

COMPRESS: Create an Adabas File

```

ADACMP COMPRESS [CODE = cipher-code]
                 [DEVICE = { device-type-list | ADARUN-device }]
                 [FACODE = file-alpha-EBCDIC-key]
                 [FDT = file-number]
                 [FILE = { file-number | 0 }]
                 [FWCODE = file-wide-key]
                 [FUWCODE = { wide-key | UWCODE-definition }]
                 [FORMAT = format]
                 [MAXPE191]
                 [NOUSERABEND]
                 [NUMREC = number-of-records]
                 [PASSWORD = "password" ]
                 [RECFM = { E | FB | V | VB | U }
                  [,LRECL = record-length ]
                 [ { USERISN | MINISN = { start-isn | 1 } } ]
                 [UACODE = userdata-alpha-key]
                 [UWCODE = { userdata-wide-key | FWCODE-definition }]
                 [UARC = { userdata-architecture-key | 2 } ]

                 FNDEF = ' field-definition '
                 [COLDE = ' collation-descriptor-definition ' ]
                 [HYPDE = ' hyperdescriptor-definition ' ]
                 [PHONDE = ' phoneticdescriptor-definition ' ]
                 [SUBDE = ' subdescriptor-definition ' ]
                 [SUBFN = ' subfield-definition ' ]
                 [SUPDE = ' superdescriptor-definition ' ]
                 [SUPFN = ' superfield-definition ' ]

```

} (field definition statements)

This chapter covers the following topics:

- Optional Parameters and Subparameters
- Essential Data Definition Syntax
- Optional Field Definition Statements
- ADACMP COMPRESS Examples

Optional Parameters and Subparameters

CODE: Cipher Code

If the data is to be loaded into the database in ciphered form, the cipher code must be specified with this parameter. See the *Adabas Security* documentation for additional information on the use of ciphering.

DEVICE: Device Type

ADACMP calculates and displays a report of this run's space requirements for each specified device type. If DEVICE= is not specified, the default is the ADARUN device type.

FACODE: Alphanumeric Field Encoding

FACODE must be specified if you want to define UES file encoding for alphanumeric fields in the file. The alphanumeric encoding must belong to the EBCDIC encoding family; that is, the space character is X'40'.

FDT: Use Existing Adabas Field Definition Table

An existing Adabas FDT is to be used. The FDT may be that of an existing file or a file that has been deleted with the KEEPFDI option of the ADADBS utility.

The input data must be consistent with the structure as defined in the specified FDT, unless the FORMAT parameter is used. When the FDT defines multiple-value fields or periodic groups, length values must be defined or already included in the FDT; refer to the sections Multiple-Value Field Count and Periodic Group Field Count.

If the FDT parameter is used, any field definitions specified will be ignored.

FILE: File Number

If the FDT contains a hyperdescriptor, this parameter must be specified. The specified file number becomes input for the related hyperexit. For more information about hyperexits, refer to the *Adabas DBA Referenced* documentation.

User exit 6 is always supplied with this file number. If FILE is not specified, a value of zero is assumed.

FORMAT: Input Record Format Definition

Use this parameter to provide a format definition that indicates the location, format, and length of fields in the input record. The format provided must follow the rules for format buffer entries for update commands as described in the *Adabas Command Reference* documentation.

Conversion rules are those described for Adabas update commands in the *Adabas Command Reference* documentation. For conversion of SQL null (NC option) field values, see NC: SQL Null Value Option. If a field is omitted in the FORMAT parameter, that field is assigned no value.

If the FORMAT parameter is omitted, the input record is processed in the order of the field definition statements provided or, if the FDT parameter is used, according to an existing Adabas field definition table.

FUWCODE: Wide-Character Field Default User Encoding

FUWCODE defines the default user encoding for wide-character fields for the file when loaded in the database. If this parameter is omitted, the encoding is taken from the UWCODE definition of the database.

FWCODE: Wide-Character Field Encoding

If fields with format W (wide-character) exist in the compressed file, you *must* specify FWCODE to define the file encoding for them.

FWCODE also determines the maximum byte length of the wide-character field.

LRECL: Input Record Length (VSE Only)

If RECFM=F or RECFM=FB is specified, this parameter must also be specified to provide the record length (in bytes) of the input data; otherwise, do not specify LRECL.

For z/OS or OS/390, the record length is taken from the input dataset label or DD statement.

For BS2000, the record length is taken from the catalog entry or /FILE statement.

MAXPE191: Enable Periodic Group Count Up to 191

Periodic groups can have up to 191 occurrences. The limit of 191 is allowed by the nucleus without further specification; however, to compress records with more than 99 periodic group occurrences, the parameter MAXPE191 must be specified.

Note:

This option is not compatible with Adabas 5.2 releases; therefore, backward conversion to Adabas 5.2 is not possible once records with more than 99 PE group occurrences have been loaded.

MINISN: Starting ISN

For automatic ISN assignment, MINISN defines the lowest ISN to be used. If MINISN is not specified, the default is 1. If USERISN is specified, MINISN cannot be specified.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUMREC: Number of Records to Be Processed

Specifies the number of input records to be processed. If this parameter is omitted, all input records contained on the input dataset are processed.

Software AG recommends using this parameter for the initial ADACMP execution if a large number of records are contained on the input dataset. This avoids unneeded processing of all records when a field definition error or invalid input data results in a large number of rejected records. This parameter is also useful for creating small files for test purposes.

Setting NUMREC to zero (0) prevents the input dataset from being opened.

PASSWORD: Password for FDT File

If the FDT parameter is specified and the file is password-protected, this parameter must be used to provide a valid password for that file.

RECFM: Input Record Format (VSE Only)

You *must* specify the input record format with this parameter as follows:

F	fixed length, unblocked (requires that you also specify the LRECL parameter)
FB	fixed length, blocked (requires that you also specify the LRECL parameter)
V	variable length, unblocked
VB	variable length, blocked
U	undefined

Under z/OS or OS/390, the record format is taken from the input dataset label or DD statement.

Under BS2000, the record format is taken from the catalog entry or FILE statement.

UACODE: User Encoding for Input Alphanumeric Fields

UACODE defines the user encoding of the sequential input of alphanumeric fields. If you specify UACODE, you *must* also specify FACODE.

UARC: Architecture for Input Uncompressed User Data

The UARC parameter specifies the architecture of the sequential input of the uncompressed user data. The "userdata-architecture-key" is an integer which is of the sum of the following numbers:

byte order	b=0	high-order byte first
	b=1	low-order byte first
encoding family	e=0	ASCII encoding family
	e=2	EBCDIC encoding family (default)
floating-point format	f=0	IBM370 floating-point format
	f=4	VAX floating-point format
	f=8	IEEE floating-point format

The default is $ARC = b + e + f = 2$; that is, high-order byte first; EBCDIC encoding family; and IBM370 floating-point format (b=0; e=2; f=0).

User data from an Intel386 PC provides the example: b=1; e=0; f=8; or $ARC=9$.

USERISN: User ISN Assignment

The ISN for each record is to be provided by the user. If this parameter is omitted, the ISN for each record is assigned by Adabas.

If USERISN is specified, the user must provide the ISN to be assigned to each record as a four-byte binary number immediately preceding each data record. If the MINISN parameter is specified, USERISN cannot be specified.

The format for fixed or undefined length input records with user-defined ISNs is:

userisn/data

The format for variable-length input records with user-defined ISNs is

length/xx/userisn/data

where

<i>length</i>	is a two-byte binary physical record length (length of record data, plus 8 bytes).
<i>xx</i>	is a two-byte field containing binary zeros.
<i>userisn</i>	is a four-byte binary ISN to be assigned to the record.
<i>data</i>	is input record data.

ISNs may be assigned in any order, must be unique (for the file), and must not exceed the MAXISN setting specified for the file (see the ADALOD utility).

ADACMP does not check for unique ISNs or for ISNs that exceed MAXISN. These checks are performed by the ADALOD utility.

UWCODE: User Encoding for Input Wide-Character Fields

UWCODE defines the user encoding of the sequential input of wide-character fields. If you specify UWCODE, you *must* also specify FWCODE.

For user input, all wide-character fields are encoded in the same code page. It is not possible to select different encodings for different fields in the same ADACMP run.

