1 now allows you to define an alternate configuration file.

Note: It is your responsibility to ensure the two configuration files are identical at all times.

Each session will attempt to use the primary system configuration file. If it is unavailable, the alternate system configuration file will be used, if one is defined. Once a configuration file has been identified for a session, that file will continue to be the primary configuration file for that session until it becomes unavailable, at which point the alternate configuration file will be used. Over time, different sessions may be using different files at the same time until you forcibly cause all sessions to switch over by making one or the other configuration file unavailable for a long period.

If you use an alternate configuration file, then both the primary and alternate configuration files must be available at both Coordinator daemon startup and shutdown. This is necessary because recovery and restart information is placed in the file, and the same information must be placed in both files so they do not get out of step.

Variable Data Container

Your own variable data can now be stored in the configuration file. You can enter this data at the same time as the base or override configuration. This can be used simply for your own documentation purposes, but you can also use a runtime API call to retrieve the data (up to 256 bytes). This means you can easily recognize differences between sessions at runtime for your own purposes according to the configuration being used.

Versioning Feature

The versioning feature allows you to go through a gradual upgrade before fully using Version 8.1. This covers the clients and databases where this software is introduced. In TP systems, a front-end to the ADALNK technology is introduced allowing (for example) ADALNK74 versus ADALNK81 to be used within the same client job. The ADALNK path is chosen according to transaction code (by default). This allows you to convert gradually.

A similar approach is taken for the software in Adabas target databases. For example, you can use Adabas System Coordinator and Adabas Fastpath 7.4 in parallel with Adabas System Coordinator and Adabas Fastpath 8.1. This accomodates clients coming through the 7.4 versus 8.1 paths simultaneously.

Applying Adabas System Coordinator To a Client Job Without Other Client-Based Add-On Products

Adabas System Coordinator provides some general features that are applicable even if the other client-based add-on products are not to be used for a job. For example, the command retry feature may be needed for a job but not Adabas Fastpath. Adabas System Coordinator 8.1 can be activated without needing any of the other products.

Adabas Fastpath Version 8 Support

Adabas Fastpath is a client-based add-on product for Adabas. Please read ***General Compatibility Information of Client-Based Add-On Products***, earlier in this section, for general information on migrating Adabas System Coordinator and the other client-based add-on products to Version 8.

The Adabas System Coordinator provides base technology for Adabas Fastpath, Adabas SAF Security, Adabas Transaction Manager, and Adabas Vista. Consequently, some enhancements have been implemented in the Adabas System Coordinator to provide benefit for these other products in a highly integrated way. For further information, read ***Adabas System Coordinator Support***, elsewhere in this section. Adabas Fastpath is compliant with the **dynamic client runtime configuration** introduced in Adabas System Coordinator 8.1.

This section covers the following topics.

- Automated Buffer Restart Enhancements

Automated Buffer Restart Enhancements

The automated buffer restart feature is enhanced so that you can now control: "Restart every “n” hours but only at “n” o’clock".

This is enhanced to provide some help for sites where the Adabas Fastpath buffer operates 24 hours, 7 days a week, but the commands statistics exceed the present 4G limit, which means the displays lose significance. With this feature, you can make sure a restart occurs prior to the normal time span in which the limit is exceeded. In addition, you can isolate the actual restart time into a quiet period of your day to make sure it does not take place at peak times.

A future release will allow statistics to exceed the 4G limit.

Direct Access Optimization for Secured Files

Adabas Fastpath’s default behavior is to avoid caching for secured files because the assumption is that security is used to differentiate between data used by different sessions. However, some sites do not secure data in this way, and so find that it is satisfactory for their secured files to be cached in this way. This feature, however, should be used with caution.

AFPLOOK Changes

The predictions made by the AFPLOOK sampler have been changed to be more in line with the optimization achieved by the present level of Adabas Fastpath. It will also be possible to preconfigure the AFPLOOK version that is shipped with Adabas without having to use the online administration tool.

Adabas SAF Security Version 8 Support

Adabas SAF Security is a client-based add-on product for Adabas. Please read ***General Compatibility Information of Client-Based Add-On Products***, earlier in this section, for general information on migrating Adabas System Coordinator and the other client-based add-on products to Version 8.

This section covers the following topics.

