# Adabas for Linux, UNIX and Windows

## Command Reference

Version 6.7

October 2018

ADABAS & NATURAL

**Document ID: ADAOS-CMDREF-67-20211006**

## Table of Contents

# Command Reference

This document describes the Adabas commands for accessing and manipulating an Adabas database. The commands are generally embedded as calls to Adabas from within an application written in a third generation language such as C (the call interface is described in the document *Calling Adabas*).

This document is intended for software developers who wish to use Adabas direct calls to develop database applications.

The document consists of the following:

*Concepts and Facilities*, provides an overview of the types of commands available, and how database integrity is maintained in a multi–user environment.

*Calling Adabas*, provides linking information for Adabas application programs, and describes the standard calling procedure for Adabas user calls. It also describes in detail the Adabas Control Block, Format Buffer, Record Buffer, Search Buffer, Value Buffer and ISN Buffer, with many examples of usage.

*Programming Considerations*, describe Adabas programming features which can provide significant improvement in the performance of an application. The topics discussed are: using command IDs; special processing based on ISN lists; using the multifetch feature to retrieve multiple records with a single Adabas call.

*Adabas Commands*, provides a detailed description for each Adabas command.

*Appendix A, File Definitions*, provides a summary of the file and record definitions that are used in the examples throughout this documentation.

*Appendix B, File Definition for Sample Program*, contains the file definition used for the sample program in Appendix C.

*Appendix C* provides an example of an application program written in C, using Adabas calls to access and modify an Adabas database.

*Appendix D* lists the example files provided in the Adabas kit.

# 1 Conventions

The following syntax conventions are used in this documentation:

**{ ... }**
: items included in curly brackets are mandatory, i.e. you must supply a value.

**[ ... ]**
: items included in square brackets are optional, i.e. you do not have to supply a value.

**a | b**
: The vertical bar "|" means that you can supply one or other of the two values on either side of it, but not both. So "a | b" means that you can either supply "a" or "b" but not both.

**,...**
: A comma followed by three dots means that you can repeat the item to the left of the comma as often as you want. You must separate successive repetitions by a comma.

## Notation used in examples

The examples given in this documentation use the following notation:

Character strings are enclosed in double quotes. Hexadecimal values are preceded by the two characters "0x" ("^X"). The character 'b' within a character string denotes a blank character.

# 2 About this Documentation

## Document Conventions

| Convention | Description |
|---|---|
| **Bold** | Identifies elements on a screen. |
| `Monospace font` | Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties. |
| *`Italic`* | Identifies:<br><br>Variables for which you must supply values specific to your own situation or environment.<br>New terms the first time they occur in the text.<br>References to other documentation sources. |
| `Monospace font` | Identifies:<br><br>Text you must type in.<br>Messages displayed by the system.<br>Program code. |
| { } | Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols. |
| \| | Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the \| symbol. |
| [ ] | Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols. |
| ... | Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...). |

## Online Information and Support

### Software AG Documentation Website

You can find documentation on the Software AG Documentation website at **https://documentation.softwareag.com**.

### Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at **https://empower.softwareag.com/**.

You can find product information on the Software AG Empower Product Support website at **https://empower.softwareag.com**.

To submit feature/enhancement requests, get information about product availability, and download products, go to **Products**.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the **Knowledge Center**.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at **https://empower.softwareag.com/public_directory.aspx** and give us a call.

**Software AG Tech Community**

You can find documentation and other technical information on the Software AG Tech Community website at **https://techcommunity.softwareag.com**. You can:

- Access product documentation, if you have Tech Community credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

## Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

# 3 Concepts And Facilities

This chapter covers the following topics:

## Adabas Command Overview

Adabas provides a powerful and flexible set of commands to perform database operations.

This section provides an overview of the Adabas commands.

The commands have been categorized by function:

- Database Query
- Data Storage Read
- Associator Read
- Database Modification
- Logical Transaction Processing
- Checkpointing
- Special Purpose

### Database Query

The S1, S2 and S4 commands are used to perform database query. The S8 and S9 commands perform special processing of ISN lists resulting from a database query.

#### S1/S4

The S1/S4 command selects a set of records which satisfy given search criteria. The search criteria may be constructed using a single field/derived descriptor or several field/derived descriptors connected by logical operators.

Adabas returns, as a result of an S1/S4 command, the number of records which satisfy the search criteria, and a list of the ISNs of the selected records.

An option is available which permits the record identified by the first ISN in the resulting ISN list to be read from Data Storage.

The S4 command may be used to place the record identified by the first ISN in the ISN list in hold status. This prevents another user from updating the record until it is released.

#### S2

The S2 command is similar to the S1 command, but the ISNs of the records selected are returned in the sort sequence of the user-specified search criteria. Also, the search criteria can specify non-descriptor fields. Ascending or descending sequence may be specified.

#### S8 and S9

The S8 command performs logical operations on two ISN lists previously created by an S1/S4, S8 or S9 command.

The logical operations AND, OR, and NOT are permitted.

AND results in an ISN list containing ISNs that are present in both ISN lists.

OR results in an ISN list containing ISNs that are present in either ISN list.

NOT results in an ISN list containing ISNs that are present in the first list but not in the second list.

The S9 command sorts an ISN list created previously by a S1/S4, S2, S8 or S9 command.

The ISN list may be sorted by ascending ISN sequence, or by one to three user-specified fields (ascending or descending sequence).

The ISN lists resulting from any Sx command may be saved on the Adabas temporary working space for later retrieval during the user session.

### Data Storage Read

The L1-L6 commands are used to read records from Data Storage.

#### L1/L4

The L1 command reads a single record from Data Storage.The user specifies the file number, ISN of the record to be read and the fields for which values are to be returned. Adabas returns the requested field values in the desired format.

The L4 command is the same as the L1 command, except that the record is placed in hold status. This prevents other users from updating the record until it is released.

The GET NEXT option may be used to read one or more records identified by ISNs contained in an ISN list without the user having to specify each ISN.

The READ ISN SEQUENCE option may be used to read records in ISN sequence. The record with the ISN specified by the user is read, unless it is not present, in which case the record which has the next-highest ISN is read.

#### L2/L5

The L2 command reads the records from a file in the sequence in which they are physically stored in Data Storage. The user specifies the file to be read and the fields for which values are to be returned. Adabas returns the requested field values.

The L5 command is the same as the L2 command, except that the record read is placed in hold status. This prevents other users from updating the record until it is released.

#### L3/L6

The L3 command reads records from Data Storage in the logical sequence of a user-specified descriptor. The user specifies the file to be read, the descriptor to be used for sequence control, the value(s) at which the command is to begin and/or end and the fields for which values are to be returned. The records are returned in ascending or descending sequence of the specified descriptor's values.

The L6 command is the same as the L3 command, except that the record read is placed in hold status. This prevents other users from updating the record until it is released.

### Associator Read

The L9 and LF commands read information directly from the Associator.

#### L9

The L9 command returns each value contained in the inverted list for a given descriptor and the number of records in which the value is contained.

The user specifies the file and descriptor for which values are to be returned and the value(s) at which the command is to begin and/or end.

The values are returned in ascending or descending sequence.

#### LF

The LF command returns the field definitions for a file.

The user specifies the file for which the field definitions are to be returned.

The field definitions for all the fields in the file are returned. Each field definition consists of the field name, level number, standard format, standard length and definition options.

## Database Modification

The A1, E1 and N1/N2 commands are used to modify the database. The record to be modified with A1 or to be deleted with E1 must be held before the modification or deletion can be performed.

### A1

The A1 command updates the contents of one or more fields within a record. The user specifies the file and the ISN of the record to be updated, together with the fields to be updated and the values to be used for updating.

Adabas performs all necessary modifications to the Associator and Data Storage. Associator updating is required only if one or more descriptors are updated.

### E1

The E1 command deletes a record. The user specifies the file and ISN of the record to be deleted.

Adabas performs all necessary modifications to the Associator and Data Storage.

### N1/N2

The N1/N2 command adds a new record to a file. The user specifies the file to which the record is to be added together with the fields and field values to be used.

Adabas performs all necessary modifications to the Associator and Data Storage.

If the N1 command is used, the ISN for the new record is assigned by Adabas. If N2 is used, the ISN is provided by the user.

## Logical Transaction Processing

The BT, ET and RE commands are used for logical transaction processing. The primary purpose of using logical transaction commands is to permit a user restart beginning after the last successfully processed transaction in the event of an abnormal termination of the user or Adabas. The concept of subtransactions allows you to back out a unit of work within the current transaction, which is not the complete transaction.

### BT

The BT command backs out the current transaction or one or more subtransactions being processed.

All modifications resulting from updates, add and delete records performed during the transaction are removed, and all records placed in hold status during the transaction are released.