Essential Data Definition Syntax

The field definitions provided as input to ADACMP are used to

- provide the length and format of each field contained in the input record. This enables ADACMP to determine the correct field length and format during editing and compression;
- create the field definition table (FDT) for the file. This table is used by Adabas during the execution of Adabas commands to determine the logical structure and characteristics of any given field (or group) in the file.

The following syntax must be followed when entering field definitions. A minimum of one and a maximum of 926 definitions may be specified.

Field	<code>FNDEF=' level , name [, length , format] [{ , option } ...]'</code>
Group	<code>FNDEF=' level , name [, PE [(n)]]'</code>
Collation descriptor	<code>COLDE=' number , name [, UQ [, XI]] = parentfield'</code>
Hyperdescriptor	<code>HYPDE=' number , name , length , format [{ , option } ...] = { parentfield } , ...'</code>
Phonetic descriptor	<code>PHONDE=' name (field)'</code>
Subdescriptor	<code>SUBDE=' name [, UQ [, XI]] = parentfield (begin , end)'</code>
Subfield	<code>SUBFN=' name = parentfield (begin , end)'</code>
Superdescriptor	<code>SUPDE=' name [, UQ [, XI]] = parentfield (begin , end) , ...'</code>
Superfield	<code>SUPFN=' name = parentfield (begin , end) [, parentfield (begin , end)] ...'</code>

User comments may be entered to the right of each definition. At least one blank must be present between a definition and any user comments.

FNDEF: Field/Group Definition

This parameter is used to specify an Adabas field (data) definition. The syntax used in constructing field definition entries is

```
FNDEF = 'level, name [ , length, format ] [ { , option } ... ]'
```

Level number and name are required. Any number of spaces may be inserted between definition entries.

level

The level number is a one- or two-digit number in the range 01-07 (the leading zero is optional) used in conjunction with field grouping. Fields assigned a level number of 02 or greater are considered to be a part of the immediately preceding group which has been assigned a lower level number.

The definition of a group enables reference to a series of fields (may also be only 1 field) by using the group name. This provides a convenient and efficient method of referencing a series of consecutive fields.

Level numbers 01-06 may be used to define a group. A group may consist of other groups. When assigning the level numbers for nested groups, no level numbers may be skipped.

FNDEF='01,GA'	group	
FNDEF='02,A1,...'		elementary or multiple-value field
FNDEF='02,A2,...'		elementary or multiple-value field
FNDEF='01,GB'	group	
FNDEF='02,B1,...'		elementary or multiple-value field
FNDEF='02,GC'	group (nested)	
FNDEF='03,C1,...'		elementary or multiple-value field
FNDEF='03,C2,...'		elementary or multiple-value field

Fields A1 and A2 are in group GA. Field B1 and group GC (consisting of fields C1 and C2) are in group GB.

name

The name to be assigned to the field (or group).

Names must be unique within a file. The name must be two characters long: the first character must be alphabetic; the second character can be either alphabetic or numeric. No special characters are permitted.

The values E0-E9 are reserved as edit masks and may not be used.

Valid Names	Invalid Names
AA	A (not two characters)
B4	E3 (edit mask)
S3	F* (special character)
WM	6M (first character not alphabetic)

length

The length of the field (expressed in bytes). The length value is used to

- indicate to ADACMP the length of the field as it appears in each input record; and
- define the standard (default) length to be used by Adabas during command processing.

The standard length specified is entered in the FDT and is used when the field is read/updated unless the user specifies a length override.

The maximum field lengths that may be specified depend on the "format" value:

Format	Maximum Length
Alphanumeric (A)	253 bytes
Binary (B)	126 bytes
Fixed Point (F)	4 bytes (always exactly 2 or 4 bytes)
Floating Point (G)	8 bytes (always exactly 4 or 8 bytes)
Packed Decimal (P)	15 bytes
Unpacked Decimal (U)	29 bytes
Wide-character (W)	253 bytes*

* Depending on the FWCODE attribute value, the maximum byte length of the W field may be less than 253. For example, if the default value of FWCODE is used (that is, Unicode), the maximum length is 252 (2 bytes per character).

Standard length may not be specified with a group name.

Standard length does not limit the size of any given field value unless the FI option is used - see FI: Fixed Storage . A read or update command may override the standard field length, up to the maximum length permitted for that format.

If standard length is zero for a field, the field is assumed to be a variable-length field. Variable-length fields have no standard (default) length. A length override for fixed-point (F) fields can specify a length of two or four bytes only; for floating-point (G) fields, the override can specify four or eight bytes only.

If a variable-length field is referenced without a length override during an Adabas command, the value in the field will be returned preceded by a one-byte binary length field (including the length byte itself). This length value must be specified when the field is updated, and also in the input records that are to be processed by ADACMP. If the field is defined with the long alpha (LA) option, the value is preceded by a two-byte binary length field (including the two length bytes).

format

The standard format of the field (expressed as a one-character code):

A	Alphanumeric (left-justified)
B	Binary (right-justified, unsigned/positive)
F	Fixed point (right-justified, signed, two's complement notation)
G	Floating point (normalized form, signed)
P	Packed decimal (right-justified, signed)
U	Unpacked decimal (right-justified, signed)
W	Wide character (left-justified)

The standard format is used to

- indicate to ADACMP the format of the field as it appears in each input record; and
- define the standard (default) format to be used by Adabas during command processing. The standard format specified is entered in the FDT and is used when the field is read/updated unless the user specifies a format override.

Standard format must be specified for a field. It may not be specified with a group name. When the group is read (written), the fields within the group are always returned (must be provided) according to the standard format of each individual field. The format specified determines the type of compression to be performed on the field.

A fixed-point field is either two or four bytes long. A positive value is in normal form, and a negative value in two's complement form.

A field defined with floating-point format may be either four bytes (single precision) or eight bytes (double precision) long. Conversion of a value of a field defined as floating point to another format is supported.

If a binary field is to be defined as a descriptor, and the field may contain both positive and negative numbers, "F" format should be used instead of "B" format because "B" format assumes that all values are unsigned (positive).

Like an alphanumeric field, a wide-character field may be a standard length in bytes defined in the FDT, or variable length. Any non-variable format override for a wide-character field must be compatible with the user encoding; for example, a user encoding in Unicode requires an *even* length. Format conversion from numbers (U, P, B, F, G) to wide-character format is not allowed.

Data Definition Field/Group Options

Options are specified by the two-character codes. These codes may be specified in any order, separated by a comma.

Code	Option
DE (Descriptor)	Field is to be a descriptor (key).
FI (Fixed Storage)	Field is to have a fixed storage length; values are stored without an internal length byte, are not compressed, and cannot be longer than the defined field length.
LA (Long Alpha Option)	This A or W format variable-length field may contain a value up to 16,381 bytes long.
MU (Multiple-Value Field)	Field may contain up to 191 values in a single record.
NC (SQL Null Value Option)	Field may contain a null value that satisfies the SQL interpretation of a field having no value; that is, the field's value is not defined (not counted).
NU (Null Value Suppression)	Null values occurring in the field are to be suppressed.
NV (No Conversion)	This A or W format field is to be processed in the record buffer without being converted.
PE (Periodic Group)	Group field is to be followed by a periodic group definition that may occur up to 191 times in a given record.
NN (SQL Not Null Option)	Field defined with NC option must always have a value defined; it cannot contain an SQL null (not null).
UQ (Unique Descriptor)	Field is to be a unique descriptor; that is, for each record in the file, the descriptor must have a different value.
XI (Exclude Instance Number)	For this field, the index (occurrence) number is excluded from the UQ option set for a PE.

DE: Descriptor

DE indicates that the field is to be a descriptor (key). Entries will be made in the Associator inverted list for the field, enabling the field to be used in a search expression, as a sort key in a FIND command, to control logical sequential reading, or as the basis for file coupling.

The descriptor option should be used judiciously, particularly if the file is large and the field that is being considered as a descriptor is updated frequently.

Although the definition of a descriptor field is independent of the record structure, note that if a descriptor field is not ordered first in a record and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to reorder the field first for each record of the file.

FI: Fixed Storage

FI indicates that the field is to have a fixed storage length. Values in the field are stored without an internal length byte, are not compressed, and cannot be longer than the defined field length.

The FI option is recommended for fields with a length of one or two bytes that have a low probability of containing a null value (personnel number, gender, etc.) and for fields containing values that cannot be compressed.