Variable Resource Renaming

With Adabas SAF Security 8.1, you will be able to use logical names for protected resources, rather than number-based names. This will make it much more flexible to make collections of resources to be gathered together.

Stored Procedure Protection

An option to protect execution of PC calls will be introduced.

Hold-Based Commands

An option to assess hold-based commands (L4, L5, etc.) as update commands rather than access commands will be introduced.

Adabas Security

An option to use the SAF-based GROUP identity as the ADASCR password will be introduced.

Adabas SAF Security Deactivation

In some cases, it is possible that Adabas SAF Security fails to initialize within a database. Some sites wish to be able to configure whether this should force Adabas nucleus termination or not, by database. This feature will be introduced in Adabas SAF Security Version 8.1.

Adabas Transaction Manager Version 8 Support

Adabas Transaction Manager is a client-based add-on product for Adabas. Please read ***General Compatibility Information of Client-Based Add-On Products***, earlier in this section, for general information on migrating Adabas System Coordinator and the other client-based add-on products to Version 8.

The Adabas System Coordinator provides base technology for Adabas Fastpath, Adabas SAF Security, Adabas Transaction Manager, and Adabas Vista. Consequently, some enhancements have been implemented in the Adabas System Coordinator to provide benefit for these other products in a highly integrated way. For further information, read ***Adabas System Coordinator Support***, elsewhere in this section. Adabas Transaction Manager is compliant with the **dynamic client runtime configuration** introduced in Adabas System Coordinator 8.1.

This section covers the following topics.

Nonactivity Timeout Handling

This will now be handled by the equivalent general Adabas System Coordinator facility rather than by any specific facility to Adabas Transaction Manager.

Adabas Vista Version 8 Support

Adabas Vista is a client-based add-on product for Adabas. Please read ***General Compatibility Information of Client-Based Add-On Products***, earlier in this section, for general information on migrating Adabas System Coordinator and the other client-based add-on products to Version 8.

The Adabas System Coordinator provides base technology for Adabas Fastpath, Adabas SAF Security, Adabas Transaction Manager, and Adabas Vista. Consequently, some enhancements have been implemented in the Adabas System Coordinator to provide benefit for these other products in a highly integrated way. For further information, read ***Adabas System Coordinator Support***, elsewhere in this section. Adabas Vista is compliant with the **dynamic client runtime configuration** introduced in Adabas System Coordinator 8.1.

This section covers the following topics.

Generations Replace Publish and Draft Copies

Previous versions of Adabas Vista allowed individual file rules to have published and draft copies. This allowed the active runtime rules to be left in place while changes were being made. However, this meant that many draft rules could not be simultaneously activated into the operational runtime.

Organizing the rules into *generations* means that a complete new set of rules can be prepared in the background and then simultaneously activated. This is a far better control mechanism for introducing change to the runtime operation.

Multiple generations of rules can be constructed at the same time, but only one generation can be named as the *active* one. At any time, you can define a different generation as the active generation. This automatically places the originally active generation in inactive status.

The base rules for a generation cannot be directly modified. You must define a *delta* modification to a generation to make changes. You can have multiple deltas in parallel to a generation. Once you are happy with a delta modification you can *apply* it to update the base generation.

File Translation Pages

With Adabas Vista 8.1, you can now define multiple pages of translation rules. The name of each page is chosen by your data base administrator. Each page contains one or more file translation file rules. Each job that is defined to use Adabas Vista can have up to eight pages named for use at runtime for that job. Adabas Vista merges all these pages into one set of translation rules for runtime use.

Some rules can be defined as being mandatory, or have a numeric priority setting. During the merge the mandatory and priority settings are honored.

In addition, it is possible to set up a mandatory page itself, but the translation rules contained within that page do not themselves have to be mandatory. The mandatory page is always used, when it exists. The mandatory page does not have to be named on Adabas Vista job controls; this is how previous version compatibility is most likely to be maintained. The Adabas Vista 8.1 conversion utility will arrange to convert your existing 7.4 rules accordingly

Source Name Removed

The source name field is removed from translation rules. The Adabas Vista 8.1 conversion utility will discard these translation rules automatically.

Target Category Removed From Partitioned Files

The target category is no longer a part of the partitioned file definition. It is only used for translation rules.

Index

A

U

Z