### ET

The ET command indicates the end of the current transaction or subtransaction.

An ET command causes Adabas to physically store all data protection information related to the transaction. This information is used to apply all the updates performed during the transaction at the start of the next Adabas session if the current session is terminated before these updates are physically applied to the database.

The ET command releases all the records which have been placed in hold status during the transaction.

The ET command may be used to store user data in an Adabas system file. This data may be retrieved with an OP or RE command and may be used in conjunction with a user restart.

### RE

The RE command reads user data previously stored in an Adabas system file with a CL or ET command.

## Checkpointing

The C1 command is used for user or Adabas checkpointing.

### C1

The C1 command causes a checkpoint to be taken.

The C1 command results in the physical writing of all current data protection information to the data protection log, and the writing of a checkpoint entry to the data protection log and the system checkpoint file. This checkpoint entry may be needed as a reference point for subsequent removal or reapplication of updates.

## Special Purpose

### CL

The CL command terminates a user session.

The CL command results in the physical writing of all current data protection information to the data protection log; the release of all records currently in hold status for the user; the release of all the command IDs (and corresponding ISN lists) currently assigned to the user and the storage of user data in an Adabas system file (optional).

### C5

The C5 command writes user data to the Adabas protection log.

### HI

The HI command places a record in hold status. The user specifies the file and ISN of the record to be placed in hold status.

A record placed in hold status cannot be updated by another user until it is released.

### MC

The MC command groups other Adabas commands together as subcalls in a single multi-call.

The MC command is only allowed with the ACB interface. If MC commands are performed with the ACBX interface, the result is undefined - this behaviour may be changed in one of the following update packages.

### OP

The OP command indicates the beginning of a user session. An OP command is required for:

- Users performing exclusive control updating;
- Access-only users;
- Users who wish to store user data in an Adabas system file for retrieval during a subsequent session;
- Users who are restricted to a subset of files in the database;
- Users who want to use subtransactions.

### RC

The RC command releases a command ID (or all command IDs) currently active for the user.

### RI

The RI command releases a record from hold status if it was not updated.

The user specifies the file and ISN of the record to be released. The user may also request that all records currently held by the user are to be released, provided that none of the records was updated.

## User Types

Depending on the type of file access and update performed by the user, Adabas recognizes several user types:

- Access-only user (user type "AC")
- ET logic user (user type "ET")
- Exclusive control user (user type "EX")
- Exclusive control user with ET logic (user type "EX, ET")
- Utility user (user type "UT")

### Access-only user

An access-only user makes read-only accesses to files. Such a user may not issue hold, update, delete, add record, ET or BT commands.

**Note:** Although shared locks may also seem to be useful for users who only want to read data, shared locks also require transactions, which are only supported for ET users. Therefore, L4-6 commands and S4 commands, which place records in shared or exclusive hold status are generally not allowed for access-only users: access-only users can only perform dirty-read operations.

A user becomes an access-only user for the current session by issuing an OP command with only the ACC parameter specified in the record buffer (see the OP command for more details).

Access-only users are identified in the ADAOPR DISPLAY=UQ display by "AC" in the Type column.

### ET logic user

An ET logic user uses transaction logic for the current session (see the section Logical Transaction Processing for further details). This means that all records that are updated during the course of a transaction must be kept in hold status. The transaction must end with an ET, BT or CL command, otherwise the whole transaction is backed out after the corresponding transaction duration time limit (TT) is exceeded (see the section Time Limits for further details).

An ET logic user can be at ET status or not. Being at ET status means that no record is held by this user.

A user becomes an ET logic user in each of the following cases (see the OP command for more details):

- by issuing an OP command with the UPD parameter specified in the record buffer.
- by starting the session without an OP command (implicit OP command).

ET logic users are identified in the ADAOPR DISPLAY=UQ display by "ET" in the Type column.

### Exclusive control user

An exclusive control user opens a file for reading or updating, and access to the file by other users is either restricted to read-only, or is prevented altogether while the exclusive control user session is active.

In addition to preventing competitive access or updating, exclusive control may be used to simplify recovery procedures, in that the file(s) may be restored regardless of other user activity.

There are two types of file access for an exclusive control user:

- Exclusive file updating: A file is opened for updating. Other users can read but not update the file. A user becomes an exclusive control user with exclusive file updating for the current session by issuing an OP command with the EXU parameter specified in the record buffer (see the OP command for more details).
- Exclusive file open: A file is opened for updating. Other users can neither read nor update the file. A user becomes an exclusive control user with exclusive file open for the current session by issuing an OP command with the EXF parameter specified in the record buffer (see the OP command for more details).

In general, exclusive control user sessions do not use transaction logic (see however the following section on exclusive control users with ET logic). They write an OPEN checkpoint at the start of the session and a CLSE checkpoint at the end of the session.

The decision whether to use record hold commands depends not only on the open mode of the current file but also on the user type. An exclusive user working with ET logic must keep all records in hold before they can be modified, in the same way as for real transaction processing, regardless of whether the file is opened in an exclusive update mode or not. Only an exclusive control user without ET logic can update records without ISN hold logic in files opened for exclusive use.

An exclusive control user cannot add an update file to its file list. This is, however, possible for an exclusive control user with ET logic (see the next section for further information).

Users performing exclusive control may use the C1 command to request a checkpoint to be written. This checkpoint may be subsequently used as a reference point to remove updates which have been applied after the checkpoint, or to re–apply the updates that were applied before the checkpoint.

Exclusive control users are identified in the ADAOPR DISPLAY=UQ display by "EX" in the Type column.

### Exclusive control user with ET logic

This applies when an exclusive control user wishes to use transaction logic for the current session. This means that all updates to a file are performed with transaction logic, even though the user has exclusive update access to the file. The OPEN and CLSE checkpoints are written at the beginning and end of the session, in the same way as for an exclusive control user.

An exclusive user working with ET logic must keep all records in hold before they can be modified, in the same way as for real transaction processing, regardless of whether the file is opened in an exclusive update mode or not.

A user becomes an exclusive control user with ET logic in each of the following cases (see the OP command for more details):

- The user issues an OP command, specifying in the record buffer either exclusive file updating (EXU) or exclusive file open (EXF), as well as concurrent file updating (UPD).
- The user issues an ET command in an exclusive control user session.

An exclusive control user with ET logic can add an update file to its file list. This is not possible for an exclusive control user without ET logic.

Users performing exclusive control may use the C1 command to request a checkpoint to be written. This checkpoint may be subsequently used as a reference point to remove updates which have been applied after the checkpoint, or to reapply the updates that were applied before the checkpoint.

Exclusive control users with ET logic are identified in the ADAOPR DISPLAY=UQ display by "EX,ET" in the Type column.

### Utility user

A utility user is defined if the session is started by an Adabas utility.

Utility users are identified in the ADAOPR DISPLAY=UQ display by "UT" in the Type column.

## Competitive Database Access

This section describes the Adabas facilities used to ensure data integrity in a competitive updating environment.

Competitive database access is in effect when two or more users are accessing the same Adabas file(s). Adabas addresses the following integrity problems that can come up in such a situation:

- Update transactions - it must not happen that updates are lost as a result of two users updating the same record at the same time.
- Read transactions - it must not happen that the records found by a search transaction are not consistent as a result of several records being updated in the same transaction; then you could see some records in the state before the update and some records in the state after the update.

Acquiring and releasing shared and exclusive locks, resource interlock and exclusive control updating, are presented in this section.

For this purpose Adabas supports shared or read (S) locks and exclusive or write (X) locks of records:

- A user can only get a shared lock for a record if no other user has an exclusive lock for the record or is waiting for an exclusive lock for the record. More than one user can have a shared lock for the same record at the same time. Acquiring a shared lock for a record is also called placing a record in shared hold status.
- A user can only get an exclusive lock for a record if no other user has a shared lock or an exclusive lock for the record. Acquiring an exclusive lock for a record is also called placing a record in exclusive hold status.
- These locks are not sufficient to fulfil all of the integrity requirements that can be necessary: Adabas first determines which records fulfill a search criterion, but only acquires a lock for the resulting records one after the other. Therefore, it may happen that at the time the records are read, some records may no longer fulfil the search criterion, or other records fulfilling the search criterion may exist before all records have been processed. If using shared and exclusive locks is not sufficient, you can lock the complete file for updates.

## Record Locking Commands

A record is locked by using the FIND WITH HOLD command (S4), READ WITH HOLD commands (L4, L5, L6), or a HOLD ISN (HI) command. An N1/N2 command issued by an ET Logic user also puts the record being added in exclusive hold status. A1 commands with an appropriate option and E1 commands can also cause a record to be exclusively locked.