The FI option is not recommended for multiple-value fields, or for fields within a periodic group. Any null values for such fields are not suppressed (or compressed), which can waste disk storage space and increase processing time.

The FI option *cannot* be specified for

- U-format fields;
- NC, NN, or NU option fields;
- variable-length fields defined with a length of zero (0) in the FNDEF statement;
- a descriptor within a periodic (PE) group.

A field defined with the FI option *cannot* be updated with a value that exceeds the standard length of the field.

Example of FI usage:

	Definition	User Data	Internal Representation
Without FI Option	FNDEF='01,AA,3,P'	33104C 00003C	0433104F (4 bytes) 023F (2 bytes)
With FI Option	FNDEF='01,AA,3,P,FI'	33104C 00003C	33104F (3 bytes) 00003F (3 bytes)

LA: Long Alpha Option

The LA (long alphanumeric) option can be specified for variable-length alphanumeric and wide format fields; i.e., A or W format fields having a length of zero in the field definition (FNDEF). With the LA option, such a field can contain a value up to 16,381 bytes long.

An alpha or wide field with the LA option is compressed in the same way as an alpha or wide field without the option. The maximum length that a field with LA option can actually have is restricted by the block size where the compressed record is stored.

When a field with LA option is updated or read, its value is either specified or returned in the record buffer, preceded by a two-byte length value that is inclusive (field length, plus two).

A field with LA option

- can also have the NU, NC/NN, NV, or MU option;

- can be a member of a PE group;
- cannot have the FI option;
- cannot be a descriptor field;
- cannot be a parent of a sub-/superfield, sub-/superdescriptor, hyperdescriptor, or phonetic descriptor; and
- cannot be specified in the search buffer, or response code 61 occurs.

For more information, see the Adabas Command Reference Documentaton section *Specifying a Field with LA (Long Alpha) Option* in the section 2 discussion of the record buffer.

Example of LA usage:

	Definition	User Data	Internal Representation
Without LA Option	FNDEF='01,BA,0,A'	X'06',C'HELLO' --	X'06C8C5D3D3D6' (1-byte length) --
With LA Option	FNDEF='01,BA,0,A,LA'	X'0007',C'HELLO' X'07D2',C' ... (2000 data bytes) ...'	X'06C8C5D3D3D6' (1-byte length) X'87D2 ... (2000 data bytes) ... '

MU: Multiple-Value Field

MU indicates that the field may contain more than one value in a single record. The actual number of values present in each record may vary from 0 to 191, although at least one value (even if null) must be present in each record input to ADACMP.

The values are stored according to the other options specified for the field. The first value is preceded by a count field that indicates the number of values currently present for the field. The number of values that are stored is equal to the number of values provided in the ADACMP input record, plus any values added during later updating of the field, less any values suppressed (this applies only if the field is defined with the NU option).

If the number of values contained in each record input to ADACMP is constant, the number can be specified in the MU definition statement in the form MU(n), where "n" equals the number of values present in each input record. For example:

```
FNDEF= ' 01, AA, 5, A, MU( 3 ) '
```

indicates that three values of the multiple-value field AA are present in each input record. Specifying a value of zero (0) indicates that no values are present for the multiple-value field in the input record.

If the number of values is not constant for all input records, a one-byte binary count field must precede the first value in each input record to indicate the number of values present in that record (see also the section Input Data Requirements).

If the FDT is provided (see the FDT parameter description in the *COMPRESS Optional Parameters and Subparameters* section), the field count must be contained as a one-byte binary value in each input record.

If the input records were created using the DECOMPRESS function, all required count fields are already contained in the input record. In this case, the count must not be specified in the field definition statement.

All values provided during input or updating will be compressed (unless the FI option has also been specified). Care should be taken when using the FI and MU options together since a large amount of disk storage may be wasted if a large number of compressible values are present.

If the NU option is specified with the MU option, null values are both logically and physically suppressed. The positional relationship of all values (including null values) is maintained in MU occurrences, unless the occurrences are defined with the NU option. If a large number of null values are present in an MU field group, the NU option can reduce the disk storage requirements for the field but should not be used if the relative positions of the values must be maintained.

The NC (or NC/NN) option *cannot* be specified for an MU field.

Example of MU usage with NU:

```
FNDEF=' 01,AA,5,A,MU,NU'
```

The original content where "L" is the length of the "value" is

- after file loading:

3	L value A	L value B	L value C
count	AA1	AA2	AA3

- after update of value B to null value:

2	L value A	L value C
count	AA1	AA2

Example of MU usage without NU:

```
FNDEF=' 01,AA,5,A,MU'
```

The original content where "L" is the length of the "value" is

- after file loading:

3	L value A	L value B	L value C
count	AA1	AA2	AA3

- after update of value B to null value:

3	L value A	L value B	L value C
count	AA1	AA2	AA3

NU: Null Value Suppression

NU suppresses null values occurring in the field.

Normal compression (NU or FI not specified) represents a null value with two bytes (the first for the value length, and the second for the value itself, in this case a null). Null value suppression represents an empty field with a one-byte "empty field" indicator. The null value itself is not stored.

A series of consecutive fields containing null values and specifying the NU option is represented by a one-byte "empty field" (binary 11nnnnn) indicator, where "nnnnn" is the number of the fields' successive bytes containing null values, up to a total of 63. For this reason, fields defined with the NU option should be grouped together whenever possible.

If the NU option is specified for a descriptor, any null values for the descriptor are not stored in the inverted list. Therefore, a find command in which this descriptor is used and for which a null value is used as the search value will always result in no records selected, even though there may be records in Data Storage that contain a null value for the descriptor. If a descriptor defined with the NU option is used to control a logical sequence in a read logical sequence (L3/L6) command, those records that contain a null value for the descriptor will not be read.

Descriptors to be used as a basis for file coupling and for which a large number of null values exist should be specified with the NU option to reduce the total size of the coupling lists.

The NU option cannot be specified for fields defined with the combined NC/NN options or with the FI option.

Example of NU usage:

	Definition	User Data	Internal Representation
Normal Compression	FNDEF='01,AA,2,B'	0000	0200 (2 bytes)
With FI Option	FNDEF='01,AA,2,B,FI'	0000	0000 (2 bytes)
With NU Option	FNDEF='01,AA,2,B,NU'	0000	C1 (1 byte)*

* C1 indicates 1 empty field.

NV: No Conversion

The "do not convert" option for alphanumeric (A) or wide-character (W) format fields specifies that the field is to be processed in the record buffer without being converted.

Fields with the NV option are not converted to or from the user: the field has the characteristics of the file encoding; that is, the default blank

- for A fields, is always the EBCDIC blank (X'40'); and
- for W fields, is always the blank in the file encoding for W format.

The NV option is used for fields containing data that cannot be converted meaningfully or should not be converted because the application expects the data exactly as it is stored.

The field length for NV fields is byte-swapped if the user architecture is byte-swapped.

For NV fields, "A" format cannot be converted to "W" format and vice versa.

PE: Periodic Group

PE indicates that a periodic group is to be defined. A periodic group

- may comprise one or more fields. A maximum of 254 elementary fields may be specified. Descriptors and/or multiple value fields and other groups may be specified, but a periodic group may not contain another periodic group.
- may occur from 0 to 99 (or 191, if the ADACMP MAXPE191 parameter is specified) times within a given record, although at least one occurrence (even if it contains all null values) must be present in each ADACMP input record.
- must be defined at the 01 level. All fields in the periodic group must immediately follow and must be defined at level 02 or higher (in increments of 1 to a maximum of 7). The next 01 level definition indicates the end of the current periodic group.
- may only be specified with a group name. Length and format parameters may not be specified with the group name.

Following are two examples of period group definition:

Periodic Group "GA":

```
FNDEF=' 01,GA,PE'
FNDEF=' 02,A1,6,A,NU'
FNDEF=' 02,A2,2,B,NU'
FNDEF=' 02,A3,4,P,NU'
```

Periodic Group "GB":

```
FNDEF=' 01,GB,PE(3)'
FNDEF=' 02,B1,4,A,DE,NU'
FNDEF=' 02,B2,5,A,MU(2),NU'
FNDEF=' 02,B3'
FNDEF=' 03,B4,20,A,NU'
FNDEF=' 03,B5,7,U,NU'
```

UQ: Unique Descriptor

UQ indicates that the field is to be a unique descriptor. A unique descriptor must contain a different value for each record in the file. In FNDEF statements, the UQ option can only be specified if the DE option is also specified. The UQ option can also be used in SUBDE, SUPDE, and HYPDE statements.

The UQ option *must* be specified if the field is to be used as an ADAM descriptor (see the ADAMER utility).

ADACMP does not check for unique values; this is done by the ADALOD utility, or by the ADAINV utility when executing the INVERT function. If a non-unique value is detected during file loading, ADALOD terminates with an error message.

Because ADAINV and ADALOD must execute separately for each file in an expanded file chain, they cannot check for uniqueness across the chain.

However, Adabas does check the value of unique descriptors across an expanded file chain. If the value being added (N1/N2) or updated (A1) is not unique across all files within the chain, response code 198 is returned.

XI: Exclude Instance Number

By default, the occurrence number of fields within periodic groups (PE) defined as unique descriptors (UQ) is included as part of the descriptor value. This means that the same field value can occur in different periodic group occurrences in different records.

The XI option is used to exclude the occurrence number from the descriptor value for the purpose of determining the value's uniqueness. If the XI option is set, any field value can occur at most once over all occurrences of the PE field in all records.

Representing SQL Null Values

Adabas includes two data definition options, NC and NN, to provide SQL-compatible null representation for Software AG's mainframe Adabas SQL Gateway (ACE) and other Structured Query Language (SQL) database query languages.

The NC and NN options *cannot* be applied to fields defined

- with Adabas null suppression (NU)
- with fixed-point data type (FI)
- with multiple-values (MU)
- within a periodic group (PE)
- as group fields

In addition, the NN option can only be specified for a field that specifies the NC option.

A parent field for sub-/superfields or sub-/superdescriptors can specify the NC option. However, parent fields for a single superfield or descriptor cannot use a mix of NU and NC fields. If any parent field is NC, no other parent field can be an NU field, and vice versa.

Examples:

A correct ADACMP COMPRESS FNDEF statement for defining the field AA and assigning the NC and NN option:

```
ADACMP FNDEF='01,AA,4,A,NN,NC,DE'
```

Incorrect uses of the NC/NN option that would result in an ADACMP utility ERROR-127:

Incorrect Example	Reason
ADACMP FNDEF='01,AA,4,A,NC,NU'	NU and NC options are not compatible
ADACMP FNDEF='01,AB,4,A,NC,FI'	NC and FI options are not compatible
ADACMP FNDEF='01,PG,PE' ADACMP FNDEF='02,P1,4,A,NC'	NC option within a PE group is not allowed

NC: SQL Null Value Option

Without the NC (not counted) option, a null value is either zero or blank depending on the field's format.

With the NC option, zeros or blanks specified in the record buffer are interpreted according to the "null indicator" value: either as true zeros or blanks (that is, as "significant" nulls) or as undefined values (that is, as true SQL or "insignificant" nulls).

If the field defined with the NC option has no value specified in the record buffer, the field value is always treated as an SQL null.

When interpreted as a true SQL null, the null value satisfies the SQL interpretation of a field having no value. This means that no field value has been entered; that is, the field's value is not defined.

The null indicator value is thus responsible for the internal Adabas representation of the null. For more information, see the following section Null Indicator Value and the section *Search Buffer Syntax* in the *Adabas Command Reference* documentation.

The following rules apply when compressing or decompressing records containing NC fields:

1. If the FORMAT parameter is specified, ADACMP behaves in the same way the nucleus does for update-type commands. See the *Adabas Command Reference* documentation.
2. If the FORMAT parameter is *not* specified

- for *compression*

Only the value of the NC field is placed in the input record; the two null value indicator bytes must be omitted. The value is compressed as if the null value indicator bytes were set to zero. It is not possible to assign a null value to an NC field using this method.

Example:

Field Definition Table (FDT) definition	<code>FNDEF= ' 01 , AA , 4 , A , NC '</code>
Input record contents:	MIKE

- for *decompression*

If the value of an NC field is *not significant*, the record is written to DDFEHL (or FEHL) with response code 55.

If the value of an NC field is *significant*, the value is decompressed as usual. There are no null indicator bytes.

Example:

Field Definition Table (FDT) definition	<code>FNDEF= ' 01 , AA , 4 , A , NC '</code>
Output record contents	MIKE

Null Indicator Value

The null indicator value is always two bytes long and has fixed-point format, regardless of the data format. It is specified in the record buffer when a field value is added or changed; it is returned in the record buffer when the field value is read.

For an update (Ax) or add (Nx) command, the null indicator value must be set in the record buffer position that corresponds to the field's designation in the format buffer. The setting must be one of the following:

Hex Value	Indicates that . . .
FFFF	the field's value is set to "undefined", an insignificant null; the differences between no value, binary zeros, or blanks for the field in the record buffer are ignored; all are interpreted equally as "no value".
0000	no value, binary zeros, or blanks for the field in the record buffer are interpreted as significant null values.

For a read (Lx) or find with read (Sx with format buffer entry) command, your program must examine the null indicator value (if any) returned in the record buffer position corresponding to the field's position in the format buffer. The null indicator value is one of the following values, indicating the meaning of the actual value that the selected field contains:

Hex Value	Indicates that . . .
FFFF	a zero or blank in the field is not significant.
0000	a zero or blank in the field is a significant value; that is, a true zero or blank.
xxxx	the field is truncated. The null indicator value contains the length (xxxx) of the entire value as stored in the database record.

Example:

The field definition of a null represented in a two-byte Adabas binary field AA defined with the NC option is

01,AA,2,B,NC

	Null Indicator Value (Record Buffer)		Adabas Internal Representation
non-zero value	0 (binary value is significant)	0005	0205
blank	0 (binary null is significant)	0000 (zero)	0200
null	FFFF (binary null is not significant)	(not relevant)	C1

NN: SQL Not Null Option

The NN ("not null" or "null value not allowed") option may only be specified when the NC option is also specified for a data field. The NN option indicates that an NC field must always have a value (including zero or blank) defined; it cannot contain "no value".

The NN option ensures that the field will not be left undefined when a record is added or updated; a significant value must always be set in the field. Otherwise, Adabas returns a response code 52.

The following example shows how an insignificant null would be handled in a two-byte Adabas alphanumeric field AA when defined with and without the NN option:

Example:

An insignificant null handled in a two-byte Adabas alphanumeric field AA when defined with and without the NN option is as following:

Option	Field Definition	Null Indicator Value	Adabas Internal Representation
With NN	01,AA,2,A,NC,NN	FFFF (insignificant null)	none; response code 52 occurs
Without NN	01,AA,2,A,NC	FFFF (insignificant null)	C1

Optional Field Definition Statements

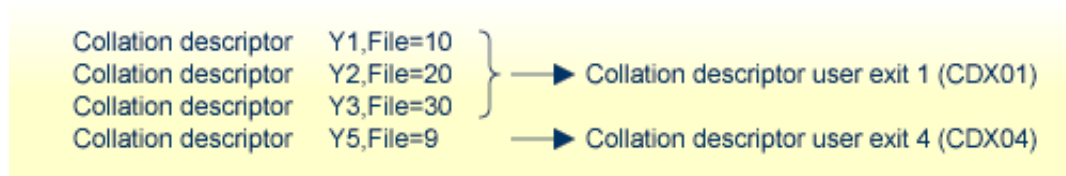
- COLDE: Collation Descriptor Definition
- HYPDE: Hyperdescriptor Definition
- PHONDE: Phonetic Descriptor
- SUBDE: Subdescriptor Definition
- SUBFN: Subfield Definition
- SUPDE: Superdescriptor Definition
- SUPFN: Superfield Definition

COLDE: Collation Descriptor Definition

The collation descriptor option enables descriptor values to be sorted (collated) based on a user-supplied algorithm.

The values are based on algorithms coded in special collation descriptor user exits (CDX01 through CDX08). Each collation descriptor must be assigned to a user exit, and a single user exit may handle multiple collation descriptors.

Example:



The Collation Exit functions are called on the following events:

INITIALIZE function

- nucleus session start
- utility initialization when collation exits have been defined (ADARUN parameters)

ENCODE function

- update/insert/delete of the parent's value (Nucleus)
- Search specifying the collation descriptor with the search value (Nucleus)
- compression of a record (ADACMP)

DECODE function

- Read Index (L9) by Collation DE, only if the exit supports the DECODE function (Nucleus)

Input parameters supplied to the user exit are described in the *Adabas DBA Reference* documentation, section *User Exits*. They include

- address and length of input string
- address and size of output area
- address of fullword for the returned output string length

The user exit sets the length of the returned output string.