The successful completion of any of these commands results in the record (ISN) being shared or exclusively locked. If this is not possible, because the record is already locked by another user, the user issuing the record hold command is placed in a wait status until the record becomes available, at which time Adabas automatically reactivates the command.

If the return option is used with any of the record hold commands and the record to be held is held by another user, Adabas returns response code 145 instead of placing the user in a wait status.

A user who issues a FIND (S1) or READ (L1, L2, L3) command is able to read the record regardless of the fact that the record is locked for another user (so-called dirty read).

## Locking Records with HI, L4, L5, L6 and S4 Commands

You can specify a given lock type by specifying the appropriate command option 3 for the commands HI, L4, L5, L6 and S4 (see table below). If the lock cannot be granted, Adabas does the following:

- If the Adabas command was issued with the return option, Adabas returns response code 145;
- If Adabas detects a deadlock situation, it returns response code 9 and rolls back the transaction to resolve the deadlock situation;
- Otherwise Adabas suspends the command execution until either the requested locking mode can be granted or the transaction time limit is exceeded.

The return option is specified in command option 1, which is also used to specify whether the multifetch option is used. The following table shows the mapping between command option 1 and multifetch option and return option:

| Command Option 1 | Multifetch Option | Return Option |
|---|---|---|
| M | Yes | No |
| O | Yes | Yes |
| R | No | Yes |
| Blank | No | No |

The lock type is specified with command option 3. This command option also specifies when a shared lock is released again. The following table shows the usage of command option 3:

| Command Option 3 | Lock Type | Time When Lock Is Released |
|---|---|---|
| Blank | X | ■ End or backout of transaction<br>■ RI command (only if record has not been updated) |
| C (not for HI) | S | ■ End of read command (only if record has not yet been locked before by another command) |
| Q (not with multifetch, L4 only with command option 2 = N, S4 only with non-blank, non-zero CID and ISN buffer length 4, not for HI) | S | ■ Start of next sequential read command or RC for this read sequence; if the same record has been read with command option Q in more than one command sequence, only when the next record has been read for all these command sequences, or an RC has been performed. It is not released if another command has locked the same record exclusively, or the record has been shared-locked with command option S.<br>■ One of the events that releases records shared locked with command option S. |
| S | S | ■ End or backout of transaction<br>■ Backout subtransaction<br>■ RI command |

**Notes:**

1. The S4 command puts only the first record in hold status.
2. The C option avoids dirty reads without keeping the record in shared hold status beyond the current read operation.
3. The Q option can be used to perform more than one consistent read operations on the same record, and the record is released again when the next record for the read sequence is read. If the current transaction is committed or rolled back before the next record is read, then this also implies the release of the lock.
4. It is possible to release the locks before the time defined for the command options blank, Q or S with an RI command, but an exclusive lock will be released only if the record has not been updated in the current transaction.
5. Since command option 3 is not contained in the ACB, shared locks are only supported only with the ACBX interface.

### Record Update Using Hold Option

A user may update/delete any record in exclusive hold status for that user by issuing an A1 or E1 command.

An A1 command will only be executed if the record is in exclusive hold status or the H, R, L, T or ' option was specified. If none of these options is used, and the record is not in hold status, response code 144 is returned. If the record is currently locked by another user and the R or U option is used, response code 145 is returned. If the record is currently locked by another user and the H or L option is used, the A1 command waits until the record is available again. If a deadlock is detected, response code 9 is returned.

For more information on using the T option with the A1 command, please refer tp *Programming Considerations, System Generated Fields*.

If an E1 command is issued for a record which is not in hold status for the user, Adabas will place the record in hold status for the user provided that the record is not in hold status for another user. If the record is currently locked by another user and the R option is used, response code 145 is returned. If the R option is not available, the E1 command waits until the record becomes available again. If a deadlock is detected, response code 9 is returned.

If a user does not place a record to be deleted in hold status when he reads the record before issuing the E1 command, there is no guarantee that the record will not be updated or deleted by another user before the E1 command is executed.

### Hold Queue Response Code Summary

Response code 9 is returned when Adabas detects a deadlock situation and performs a backout transaction to resolve the deadlock.

Response code 47 is returned if the user has reached the maximum number of held records allowed for one user (NISNHQ parameter of ADANUC and ADAOPR).

Response code 144 is returned when an A1 command is issued and the record is not in hold status for the user.

Response code 145 is returned if any of the following conditions arise:

- A command is issued which requires a record to be placed in hold status, the record is already held by another user and the return option (command option 1 field = R) was specified with the command. In this case, the value 0 is returned in the Additions 2 field;
- An A1, E1 or N1/N2 command was issued and there was no available entry in the hold queue for the record. In this case, the hexadecimal value FFFFFFFF is returned in the Additions 2 field.

## Record Lock Release

ET, BT, CL or OP commands, which commit or roll back a transaction, release the locks for all records locked by the user; for ET and BT it is also possible to keep the lock for a subset of the records.

The lock for a single record can also be released with the RI command. However the lock will not be released if the record has been updated in the current transaction.

The following example shows how commands must wait because of record hold logic:

```
USER 1                 USER 2                            USER 3

S4 with CO3=S:
Find ISN 1 and 2
Shared lock for ISN 1
Read ISN 1

                       S4 with CO3=C:
                       Find ISN 1 and 2
                       Shared lock for ISN 1
                       Read ISN 1
                       Release shared lock for ISN 1

                                                        S4 with CO3=blank:
                                                        Find ISN 1 and 2
L4 with CO3=S:                                          Wait for ISN 1
Shared Lock for ISN 2                                   ¦
Read next ISN 2                                         ¦
                                                        ¦
L4 with CO3=S:                                          ¦
EOF                                                     ¦
                                                        ¦
ET:                                                     ¦
Release ISNs 1 and 2                                    V
                                                      Exclusive lock for ISN 1
                                                        Read ISN 1

                                                        A1:
                                                        Update ISN 1

                                                        L4 with CO3=blank:
                                                      Exclusive lock for ISN 2
                                                        Read ISN 2

                      L4 with CO3=C:
                      Wait for next ISN 2
                      ¦                                 A1:
                      ¦                                 Update ISN 2
                      ¦
                      ¦                                 ET:
                      ¦                                   Commit updates
                      ¦                                   Release ISNs 1 and 2
                      V
                      Read ISN 2

                      L4 with CO3=C:
                      EOF
```

### Record Lock Upgrading and Downgrading

A record lock can be upgraded from shared to exclusive by using the same commands that acquire an exclusive lock for a record that is not yet locked in any way.

A record lock can be downgraded from exclusive to shared by an RI command with command option 3 = S, but only if the record has not been updated in the current transaction.

### Keeping Records in Hold Status beyond Transaction End

Normally when a transaction is committed or rolled back, all records locked for this transaction are released from hold status. However, it is also possible to keep some or all of the records in hold status:

- If you specify an ET or BT command with command option 1 = M, only the records specified in the ISN buffer are released from hold status. The remaining locks for the user remain unchanged.
- If you specify an ET or BT command with command option 3 = H, all records locked by the user remain in hold status, but exclusive locks are downgraded to shared locks.

### Subtransactions

The concept of subtransactions allows you to back out a unit of work within the current transaction, which is not the complete transaction. In Adabas, subtransactions include the delay of uniqueness checks and referential integrity checks until the end of a subtransaction.

A user session is enabled for subtransactions by an OP command with the S option. This OP command starts the first subtransaction:

- A subsequent ET command with the S option terminates the current subtransaction and starts a new subtransaction. A savepoint is defined; a savepoint means that you subsequently can back out all updates in the database performed after this savepoint. Each savepoint has a savepoint ID, which is returned in the command ID field in the Adabas control block for an ET command with S option. The start of a transaction always gets the savepoint ID 0. Usually the savepoint ID is incremented by 1 at the start of a new subtransaction, but the savepoint ID can remain unchanged if there were no locking or update activities in the previous subtransaction.
- A subsequent BT command with the S option backs out all updates performed after the savepoint which is specified in this BT command.

**Note:** If you performed an RI command in a subtransaction that is rolled back, the lock for the record remains lost. This is necessary since other users may have locked the record in the meantime. If the lock for an ISN is still required in the case of a backout subtransaction, you must not perform an RI command for this ISN.

- A subsequent BT command without the S option backs out the current complete transaction and also starts a new subtransaction.
- A subsequent ET command without the S option commits the current complete transaction and also starts a new subtransaction.