See the ADARUN parameter CDXnn in the *Adabas Operations* documentation for more information.

Notes:

1. A collation descriptor can be defined for an alphanumeric (A) or wide alphanumeric (W) parent field. The format, length, and options (except UQ and XI) are taken from the parent field defined in the COLDE parameter. The unique descriptor (UQ) and exclude index (XI) options are separately defined for the collation descriptor itself.
2. A search using a collation descriptor value is performed in the same manner as for standard descriptors.
3. The user is responsible for creating correct collation descriptor values. There is no standard way to check the values of a collation descriptor for completeness against the Data Storage. The maintenance utility ADAICK only checks the structure of an index, not the contents. The user must set the rules for each value definition and check the value for correctness.
4. If a file contains more than one collation descriptor, the assigned exits are called in the alphabetical order of the collation descriptor names.

Collation Descriptor Syntax

A collation descriptor is defined using the following syntax:

```
COLDE = 'number, name [, UQ [, XI ]] = parent-field'
```

where

<i>number</i>	is the user exit number to be assigned to the collation descriptor. The Adabas nucleus uses this number to determine the collation descriptor user exit to be called.
<i>name</i>	is the name to be used for the collation descriptor. The naming conventions for collation descriptors are identical to those for Adabas field names.
<i>UQ</i>	indicates that the unique descriptor option is to be assigned to the collation descriptor.
<i>XI</i>	indicates that the uniqueness of the collation descriptor is to be determined with the index (occurrence) number excluded.
<i>parent-field</i>	is the name of an elementary A or W field. A collation descriptor can have one parent field. The field name and address is passed to the user exit.

MU, NU, and PE options are taken from the parent field and are implicitly set in the collation descriptor.

If a parent field with the NU option is specified, no entries are made in the collation descriptor's inverted list for those records containing a null value for the field. This is true regardless of the presence or absence of values for other collation descriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated, for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

Collation Descriptor Definition Example:

Field definition:

```
FNDEF='01, LN, 20, A, DE, NU'      Last-Name
```

Collation descriptor definition:

```
COLDE='1, Y2=LN'
```

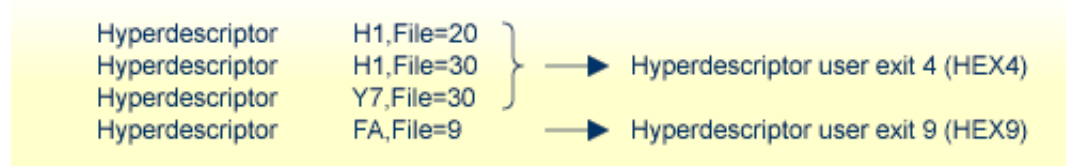
- Collation descriptor user exit 1 (CDX01) is assigned to this collation descriptor, and the name is Y2.
- The collation descriptor length and format are taken from the parentfield: 20 and alphanumeric, respectively. The collation descriptor is a multiple value (MU) field with null suppression (NU).
- The values for the collation descriptor are to be derived from the parentfield LN.

HYPDE: Hyperdescriptor Definition

The hyperdescriptor option enables descriptor values to be generated, based on a user-supplied algorithm.

The values are based on algorithms coded in special hyperdescriptor user exits (HEX01 through HEX31). Each hyperdescriptor must be assigned to a user exit, and a single user exit may handle multiple hyperdescriptors.

Example:



The exit is called whenever a hyperdescriptor value is to be generated by the Adabas nucleus or by the ADACMP utility.

Input parameters supplied to the user exit are

- hyperdescriptor name
- file number
- addresses of fields taken from the Data Storage record, together with field name and PE index (if applicable). These addresses point to the compressed values of the fields. The names of these fields must be defined using the HYPDE parameter of ADACMP or ADAINV.

The user exit must return the descriptor value(s) (DVT) in compressed format. No value, or one or more values may be returned depending on the options (PE, MU) assigned to the hyperdescriptor.

The original ISN assigned to the input value(s) may be changed.

See the *Adabas DBA Reference* documentation, section *User Exits*, for more information about the hyperdescriptor user exit.

Notes:

1. The format, the length, and the options of a hyperdescriptor are user-defined. They are not taken from the parent fields defined in the HYPDE parameter.
2. A search using a hyperdescriptor value is performed in the same manner as for standard descriptors.
3. The user is responsible for creating correct hyperdescriptor values. There is no standard way to check the values of a hyperdescriptor for completeness against the Data Storage. The maintenance utility ADAICK only checks the structure of an index, not the contents. The user must set the rules for each value definition and check the value for correctness.
4. If a hyperdescriptor is defined as packed or unpacked format, Adabas checks the returned values for validity. The sign half-byte for packed values can contain A, C, E, F (positive) or B, D (negative). Adabas converts the sign to F or D.
5. If a file contains more than one hyperdescriptor, the assigned exits are called in the alphabetical order of the hyperdescriptor names.

Hyperdescriptor Syntax

A hyperdescriptor is defined using the following syntax:

```
HYPDE = 'number, name, length, format [ { , option } ... ] = { parent-field }, ...'
```

where

<i>number</i>	is the user exit number to be assigned to the hyperdescriptor. The Adabas nucleus uses this number to determine the hyperdescriptor user exit to be called.														
<i>name</i>	is the name to be used for the hyperdescriptor. The naming conventions for hyperdescriptors are identical to those for Adabas field names.														
<i>length</i>	is the default length of the hyperdescriptor.														
<i>format</i>	<p>is the format of the hyperdescriptor:</p> <table border="1"> <thead> <tr> <th>Format</th> <th>Maximum Length</th> </tr> </thead> <tbody> <tr> <td>Alphanumeric (A)</td> <td>253 bytes</td> </tr> <tr> <td>Binary (B)</td> <td>126 bytes</td> </tr> <tr> <td>Fixed Point (F)</td> <td>4 bytes (always 4 bytes)</td> </tr> <tr> <td>Floating Point (G)</td> <td>8 bytes (always 4 or 8 bytes)</td> </tr> <tr> <td>Packed Decimal (P)</td> <td>15 bytes</td> </tr> <tr> <td>Unpacked Decimal (U)</td> <td>29 bytes</td> </tr> </tbody> </table> <p>Note: Wide-character (W) format is not valid for a hyperdescriptor.</p>	Format	Maximum Length	Alphanumeric (A)	253 bytes	Binary (B)	126 bytes	Fixed Point (F)	4 bytes (always 4 bytes)	Floating Point (G)	8 bytes (always 4 or 8 bytes)	Packed Decimal (P)	15 bytes	Unpacked Decimal (U)	29 bytes
Format	Maximum Length														
Alphanumeric (A)	253 bytes														
Binary (B)	126 bytes														
Fixed Point (F)	4 bytes (always 4 bytes)														
Floating Point (G)	8 bytes (always 4 or 8 bytes)														
Packed Decimal (P)	15 bytes														
Unpacked Decimal (U)	29 bytes														
<i>option</i>	<p>is an option to be assigned to the hyperdescriptor. The following options may be used together with a hyperdescriptor:</p> <ul style="list-style-type: none"> ● MU (multiple-value field) ● NU (null-value suppression) ● PE (field of a periodic group) ● UQ (unique descriptor) <p>The parent field of a hyperdescriptor <i>cannot</i> be a long alphanumeric (LA) field.</p>														
<i>parent-field</i>	<p>is the name of an elementary field. A hyperdescriptor can have 1-20 parent fields. The field names and addresses are passed to the user exit.</p> <p>Note: A hyperdescriptor parent-field may not have W (wide-character) format.</p>														

If a parent field with the NU option is specified, no entries are made in the hyperdescriptor's inverted list for those records containing a null value for the field. This is true regardless of the presence or absence of values for other hyperdescriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated, for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

Hyperdescriptor Definition Example:

Field definitions:

FNDEF='01, LN, 20, A, DE, NU'	Last-Name
FNDEF='01, FN, 20, A, MU, NU'	First-Name
FNDEF='01, ID, 4, B, NU'	Identification
FNDEF='01, AG, 3, U'	Age
FNDEF='01, AD, PE'	Address
FNDEF='02, CI, 20, A, NU'	City
FNDEF='02, ST, 20, A, NU'	Street
FNDEF='01, FA, PE'	Relatives
FNDEF='02, NR, 20, A, NU'	R-Last-Name
FNDEF='02, FR, 20, A, MU, NU'	R-First-Name

Hyperdescriptor definition:

```
HYPDE='2, HN, 60, A, MU, NU=LN, FN, FR'
```

- Hyperdescriptor user exit 2 is assigned to this hyperdescriptor, and the name is HN.
- The hyperdescriptor length is 60, the format is alphanumeric, and is a multiple-value (MU) field with null suppression (NU).
- The values for the hyperdescriptor are to be derived from fields LN, FN and FR.