**Example:**

This example assumes the following command sequence:

```
ET=>CID=7… ET-S=>CID=1… ET-S=>CID=2… BT-S(CID=2)… ET-S=>CID=3… BT-S(CID=1)… ET=>CID=8
```

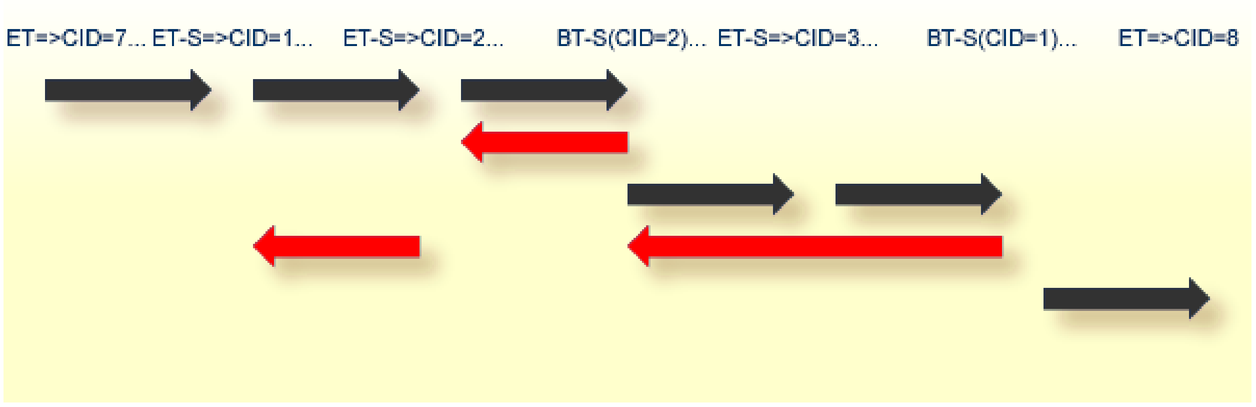
Note that there is a semantic difference between the CID returned by an end of transaction and the CID returned by an end of subtransaction:

- An end of transaction returns the transaction sequence number of the committed transaction.
- An end of subtransaction returns the current savepoint ID. The savepoint at the start of the first subtransaction always gets the savepoint ID 0; therefore, the savepoint ID of the first end of subtransaction after the ET gets the savepoint ID 1.

The first BT-S command is specified with CID=2, the last subtransaction (with savepoint ID 2) is rolled back, all update operations after the ET-S=>CID=2 are rolled back.

The second BT-S command is specified with CID=1, the previously started subtransaction is rolled back, all update operations after the last ET-S=>CID=1 are rolled back.

This means that the transaction finally contains the operations marked by the right (black) arrows where no left arrow is below; and the left (red) arrows show the update operations rolled back:



Note that for a BT command, independent of the BT command that is issued for a subtransaction or for the whole transaction, Adabas must not roll back operations again which already have been rolled back by a previous BT subtransaction.

Within a subtransaction, all Adabas commands and all Ux commands that log their modifications on WORK are permitted. A combination of both command types is also accepted.

**Note:** If a user session with subtransaction logic disappears after returning response code 9, a new OP command with the S option is necessary to continue with subtransaction logic.

The start and roll back of a subtransaction are always logged on WORK and PLOG.

## Referential Integrity

If referential integrity constraints are defined for a file, the referential integrity is checked at the end of a subtransaction, if subtransactions are activated, and at the end of a store/update delete command for the file otherwise. Please refer to *Administration, FDT Record Structure, Referential Constraints* for further information about how to define referential integrity constraints.

**Note:** A referential integrity check can result in a large number of implied database operations, e.g. cascaded deletes.

## Resource Interlock

Resource interlock would occur when two users are placed in wait status because each has requested a record which is currently in hold status for the other user. However, Adabas detects this situation, and resolves it by backing out the transaction of the user who would have caused the deadlock

**Deadlock Detection**

```
      USER 1                ADABAS               USER 2
READ WITH HOLD (L4)
FILE 1 ISN 1
                      ISN 1 HELD FOR
                      USER 1
                                             READ WITH HOLD
                                             (L4) FILE 1 ISN 2
                      ISN 2 HELD FOR
                      USER 2
READ WITH HOLD (L4)
FILE 1 ISN 2
                      USER 1 MUST WAIT...
                      ISN 2 HELD BY USER 2
                                             READ WITH HOLD
                                             (L4) FILE 1 ISN 1
                      DEADLOCK DETECTED
                      THE CURRENT TRANSACTION
```

```
                              OF USER 2 IS BACKED OUT.
                              USER 2 RECEIVES RESPONSE 9
                              AND 'DEADLOCK DETECTED' IN
                              THE ADDITIONS 2 FIELD OF THE
                              CONTROL BLOCK.
```

### Time Limits

Resource interlock is automatically resolved by Adabas by means of a transaction-duration time limit.

The time measurement for a transaction begins when the first command which results in a record being placed in hold status is issued, and ends when an ET, BT or CL command is issued.

If a transaction exceeds the prescribed limit, Adabas automatically generates a backout transaction (BT) command. This results in the removal of all the updates performed during the transaction and the release of all the records held during the transaction. Response code 9 is returned on the next call issued by the user.

Two types of time limits are defined:

1. The transaction time limit (TT), that is the maximum time interval in which a transaction must be performed. The transaction time limit applies only to ET users not at ET status;
2. The non–activity time limits (TNAx), i.e. time intervals after which certain actions are taken if no user activity occurred during these time intervals.

The values for these time limits can be set by an OP call, or by ADANUC or ADAOPR (see the *Adabas Utilities* for further information). User-defined values specified in the OP call for the current session override values defined in ADANUC or ADAOPR.

A user activity can be stopped with the STOP command in ADAOPR.

The following actions are performed if a time limit is exceeded and the nucleus is running without OPTIONS=OPEN_REQUIRED (the abbreviations used here are described after the tables):

| **USER TYPE** | **TIME OUT** | | | | |
|---|---|---|---|---|---|
| | **TT** | **TNAx** | | **STOP** | |
| | | **ID User** | **Non-ID User** | **ID User** | **Non-ID User** |
| Access only | - | SUQE | CL | SUQE | CL |
| ET logic user at ET status | - | SUQE | CL | SUQE | CL |
| ET logic user not at ET status | BT, RSP9 | BT, SUQE | | BT,SUQE | |
| Exclusive control user with ET logic, at ET status | - | CLSE checkpoint, SUQE | | CLSE checkpoint, SUQE | |
| Exclusive control user with ET logic, not at ET status | BT, RSP9 | BT, CLSE checkpoint, SUQE | | BT, CLSE checkpoint, SUQE | |

| USER TYPE | TIME OUT | | | | |
|---|---|---|---|---|---|
| | TT | TNAx | | STOP | |
| | | ID User | Non-ID User | ID User | Non-ID User |
| Exclusive control user without ET logic | - | CLSE checkpoint, SUQE | | CLSE checkpoint, SUQE | |
| Utility user | - | - | | CL | |

The following actions are performed if a time limit is exceeded and the nucleus is running with OPTIONS=OPEN_REQUIRED:

| USER TYPE | TIME OUT | | |
|---|---|---|---|
| | TT | TNAX | STOP |
| Access only | - | CL | CL |
| ET logic user at ET status | - | CL | CL |
| ET logic user not at ET status | BT, RSP9 | BT, CL | BT, CL |
| Exclusive control user with ET logic, at ET status | - | CL | CL |
| Exclusive control user with ET logic, not at ET status | BT, RSP9 | BT, CL | BT, CL |
| Exclusive control user without ET logic | - | CL | CL |
| Utility user | - | - | CL |

Note that user queue elements arising from global transactions will be handled differently. See the *Administration Manual*, section *XA Support, User Queue Handling* for related information.

SUQE (Scratch a user queue element) means: release all command IDs, scratch the file list, scratch the User ID (if any), scratch the user type, and set response 9 for the next call. After the next call has received this response 9, the user queue element is deleted: such user queue elements are not subject to timeout. (See the *OP command* for additional information about the user queue element).

BT means: Adabas backs out the user's current transaction.

CL means: Adabas closes the user session.

RSP9 means: Adabas will return response code 9 for the next user command.

ID User means: a user who has specified an identification during opening of a session, which is used later on to mark entries in logs caused by this user's activities.

### Example

In the example shown in the figure below, user 1 will eventually exceed the transaction time limit first, causing Adabas to backout user 1's transaction, thereby permitting user 2 to resume processing.

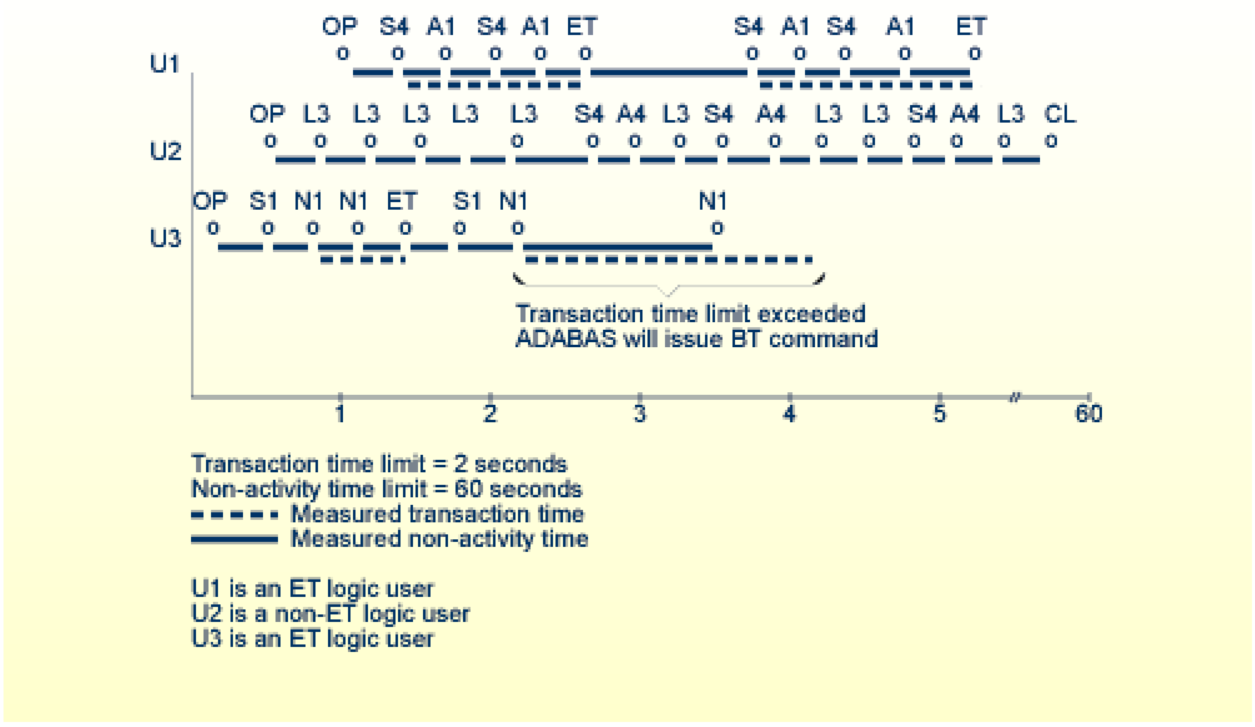
Adabas informs user 1 that his current transaction has been backed out by returning response code 9 for the next command issued by user 1. User 1 may repeat the backed–out transaction from the beginning, or may issue another transaction.

The transaction time limit applies only to programs which employ ET logic.

**Transaction Time Limit**

```
     USER 1              ADABAS                 USER 2
                     ISN 1 HELD FOR
                     USER 1
                                            READ WITH HOLD
                                            (L4) FILE 1 ISN 2
                    ISN 2 HELD FOR USER 2
READ WITH HOLD
(L4) FILE 1 ISN 2
                     USER 1 MUST WAIT*
                    (ISN 2 HELD BY USER 2)
                                            READ WITH HOLD
                                            (L4) FILE 1 ISN 1
                     USER 2 MUST WAIT*
                    (ISN 1 HELD BY USER 1)
                     (BOTH USERS WAITING)
                     USER 1 HAS EXCEEDED
                     TRANS. TIME LIMIT.
                     ADABAS ISSUES BT
                     FOR USER 1.
                     ISN 1 RELEASED.
                     USER 1 NOTIFIED THAT
                     HIS TRANSACTION HAS
                     BEEN BACKED OUT
                     (RESPONSE CODE = 9).
                     ISN 1 READ AND HELD
                     FOR USER 2.
USER 1 MAY                                  NEXT COMMAND
REISSUE TRANSACTION
OR ISSUE
ANOTHER TRANSACTION.
```

(*) If the RETURN option is used for the hold command, the user is not placed in wait status, but will receive response code 145 instead.


U1
OP S4 A1 S4 A1 ET S4 A1 S4 A1 ET
U2
OP L3 L3 L3 L3 L3 S4 A4 L3 S4 A4 L3 L3 S4 A4 L3 CL
U3
OP S1 N1 N1 ET S1 N1 N1
Transaction time limit exceeded
ADABAS will issue BT command
1 2 3 4 5 60
Transaction time limit = 2 seconds
Non-activity time limit = 60 seconds
Measured transaction time
Measured non-activity time
U1 is an ET logic user
U2 is a non-ET logic user
U3 is an ET logic user

**Transaction and Non-Activity Time Limits**

## Recovery/Restart

This section describes the Adabas command facilities related to data protection, recovery and user restart.

The two types of Adabas users which may perform database updating (ET logic and exclusive control) are presented, checkpointing procedures for each are summarized and the autobackout feature is described.

## ET Logic Users

The logical transaction commands ET, BT and RE are used to ensure a transaction restart capability. Transaction restart is the ability to begin a user session starting with the first transaction which follows the last successful transaction of the user's previous session. Users who use logical transaction commands are called ET logic users.

## Logical Transaction

A logical transaction is the smallest unit of work (as defined by the user) which must be performed in its entirety to ensure that the information contained in the database is logically consistent.

A logical transaction may consist of one or more Adabas commands, which together cause the database updating required to complete a logical unit of work. A logical transaction begins with the first command which places a record in hold status and ends when an ET (or BT or CL) command is issued.

## ET Command

The ET command must be issued at the end of each logical transaction. Successful execution of an ET command ensures that all the updates performed during the transaction will be physically applied to the database, regardless of subsequent user or Adabas session interruption.

Updates performed within transactions for which ET commands have not been successfully executed will be backed out by Adabas.

The ET command results in the release of all records held by the user during the transaction. Adabas returns a unique transaction sequence number which the user may use to identify the transaction for auditing or restart purposes.

The ET command may also be used to store user data in an Adabas system file. This data may be used for user restart purposes and may be read with an OP or RE command.

**ET Command**

```
    USER PROGRAM                         ADABAS
FIND (S4), UPDATE (A1)
                                 RECORD UPDATED IN ADABAS
                                 BUFFER BUT NOT NECESSARILY
                                 WRITTEN TO THE DATABASE
FIND (S1), HOLD ISN (HI),
UPDATE (A1)
                                 RECORD UPDATED IN ADABAS
                                 BUFFER BUT NOT NECESSARILY
                                 WRITTEN TO THE DATABASE
END TRANSACTION (ET)
                                 DATA PROTECTION INFORMATION
                                 FOR THE TRANSACTION IS WRITTEN
                                 TO THE ADABAS WORK AND LOG
FIND (S4), UPDATE (A1)
                                 RECORD UPDATED IN ADABAS
                                 BUFFER BUT NOT NECESSARILY
                                 WRITTEN TO THE DATABASE
FIND (S1), HOLD ISN (HI),
UPDATE (A1)
```

```
                                       RECORD UPDATED IN ADABAS
                                       BUFFER BUT NOT NECESSARILY
                                       WRITTEN TO THE DATABASE
      . . . ADABAS OR USER SESSION INTERRUPTION . . .
```

When the next Adabas session is started, or when the user is timed out, both the updates for transaction 1 will be physically written to the database (if they had not been previously written). The updates for transaction 2 will not be physically written (or will be backed out) because no ET command was processed for this transaction.

## BT Command

The BT command may be issued to remove all the updates which have been performed during the transaction currently being processed. This may be necessary because of a program error or the determination that the entire transaction cannot be successfully completed.

A BT command causes all the records held during the transaction to be released from hold status.

The command sequence

```
FIND (S4)
UPDATE (A1) (modify field XX to value 20)
FIND (S4)
UPDATE (A1) (modify field YY to value 50)
END TRANSACTION (ET)

FIND (S4)
UPDATE (A1) (modify field XX to value 10)
BACKOUT TRANSACTION (BT)
```

will result in the field values XX = 20 and YY = 50. The second update to field XX is removed as a result of the BT command.

Adabas automatically generates a BT command if the user transaction exceeds the transaction time limit or non–activity time limit. Adabas will return response code 9 to the user to indicate that the last transaction has been backed out. All records held during the back out transaction are released from hold status.

This backout procedure:

- Removes all the updates performed within a partially-completed transaction that was issued by a user who has terminated abnormally;
- Resolves a resource interlock between two users (see the section Resource Interlock for additional information).

If a logical transaction is backed out for an active user, the user may reissue the backed-out transaction, or issue another transaction.