The ADACMP HYPDE= statement may be continued on another line, as shown in the following example. To do so, first specify a minus (-) after a whole argument and before the closing apostrophe on the first line. Then enter the remaining positional arguments, beginning after the statement name (ADACMP) enclosed in apostrophes on the following line:

```
ADACMP HYPDE='1, HY, 20, A=AA, BB, CC, -'  
ADACMP          'DD, EE, FF'
```

PHONDE: Phonetic Descriptor

The use of a phonetic descriptor in a FIND command results in the return of all the records that contain similar phonetic values. The phonetic value of a descriptor is based on the first 20 bytes of the field value. Only alphabetic values are considered; numeric values, special characters, and blanks are ignored. Lower- and uppercase alphanumeric characters are internally identical.

A phonetic descriptor is defined using the following syntax:

```
PHONDE = ' name (field)'
```

where

<i>name</i>	is the name to be used for the phonetic descriptor. The naming conventions for phonetic descriptors are identical to those for Adabas field names.
<i>field</i>	is the name of the field to be phoneticized.

The field *must* be

- an elementary or a multiple value field; and
- defined with alphanumeric format.

The field can be a descriptor.

The field *cannot* be

- a subdescriptor, superdescriptor, or hyperdescriptor;
- contained within a periodic group;
- used as the source field for more than one phonetic descriptor.
- format W (wide-character)

The parent field of a phonetic descriptor *cannot* be a long alphanumeric (LA) field.

If the field is defined with the NU option, no entries are made in the phonetic descriptor's inverted list for those records that contain a null value (within the byte positions specified) for the field. The format is the same as for the field.

If the field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

Phonetic Descriptor Definition Example:

Field definition:

```
FNDEF= ' 01 , AA , 20 , A , DE , NU '
```

Phonetic definition:

```
PHONDE= ' PA ( AA ) '
```

SUBDE: Subdescriptor Definition

A subdescriptor is a descriptor created from a portion of an elementary field. The elementary field may or may not be a descriptor itself. A subdescriptor can also be used as a subfield; that is, it can be specified in the format buffer to control the record's output format.

A subdescriptor definition is entered using the following syntax:

```
SUBDE = 'name [, UQ [ XI ]] = parent-field (begin, end)'
```

where

<i>name</i>	is the subdescriptor name. The naming conventions for a subdescriptor are identical to those for Adabas field names.
<i>UQ</i>	indicates that the subdescriptor is to be defined as unique (see the definition of option UQ).
<i>XI</i>	indicates that the uniqueness of the subdescriptor is to be determined with the index (occurrence) number excluded.
<i>parent-field</i>	is the name of the field from which the subdescriptor is to be derived.
<i>begin</i>	is the relative byte position within the parent field where the subdescriptor definition is to begin.
<i>end</i>	is the relative byte position within the parent field where the subdescriptor definition is to end.

* Counting is from left to right beginning with 1 for alphanumeric or wide-character fields, and from right to left beginning with 1 for numeric or binary fields. If the parent field is defined with P format, the sign of the resulting subdescriptor value is taken from the 4 low-order bits of the low-order byte (that is, byte 1).

A parent field of a subdescriptor can be

- a descriptor
- an elementary field
- a multiple-value field (but *not* a particular occurrence of a multiple-value field)
- contained within a periodic group (but *not* a particular occurrence of a periodic group)

A parent field or a subdescriptor *cannot* be

- a sub/super field, subdescriptor, superdescriptor, or phonetic descriptor
- format G (floating point)
- a long alphanumeric (LA) field.

If the parent field is defined with the NU option, no entries are made in the subdescriptor's inverted list for those records that contain a null value (within the byte positions specified) for the field. The format is the same as for the parent field.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

Subdescriptor Definition Example 1:

Parent-field definition:

```
FNDEF='01,AR,10,A,NU'
```

Subdescriptor definition:

```
SUBDE='SB=AR(1,5)'
```

The values for subdescriptor SB are derived from the first five bytes (counting from left to right) of all the values for the parent field AR. All values are shown in character format.

AR Values	SB Values
DAVENPORT	DAVEN
FORD	FORD
WILSON	WILSO

Subdescriptor Definition Example 2:

Parent-field definition:

```
FNDEF='02,PF,6,P'
```

Subdescriptor definition:

```
SUBDE='PS=PF(4,6)'
```

The values for subdescriptor PS are derived from bytes 4 to 6 (counting from right to left) of all the values for the parent field PF. All values are shown in hexadecimal.

PF Values	PS Values
00243182655F	02431F
00000000186F	0F (see note)
78426281448D	0784262D

Note:

If the NU option had been specified for parent field PF, no value would have been created for PS for this value.

Subdescriptor Definition Example 3:

Source-field definition:

```
FNDEF='02,PF,6,P'
```

Subdescriptor definition:

```
SUBDE='PT=PF(1,3)'
```

The values for PT are derived from bytes 1 to 3 (counting from right to left) of all the values for PF. All values are shown in hexadecimal.

PF Values	PT Values
00243182655F	82655F
00000000186F	186F
78426281448D	81448D

SUBFN: Subfield Definition

A subfield

- is a portion of an elementary field that can be read using an Adabas read command;
- cannot be updated;
- can be changed to a subdescriptor using ADAINV INVERT SUBDE=... .

A subfield definition is entered using the following syntax:

```
SUBFN = ' name = parent-field (begin, end)'
```

where

<i>name</i>	is the subfield name. The naming conventions for a subfield are identical to those for Adabas field names.
<i>parent-field</i>	is the name of the field from which the subfield is to be derived.
<i>begin*</i>	is the relative byte position within the parent field where the subfield definition is to begin.
<i>end*</i>	is the relative byte position within the parent field where the subfield definition is to end.

* Counting is from left to right beginning with 1 for alphanumeric or wide-character fields, and from right to left beginning with 1 for numeric or binary fields. If the parent field is defined with "P" format, the sign of the resulting subfield value is taken from the 4 low-order bits of the low-order byte (that is, byte 1).

The parent field for a subfield can be:

- a multiple-value field
- within a periodic group

The parent field for a subfield *cannot*:

- have format "G" (floating point)
- be a long alphanumeric (LA) field.

Subfield Definition Example:

```
SUBFN= ' X1=AA ( 1 , 2 ) '
```

SUPDE: Superdescriptor Definition

A superdescriptor is a descriptor created from several fields, portions of fields, or a combination thereof.

Each source field (or portion of a field) used to define a superdescriptor is called a *parent* . From 2 to 20 parent fields or field portions may be used to define a superdescriptor.

A superdescriptor may be defined as a unique descriptor.

A superdescriptor can be used as a superfield; that is, it can be specified in the format buffer to determine the record's output format.

A superdescriptor description has the following syntax:

```
SUPDE = ' name [ , UQ [ XI ] ] = { parent-field ( begin , end ) } , ...'
```

where

<i>name</i>	is the superdescriptor name. The naming conventions for superdescriptors are identical to those for Adabas names.
UQ	indicates that the superdescriptor is to be defined as unique (see the definition option UQ).
XI	indicates that the uniqueness of the superdescriptor is to be determined with the index (occurrence) number excluded.
<i>parent-field</i>	is the name of a parent field from which a superdescriptor element is to be derived; up to 20 parent fields can be specified.
<i>begin*</i>	is the relative byte position within the field where the superdescriptor element begins.
<i>end*</i>	is the relative byte position within the field where the superdescriptor element is to end.