BT command generated by Adabas (user still active)

```
FIND (S4)
UPDATE (A1)
FIND (S4)
UPDATE (A1)
END TRANSACTION (ET) (end transaction 1)

FIND (S4)
UPDATE (A1)
FIND (S4)
User must wait (record not available)

-------TRANSACTION TIME LIMIT EXCEEDED

BACKOUT  (BT) issued by Adabas; logical transaction
2 backed out; user receives response code 9 for  S4
command.
```

BT command generated by Adabas (user no longer active)

```
FIND (S4)
UPDATE (A1)
FIND (S4)
UPDATE (A1)

-------USER PROGRAM TERMINATES ABNORMALLY

BACKOUT  (BT)  issued  by  Adabas; both updates are
backed out and all held records are released.
```

## Autobackout

Autobackout is performed by the Adabas nucleus at the beginning of each Adabas session to remove all the updates which were performed within partially-completed transactions issued by ET logic users.

Autobackout is performed for ET logic users only.

## System-Generated Fields

Adabas files can contain fields for which the values are automatically generated by Adabas itself. These fields are called system-generated fields.

System-generated fields can have the following values:

- The time stamp of when a record was created or the last time it was updated;
- Information about the user who created a record or who performed the last update of the record.

Please refer to *Administration*, *FDT Record Structure, Definition Options, System-Generated Fields* for further information.

# 4 Calling Adabas

This chapter describes the procedures used to link application programs, and to call Adabas to execute an Adabas command.

There are two kinds of Adabas direct calls, one for each of the different control block interfaces supported by Adabas:

- The *ACB direct call interface* is the classic direct call interface, used for Adabas releases prior to Adabas Version 6.1. Direct calls in this format require the use of the classic Adabas control block (ACB). If you have been using releases of Adabas prior to Adabas Version 6.1, the direct calls used by your applications use the ACB direct call interface.
- The *ACBX direct call interface* is the extended direct call interface, used for Adabas releases starting with Adabas Version 6.1. Direct calls in this format require the use of the *extended* Adabas control block (ACBX). If you have purchased and installed Adabas Version 6.1 (or later), you can use this format of direct call in your applications. Otherwise, you cannot.

Adabas Version 6.1 (and subsequent versions) fully supports *both* the ACB and the ACBX direct call interfaces:

- Existing application programs that use the ACB 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.

The control block and the related buffers specify which Adabas command is to be executed and provide any additional information (parameters or operands) required for the command. The pointer to the appropriate control block (ACB or ACBX) must always be the first operand specified in an Adabas call.

This chapter covers the following topics:

## Linking Application Programs

All application programs have to be linked to an Adabas interface. The following interfaces are available:

| Interface name | Name of the shared library on UNIX | Name of the DLL on Windows |
|---|---|---|
| adalnkx | libadalnkx.s[ol] | adalnkx.dll |
| adalnk | libadalnk.s[ol] | adalnk32.dll |
| adalnknc | libadalnknc.s[ol] | adalnknc.dll |

The following table shows when to use which interface:

| | adalnkx | adalnk | adalnknc |
|---|---|---|---|
| Multi-thread applications | x | | |
| Single-thread applications | x | x | x |
| Net-work 2 support | | x | x |
| Net-work 7 support | x | x | |
| XA support | x | x | x |
| ACB direct call interface | x | x | x |
| ACBX direct call interface | x | | |

**Notes:**

1. Because starting with Adabas Version 5.1, both adalnkx and adalnk support XA, there is no longer a requirement for adalnknc. It is only provided for reasons of compatibility with Adabas Version 3.3.
2. You should use either adalnk or adalnkx (but not both at the same time) to link to a client application.

### Linking on UNIX Platforms

There are various ways of linking the application to the Adabas interface:

**Dynamic Link**

The application is linked with the required shared library for the Adabas interface. LD_LIBRARY_PATH must contain $ACLDIR/lib so that the shared library can be found at runtime.

If you have set the environment as suggested in the Adabas installation, the library path contains $ACLDIR/lib, where the shared library can be found at runtime.

**Loading via stub**

The application is linked with the Adabas stub object adabasx.o (for loading adalnkx) or adabas.o (for loading adalnk) in the directory $ACLDIR/lib. Then at runtime the $ADALNK environment variable has to point to the shared library to be used. When linking with the stub adabasx.o, the environment variable $ADALNKX has to point to the shared library libadalnkx.so. LD_LIBRARY_PATH must contain $ACLDIR/lib because the Adabas interface calls other shared libraries from this directory.

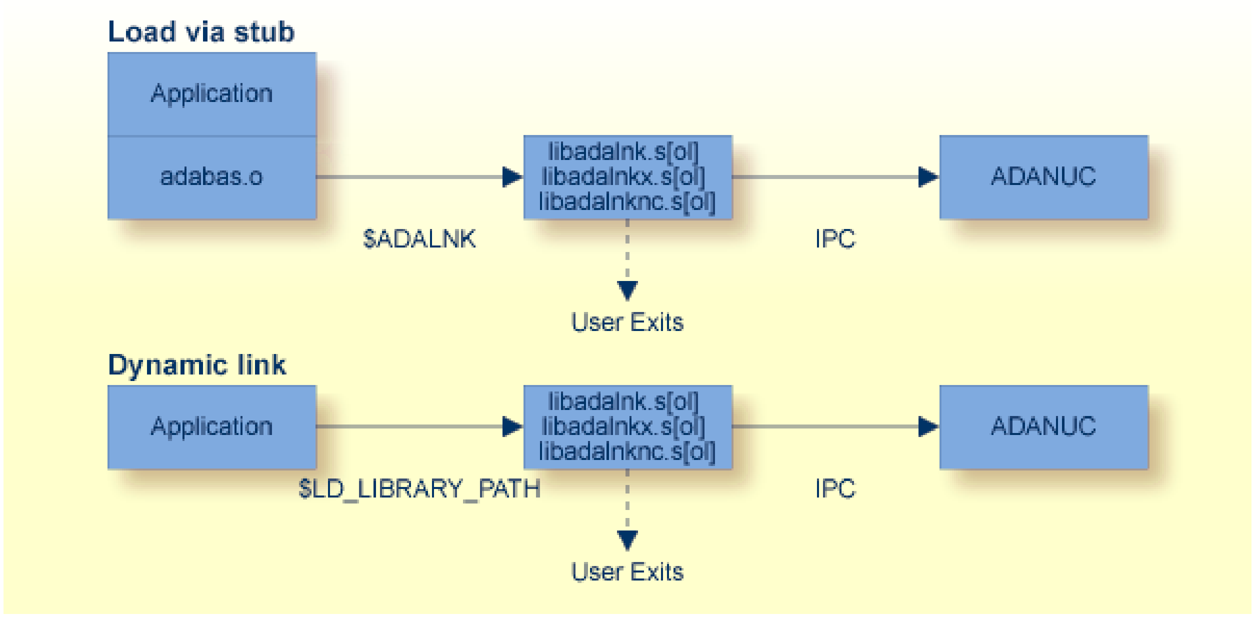
**Loading via dlopen**

You can also load the Adabas interface via dlopen. It is not necessary to specify the full path in the dlopen call if you have set the environment as suggested in the Adabas installation; then the library path contains $ACLDIR/lib where the Adabas interface is stored.

**Note:** The Adabas interface may only be loaded once. Closing the Adabas interface with dlclose and reloading it can cause errors.

**Note:** If you want to use an Adabas interface when an application has the S-bit set, the installation root directory must be /opt/softwareag. Then /opt/softwareag/AdabasClient/lib contains the current Adabas interfaces, which can be used to load the Adabas interfaces from applications with S-bits.

**Important:** adalnk/adalnkx contain a signal handler which, for example, displays a stack backtrace after a signal SIGUSR1. This signal handler can be deactivated by setting the environment variable SMP_SIGNAL_HANDLING to 0. This is necessary if your application has its own signal handling, because otherwise signals might no longer be processed as expected by the application's signal handler because of conflicts with the adalnk/adalnkx signal handler. For example, Natural uses this signal for printing. If you use the environment variable settings provided by the Adabas installation, they already include the setting of SMP_SIGNAL_HANDLING to 0.



### Linking 32 Bit-Mode Applications on 64 Bit UNIX Platforms

Unlike Adabas Version 3, where a different interface (adalnk32) was provided for linking 32 bit-mode applications on 64 bit UNIX platforms, starting with Adabas Version 5.1, Adabas provides the same interfaces as described above in a 32 bit-mode version in a separate directory. Starting with Adabas Version 6.1, the interfaces for the 32 bit-mode can be found in $ACLDIR/$ACLVERS/lib_32.

**Note:** While the installation provides the setting of the environment variable $LD_LIBRARY_PATH for the 64 bit-mode, it does not for the 32 bit-mode.Users running Adabas applications in 32 bit-mode on 64-bit platforms must take care to ensure that they set the $LD_LIBRARY_PATH correctly before they start the applications.

## Linking on Windows

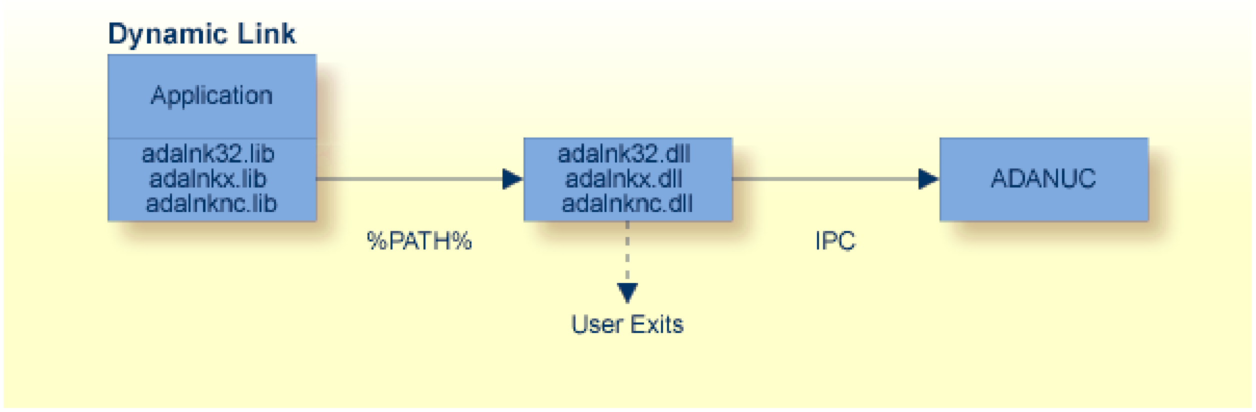
### Dynamic Link

Link with adankx.lib, adalnk32.lib or adalnknc.lib. This way, at runtime, the corresponding DLL is dynamically linked from %PATH%. If you haven't modified %PATH% since installing Adabas, the DLLs are loaded from %ACLDIR% (32-bit mode) or %ACL64DIR% (64-bit mode).

**Notes:**

1. In 64-bit mode, only the interface adalnkx is supported.
2. The 64-bit mode interface is only available for those platforms on which Adabas is installed in 64-bit mode.

For further information, such as compile and link options, see the example makefile un-der%ACLDIR%\..\examples\client and the sample C code in the same directory.



### Loading via LoadLibrary or LoadLibraryEx

It is also possible to use LoadLibrary or LoadLibraryEx to load the Adabas interface. It is recommended not to specify the full path for the DLL, since the folder where the correct Adabas interface is stored depends on the installed Adabas versions and the Windows operating system and the mode (32 or 64 bit); the Adabas installation sets the PATH in such a way, that the correct Adabas interface is found.

**Note:** The Adabas interface may only be loaded once. Freeing the Adabas interface with FreeLibrary or FreeLibraryAndExitThread and reloading it may cause errors.

## Specifying an ACB Interface Direct Call

When making a direct call using the ACB interface, syntax such as the following should be used (this is a COBOL example):

```
CALL 'ADABAS' USING acb-control-block-name
                    [format-buffer]
                    [record-buffer]
                    [search-buffer]
                    [value-buffer]
                    [ISN-buffer]
```

In an ACB direct call, Adabas expects buffers to be specified in the order shown in this syntax. If no buffers are required for a call, no buffers need be specified. However, if a given call does not require a format buffer, but does require one of the other buffers (for example, a record buffer), a dummy (or blank) format buffer must be specified prior to the record buffer. Likewise, if a call requires only an ISN buffer, dummy format, record, search, and value buffers must be supplied as well.

The following table describes each of the italicized, replaceable items in this syntax. For more information about the format of the ACB control block and Adabas buffers, read *Adabas Control Block (ACB)* and *Defining Buffers*. For information about the relationships between different ABD types, read *Understanding the Different Buffer Types*.

| Replace | With |
|---|---|
| *acb-control-block-name* | The pointer to the Adabas Control Block (ACB) to use for the call. |
| *format-buffer* | The name of or pointer to the format buffer to use for the call. Only one format buffer can be specified in a single ACB direct call. |
| *ISN-buffer* | The name of or pointer to the ISN buffer to use for the call. Only one ISN buffer can be specified in a single ACB direct call. |
| *record-buffer* | The name of or pointer to the record buffer to use for the call. Only one record buffer can be specified in a single ACB direct call. |
| *search-buffer* | The name of or pointer to the search buffer to use for the call. Only one search buffer can be specified in a single ACB direct call. |
| *value-buffer* | The name of or pointer to the value buffer to use for the call. Only one value buffer can be specified in a single ACB direct call. |

## Specifying an ACBX Interface Direct Call

The way direct calls are made in your applications when using the new ACBX interface is different than when using the classic ACB interface. In addition, the calls are different for mainframe applications and open systems applications. This section covers the following topics:

### C-Interface for ACBX Interface Direct Calls

The way direct calls are made in your applications when using the new ACBX interface is different than when using the classic ACB interface. When making a direct call using the ACBX interface in open system applications, syntax such as the following should be used (this is a C example):

```
adabasx(ACBX_pointer, ABD-count, ABD-list-pointer)
```

Each ABD either directly precedes its associated buffer or contains a pointer to the buffer. It effectively represents the buffer.

ABDs can be specified in any sequence in an ACBX interface direct call. However, if an ABD requires a matching ABD of another type, Adabas will match them sequentially. For example, if three format buffer ABDs and three record buffer ABDs are included in the call, the first format buffer ABD in the call is matched with the first record buffer ABD in the call, the second format buffer ABD is matched with the second record buffer ABD, and the third format buffer ABD is matched with third record buffer ABD.

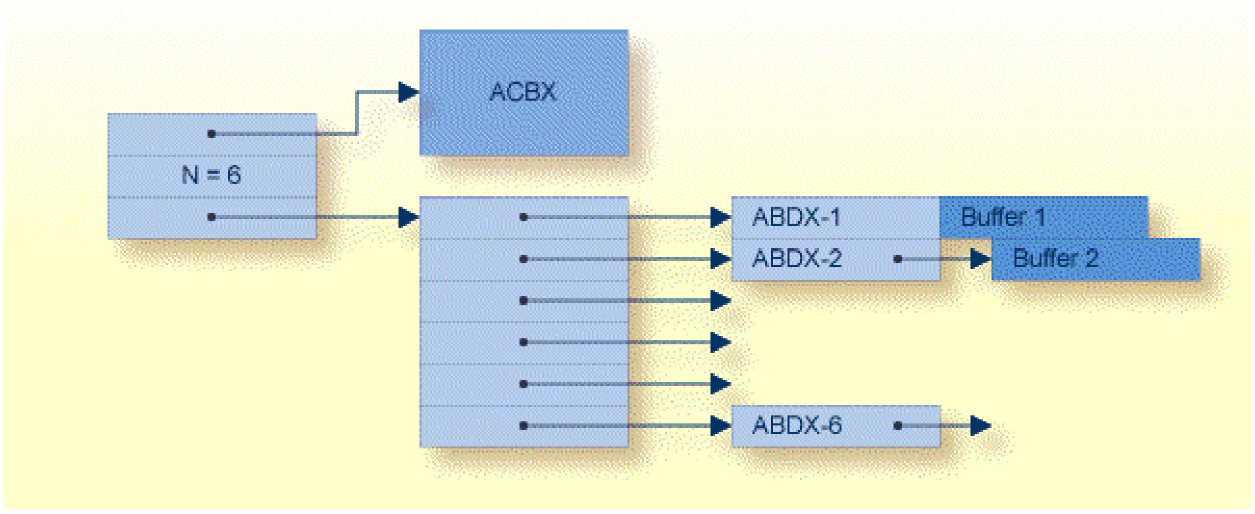
If unequal numbers of match-requiring ABDs are specified, Adabas will generate a dummy ABD (with a buffer length of zero) for the missing ABD. For example, if three format buffer ABDs are specified, but only two record buffer ABDs are specified, a dummy record buffer ABD is created for use with the third format buffer ABD. If you would prefer that the dummy record buffer ABD be used for the second format buffer ABD instead, you must specify the dummy record buffer ABD yourself prior to the record buffer ABD to be used by the third format buffer ABD.

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. This applies to only a few commands; the most prominent of them is OP.