** Counting is from left to right beginning with 1 for fields defined with alphanumeric or wide-character format, and from right to left beginning with 1 for fields defined with numeric or binary format. For any parent field except those defined as "FI", any begin and end values within the range permitted for the parent field's data type are valid.*

A parent field of a superdescriptor can be

- an elementary field; or
- a maximum of one multiple-value field (but not a specific multiple-value field value);
- within a periodic group (but not a specific occurrence);
- a descriptor.

A parent field of a superdescriptor *cannot* be

- a super-, sub-, or phonetic descriptor;
- format G (floating point);
- an NC option field if another parent field is an NU option field;
- a long alphanumeric (LA) field.

If a parent field with the NU option is specified, no entries are made in the superdescriptor's inverted list for those records containing a null value for the field. This is true regardless of the presence or absence of values for other superdescriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

The total length of any superdescriptor value may not exceed 253 bytes (alphanumeric) or 126 bytes (binary).

The superdescriptor format is B (binary) if no element of the superdescriptor is derived from an A (alphanumeric) or W (wide-character) parent field; if any element of the superdescriptor is derived from an A or W parent field, the format of the superdescriptor reflects the last occurring A or W element; for example, if the last occurring A or W element is W, the format of the superdescriptor is W.

All binary format superdescriptor values are treated as unsigned numbers.

The ADACMP SUPDE= statement may be continued on another line by specifying a minus (-) after an argument just before the closing apostrophe on the first line. Then enter the remaining positional arguments enclosed in apostrophes on the following line beginning after the statement name (ADACMP). For example:

```
ADACMP SUPDE='SI=AA(10,20),BB(20,21),-'  
ADACMP      'CC(12,13),DD(14,15)'
```

- Superdescriptor Definition Example 1:
- Superdescriptor Definition Example 2:
- Superdescriptor Definition Example 3:
- Superdescriptor Definition Example 4:
- Superdescriptor Definition Example 5:
- Format Conversions of Superdescriptors

Superdescriptor Definition Example 1:

Field definitions:

```
FNDEF='01, LN, 20, A, DE, NU'    Last-Name  
FNDEF='01, FN, 20, A, MU, NU'    First-Name  
FNDEF='01, ID, 4, B, NU'        Identification  
FNDEF='01, AG, 3, U'            Age  
FNDEF='01, AD, PE'              Address  
FNDEF='02, CI, 20, A, NU'        City  
FNDEF='02, ST, 20, A, NU'        Street  
FNDEF='01, FA, PE'              Relatives  
FNDEF='02, NR, 20, A, NU'        R-Last-Name  
FNDEF='02, FR, 20, A, MU, NU'    R-First-Name
```

Superdescriptor definition:

```
SUPDE='SD=LN(1,4),ID(3,4),AG(2,3)'
```

Superdescriptor SD is to be created. The values for the superdescriptor are to be derived from bytes 1 to 4 of field LN (counting from left to right), bytes 3 to 4 of field ID (counting from right to left), and bytes 2 to 3 of field AG (counting from right to left). All values are shown in hexadecimal.

LN	ID	AG	SD
C6D3C5D4C9D5C7	00862143	F0F4F3	C6D3C5D40086F0F4
D4D6D9D9C9E2	02461866	F0F3F8	D4D6D9D90246F0F3
D7C1D9D2C5D9	00000000	F0F3F6	No value is stored (because of ID)
404040404040	00432144	F0F0F0	No value is stored (because of LN)
C1C1C1C1C1C1	00000144	F1F1F1	C1C1C1C10000F1F1
C1C1C1C1C1C1	00860000	F0F0F0	C1C1C1C10086F0F0

The format for SD is alphanumeric since at least one element is derived from a parent field defined with alphanumeric format.

Superdescriptor Definition Example 2:

Field definitions:

FNDEF='01, LN, 20, A, DE, NU'	Last-Name
FNDEF='01, FN, 20, A, MU, NU'	First-Name
FNDEF='01, ID, 4, B, NU'	Identification
FNDEF='01, AG, 3, U'	Age
FNDEF='01, AD, PE'	Address
FNDEF='02, CI, 20, A, NU'	City
FNDEF='02, ST, 20, A, NU'	Street
FNDEF='01, FA, PE'	Relatives
FNDEF='02, NR, 20, A, NU'	R-Last-Name
FNDEF='02, FR, 20, A, MU, NU'	R-First-Name

Superdescriptor definition:

SUPDE='SY=LN(1,4),FN(1,1)'

Superdescriptor SY is to be created from fields LN and FN (which is a multiple-value field). All values are shown in character format.

LN	FN	SY
FLEMING	DAVID	FLEMD
MORRIS	RONALD RON	MORRR MORRR
WILSON	JOHN SONNY	WILSJ WILSS

The format of SY is alphanumeric since at least one element is derived from a parent field defined with alphanumeric format.

Superdescriptor Definition Example 3:

Field definitions:

```
FNDEF='01,PN,6,U,NU'
FNDEF='01,NA,20,A,DE,NU'
FNDEF='01,DP,1,B,FI'
```

Superdescriptor definition:

```
SUPDE='SZ=PN(3,6),DP(1,1)'
```

Superdescriptor SZ is to be created. The values for the superdescriptor are to be derived from bytes 3 to 6 of field PN (counting from right to left), and byte 1 of field DP. All values are shown in hexadecimal.

PN	DP	SZ
F0F2F4F6F7F2	04	F0F2F4F604
F8F4F0F3F9F8	00	F8F4F0F300
F0F0F0F0F1F1	06	F0F0F0F006
F0F0F0F0F0F1	00	F0F0F0F000
F0F0F0F0F0F0	00	no value is stored (because of PN)
F0F0F0F0F0F0	01	no value is stored (because of PN)

The format of SZ is binary since no element is derived from a parent field defined with alphanumeric format. A null value is not stored for the last two values shown because the superdescriptor option is NU (from the PN field) and the PN field value contains unpacked zeros (X'F0'), the null value.

Superdescriptor Definition Example 4:

Field definitions:

```
FNDEF='01,PF,4,P,NU'
FNDEF='01,PN,2,P,NU'
```

Superdescriptor definition:

```
SUPDE='SP=PF(3,4),PN(1,2)'
```

Superdescriptor SP is to be created. The values for the superdescriptor are to be derived from bytes 3 to 4 of field PF (counting from right to left), and bytes 1 to 2 of field PN (counting from right to left). All values are shown in hexadecimal.

PF	PN	SP
0002463F	003F	0002003F
0000045F	043F	0000043F
0032464F	000F	No value is stored (because of PN)
0038000F	044F	0038044F

The format of SP is binary since no element is derived from a parent field defined with alphanumeric format.

Superdescriptor Definition Example 5:

Field definitions:

```
FNDEF=' 01,AD,PE'
FNDEF=' 02,CI,4,A,NU'
FNDEF=' 02,ST,5,A,NU'
```

Superdescriptor definition:

```
SUPDE='XY=CI(1,4),ST(1,5)'
```

Superdescriptor XY is to be created from fields CI and ST. All values are shown in character format.

CI	ST	XY
(1st occ.) BALT	(1st occ.) MAIN	BALTMAN
(2nd occ.) CHI	(2nd occ.) SPRUCE	CHI SPRUC
(3rd occ.) WASH	(3rd occ.) 11TH	WASH11TH
(4th occ.) DENV	(4th occ.) bbbbb	No value stored (because of ST)

The format of XY is alphanumeric since at least 1 element is derived from a parent field which is defined with alphanumeric format.

Format Conversions of Superdescriptors

Superdescriptors have a final superdescriptor format which is calculated as follows:

Superdescriptor Format	Used if:
A (alphabetic)	At least one parent field has format A.
U (unpacked, decimal)	All parent fields have format U and format U is explicitly specified in the superdescriptor definition
B (binary)	All other superdescriptor fields (those with no parent fields with format A or that meet the requirements for format U listed above).

This section covers the following topics:

- Format Conversions During Updates
- Format Conversions In Value Buffers
- Format Conversions For Output (L9 Command)

Note:

Conversions are only performed if a parent field has not been set with the NV option.

Format Conversions During Updates

Superdescriptors should be built so that they have the same collating sequence in all environments. However, problems exist for some combinations, as described in this section:

- *Alphanumeric (Format A) Values in IBM (EBCDIC) and UNIX (ASCII) Environments*
- *Numeric Values (Format U) in IBM (EBCDIC) and UNIX (ASCII) Environments*
- *Binary Values (Formats B, F, G) with Big-endian and Little-endian Storage Formats*
- *Different Packed Value Signs (Format P)*

Alphanumeric (Format A) Values in IBM (EBCDIC) and UNIX (ASCII) Environments

All alphabetic field values will be converted from EBCDIC to ASCII if an insert or update call comes from an IBM mainframe environment to a UNIX database. Consequently, the superdescriptor parent values are automatically converted to ASCII. In this case, an application might fail if it expects a specific sort sequence (for example using uppercase and lowercase characters). In EBCDIC formats, lowercase characters come prior to uppercase characters; in ASCII formats, this sequence is reversed (uppercase characters come prior to lowercase characters).

One of two methods can be used to resolve this problem:

- Use the NV option on parent fields with EBCDIC-ASCII conflicts. This will disable the EBCDIC-ASCII conversion.
- Use a hyperdescriptor instead of a superdescriptor.

Numeric Values (Format U) in IBM (EBCDIC) and UNIX (ASCII) Environments

All numeric field values will be converted from EBCDIC to ASCII if an insert or update call comes from an IBM mainframe environment to a UNIX database. Consequently, the superdescriptor parent values are automatically converted to ASCII, even if the final superdescriptor requests formats of A (alphabetic), B (binary), or U (unpacked).

Binary Values (Formats B, F, G) with Big-endian and Little-endian Storage Formats

Some platforms store binary byte sequences in big-endian format; others store them in little-endian format. For example, IBM and HP-UX processors use big-endian format (the byte significance runs from right to left), while Intel processors use little-endian sequence (the byte significance runs from left to right).

Adabas performs conversions on superdescriptors containing binary values in swapped architectures (little-endian binary values with significance running from left to right) to get them into a standard sort sequence before storing them in the index.

- For alphabetic superdescriptors containing at least one binary field with parent lengths greater than one, the binary parent values will be swapped.
- For binary superdescriptors, the order of the parent entries will be swapped and the non-binary parent values will be swapped.

Different Packed Value Signs (Format P)

Sign information of packed values is represented differently on different platforms. Adabas on open systems converts positive values (A, C, or F) to C and negative values (B or D) to D. Adabas for mainframes uses F to represent positive values. Consequently, collating sequence problems arise if packed values are used in superdescriptors because the packed value signs lose their meaning; they become normal bit patterns. When this happens, positive packed values can be sorted as negative packed values.

In addition, when combined in a superdescriptor in Adabas for mainframes, negative packed values are sorted before positive packed values, while on Adabas for open systems, positive packed values are sorted before negative packed values.

To resolve these problems, we recommend that you use a hyperdescriptor instead of a superdescriptor.

Format Conversions In Value Buffers

When superdescriptors are specified in a value buffer, they are converted so they can be matched to an associated index entry.

Format Conversions For Output (L9 Command)

Superdescriptor values retrieved by L9 commands must be converted before they are returned in the record buffer. Alphabetic fields are converted from ASCII to EBCDIC, if required. In addition, binary parts of the superdescriptor are swapped if necessary. The packed signs of packed value parts of the superdescriptor are not converted.

SUPFN: Superfield Definition

A superfield is a field composed of several fields, portions of fields, or combinations thereof, which may be read using an Adabas read command. A superfield *cannot*:

- be updated;
- comprise fields defined with the NC option if another parent field has the NU option;
- be used as a descriptor.

A superfield *can* be changed to a superdescriptor using the ADAINV utility function INVERT SUPDE=....

A superfield is defined using the following syntax:

```
SUPFN = 'name = parent-field (begin , end)[, parent-field ( begin , end )]...'
```

where

<i>name</i>	superfield name. The naming conventions for superfields are identical to those for Adabas names.
<i>parent-field</i>	name of the field from which a superfield element is to be derived.
<i>begin*</i>	relative byte position within the field where the superfield element is to begin.
<i>end*</i>	relative byte position within the field where the superfield element is to end.

* Counting is from left to right beginning with 1 for fields defined with alphanumeric or wide-character format, and from right to left beginning with 1 for fields defined with numeric or binary format.

A parent field of a superfield can be:

- a multiple-value field
- contained within a periodic group

A parent field of a superfield *cannot*:

- have format "G" (floating point)
- be a long alphanumeric (LA) field.

The total length of any superfield value may not exceed 253 bytes (alphanumeric) or 126 bytes (binary).

The superfield format is B (binary) if no element of the superfield is derived from an A (alphanumeric) or W (wide-character) parent field; if any element of the superfield is derived from an A or W parent field, the format of the superfield reflects the last occurring A or W element; for example, if the last occurring A or W element is W, the format of the superfield is W.

Superfield Definition Example:

```
SUPFN='X2=AA(1,2),AB(1,4),AC(1,1)'
```

ADACMP COMPRESS Examples

Example 1:

ADACMP	COMPRESS	
ADACMP	FNDEF='01,AA,7,A,DE,FI'	Field AA
ADACMP	FNDEF='01,AB,15,A,DE,MU,NU'	Field AB
ADACMP	FNDEF='01,GA'	Group GA
ADACMP	FNDEF='02,AC,15,A,NU'	Field AC
ADACMP	FNDEF='02,AD,2,P,FI'	Field AD
ADACMP	FNDEF='02,AE,5,P,NU'	Field AE
ADACMP	FNDEF='02,AF,6,W'	Field AF
ADACMP	COLDE='7,Y1=AF'	Collation descriptor Y1
ADACMP	SUBDE='BB=AA(1,4)'	Subdescriptor BB
ADACMP	SUPDE='CC=AA(1,4),AD(1,1)'	Superdescriptor CC
ADACMP	HYPDE='1,DD,4,A,MU=AB,AC,AD'	Hyperdescriptor DD
ADACMP	PHONDE='EE(AA)'	Phonetic descriptor EE
ADACMP	SUBFN='FF=AA(1,2)'	Subfield FF
ADACMP	SUPFN='GG=AA(1,4),AD(1,1)'	Superfield GG

Field AA	is defined as level 1, 7 bytes alphanumeric, descriptor, fixed storage option.
Field AB	is defined as level 1, 15 bytes alphanumeric, descriptor, multiple value field, null value suppression.
GA	is a group containing fields AC, AD, AE, and AF.
BB	is a subdescriptor (positions 1-4 of field AA).
CC	is a superdescriptor (positions 1-4 of field AA and position 1 of field AD).
DD	is a hyperdescriptor consisting of fields AB, AC and AD. DD is assigned hyperexit 1.
EE	is a phonetic descriptor derived from field AA.
FF	is a subfield (positions 1-2 of field AA).
GG	is a superfield (positions 1-4 of AA and position 1 of AD).
Y1	is a collation descriptor for AF and is assigned to collation descriptor user exit 7 (CDX07).

Example 2:


```

ADACMP COMPRESS
ADACMP FORMAT='AG,6,U,AF,4X,AA,' input record format
ADACMP FORMAT='AB,AC' continuation of FORMAT statement
ADACMP FNDEF='01,AA,10,A,NU' field definitions
ADACMP FNDEF='01,AB,7,U,NU'
ADACMP FNDEF='01,AF,5,P,NU'
ADACMP FNDEF='01,AG,12,P,NU,DE'
ADACMP FNDEF='01,AC,3,A,NU,DE'

```

The input record format is provided explicitly using the FORMAT parameter. ADACMP uses this format as the basis for processing fields from the input record. The FDT for the file corresponds to the structure specified in the FNDEF statements.

Example 3:

```

ADACMP COMPRESS
ADACMP FORMAT='AG,AF,4X,AA,AB,AC' input record format
ADACMP FDT=8 FDT same as file 8

```

The input record format is provided explicitly using the FORMAT parameter. The FDT to be used is the same as that currently defined for Adabas file 8.

Example 4:

```

ADACMP COMPRESS NUMREC=2000,USERISN
ADACMP FNDEF='01,AA,7,A,DE,FI' Field AA
ADACMP FNDEF='01,AB,15,A,DE,MU,NU' Field AB

```

The number of input records to be processed is limited to 2,000. The ISN for each record is to be provided by the user.

Example 5:

```

ADACMP COMPRESS RECFM=FB,LRECL=100
ADACMP FNDEF='01,AA,7,A,DE,FI' Field AA
ADACMP FNDEF='01,AB,15,A,DE,MU,NU' Field AB

```

A VSE input file contains fixed length (blocked) records. The record length is 100 bytes.