The following table describes each of the italicized, replaceable items in this syntax. For more information about the format of the extended Adabas control block (ACBX), Adabas buffer descriptions (ABDs), and Adabas buffers, read *Extended Adabas Control Block (ACBX)*, *Adabas Buffer Descriptions (ABDs)*, and *Defining Buffers*. For information about the relationships between different buffer types, read *Understanding the Different Buffer Types*.

| Parameter | Description |
|---|---|
| *ACBX_pointer* | The pointer to the extended Adabas control block (ACBX) to use for the call. |
| *ABD-count* | The number of the ABD pointers included in the ABD list for the direct call. |
| *ABD-list-pointer* | The pointer to the ABD list for the direct call. The ABD list contains pointer references for all of the ABDs used by the ACBX direct call. For more information about the ABD list, read *ABD Lists*. |

These adabasx parameters are shown in the following graphic:



**Important:** The ACBX and the ABDs must be initialized before the first Adabas call by using the calls SETACBX(ACBX_pointer) and SETABD(ABD_pointer) respectively.

### Mainframe Interface for ACBX Interface Direct Calls

The way direct calls are made in your applications when using the new ACBX interface is different than when using the classic ACB interface. When making a direct call using the ACBX interface in mainframe applications, syntax such as the following should be used (this is a COBOL example):

```
CALL 'ADABAS' USING acbx-control-block-name
                    reserved-fullword
                    reentrancy-token
                  [format-buffer-ABD record-buffer-ABD [multifetch-buffer-ABD]]...
                    [search-buffer-ABD]
                    [value-buffer-ABD]
                    [ISN-buffer-ABD]
                    [performance-buffer-ABD]
                    [user-buffer-ABD]
```

Each ABD either directly precedes its associated buffer or contains a pointer to the buffer. It effectively represents the buffer.

ABDs can be specified in any sequence in an ACBX interface direct call. However, if an ABD requires a matching ABD of another type, Adabas will match them sequentially. For example, if three format buffer ABDs and three record buffer ABDs are included in the call, the first format buffer ABD in the call is matched with the first record buffer ABD in the call, the second format buffer ABD is matched with the second record buffer ABD, and the third format buffer ABD is matched with third record buffer ABD.

If unequal numbers of match-requiring ABDs are specified, Adabas will generate a dummy ABD (with a buffer length of zero) for the missing ABD. For example, if three format buffer ABDs are specified, but only two record buffer ABDs are specified, a dummy record buffer ABD is created for use with the third format buffer ABD. If you would prefer that the dummy record buffer ABD be used for the second format buffer ABD instead, you must specify the dummy record buffer ABD yourself prior to the record buffer ABD to be used by the third format buffer ABD.

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. This applies to only a few commands; the most prominent of them is OP.

The following table describes each of the italicized, replaceable items in this syntax. For more information about the format of the extended Adabas control block (ACBX), Adabas buffer descriptions (ABDs), and Adabas buffers, read *Extended Adabas Control Block (ACBX)*, *Adabas Buffer Descriptions (ABDs)*, and *Defining Buffers*. For information about the relationships between different buffer types, read *Understanding the Different Buffer Types*.

| Replace | With |
|---|---|
| *acbx-control-block-name* | The pointer to the extended Adabas control block (ACBX) to use for the call. |
| *format-buffer-ABD* | The name of or pointer to the format buffer ABD that defines a format buffer segment to use for the call. Each format buffer segment must end with a period and be a complete and valid standalone format buffer. Multiple format buffer ABDs can be specified in a single ACBX direct call. |
| *ISN-buffer-ABD* | The name of or pointer to the ISN buffer ABD that defines an ISN buffer segment to use for the call. Only one ISN buffer ABD can be specified in a single ACBX direct call. |
| *multifetch-buffer-ABD* | The name of or pointer to the multifetch buffer ABD that defines a multifetch buffer segment to use for the call. Multiple multifetch buffer ABDs can be specified in a single ACBX direct call. Currently not supported by Adabas on open systems. |
| *performance-buffer-ABD* | The name of or pointer to the performance buffer ABD that defines a performance buffer segment. Currently not supported by Adabas on open systems. |
| *record-buffer-ABD* | The name of or pointer to the record buffer ABD that defines a record buffer segment to use for the call. Multiple record buffer ABDs can be specified in a single ACBX direct call. |
| *reentrancy-token* | The ADALNK reentrancy token. This is a fullword in the calling program's storage where ADALNK stores the address of its static data area. This |

| Replace | With |
|---|---|
| | fullword should be set to zero before the first Adabas call. It should then remain unchanged for all subsequent direct calls while the program runs. |
| *reserved-fullword* | The fullword containing binary zeros. This fullword is reserved for use by Adabas and should be set to binary zeros before the first Adabas call. |
| *search-buffer-ABD* | The name of or pointer to the search buffer ABD that defines a search buffer segment to use for the call. Only one search buffer ABD can be specified in a single ACBX direct call. |
| *user-buffer-ABD* | The name of or pointer to the user buffer ABD that defines a user buffer segment (extension) to use for the call. A single user buffer ABD can be specified in an ACBX direct call. |
| *value-buffer-ABD* | The name of or pointer to the value buffer ABD that defines a value buffer segment to use for the call. Only one value buffer ABD can be specified in a single ACBX direct call. |

**How Adabas Distinguishes Between ACB and ACBX Direct Calls**

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 determines which control block is used for a call by the presence of a value starting 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 starting 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 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.

## Mixing ACB and ACBX Direct Calls

You can freely mix ACB and ACBX direct calls in the same application.

## Adabas Control Block Structures (ACB and ACBX)

Two kinds of control blocks are now supported by Adabas:

- The Adabas control block (ACB) is the classic control block, used for Adabas releases prior to Adabas Version 6.1. If you have been using releases of Adabas prior to Adabas Version 6.1, the direct calls used by your applications use the ACB. It is important to note that Adabas Version 6.1 (and later) fully supports the ACB, so you are not required to update your existing applications once you install Adabas Version 6.1 (or later).
- The extended Adabas control block (ACBX) can be used in Adabas releases starting with Adabas Version 6.1. The ACBX supports the increased buffer sizes and segmented buffers introduced in Adabas 6.1. If you have purchased and installed Adabas Version 6.1 (or later), you can use the ACBX in direct calls from your applications. Otherwise, you cannot.

To ensure user program compatibility with later Adabas releases, all control block fields not used by a particular command should be set to zeros or blanks, depending on field type.

The position of each field in a control block is fixed. In addition, all values in the control block must be entered in the data type defined for the field. For example, the ISN field is defined as binary format; therefore, any entry made in this field must be in binary format.

**Notes:**

1. Adabas and other Software AG program products use some control block fields for internal purposes, and may return values in some fields that have no meaning to the user. These uses and values may be release-dependent, and are not appropriate for program use. Software AG therefore recommends that you use only the fields and values described in this documentation. *In addition, you should always initialize unused control block fields with either zeros or blanks, according to their field types.*
2. Some Adabas-dependent Software AG products return control block values such as response codes and subcodes. Refer to the documentation for those products for a description of the product-specific control block values.

This section covers the following topics:

- Adabas Control Block (ACB)
- Extended Adabas Control Block (ACBX)

- Differences between the ACB and the ACBX

## Adabas Control Block (ACB)

The Adabas control block (ACB) is 80 bytes long. This section covers the following topics:

- ACB Format
- ACB Fields

### ACB Format

The following table describes the format of the ACB. We recommend that you set unused ACB fields to binary zeros before the direct call is initiated.

| Name | Field | Control Block Position | Offset | Length (in Bytes) | Format |
|---|---|---|---|---|---|
| ACBTYPE | Call Type | 1 | 00 | 1 | binary |
| reserved | (reserved) | 2 | 01 | 1 | binary |
| ACBCMD | Command Code | 3-4 | 02 | 2 | alphanumeric |
| ACBCID | Command ID | 5-8 | 04 | 4 | binary |
| ACBFNR | File Number | 9-10 | 08 | 2 | binary |
| ACBRSP | Response Code | 11-12 | 0A | 2 | binary |
| ACBISN | ISN | 13-16 | 0C | 4 | binary |
| ACBISL | ISN Lower Limit | 17-20 | 10 | 4 | binary |
| ACBISQ | ISN Quantity | 21-24 | 14 | 4 | binary |
| ACBFBL | Format Buffer Length | 25-26 | 18 | 2 | binary |
| ACBRBL | Record Buffer Length | 27-28 | 1A | 2 | binary |
| ACBSBL | Search Buffer Length | 29-30 | 1C | 2 | binary |
| ACBVBL | Value Buffer Length | 31-32 | 1E | 2 | binary |
| ACBIBL | ISN Buffer Length | 33-34 | 20 | 2 | binary |
| ACBCOP1 | Command Option 1 | 35 | 22 | 1 | alphanumeric |
| ACBCOP2 | Command Option 2 | 36 | 23 | 1 | alphanumeric |
| ACBADD1 | Additions 1 | 37-44 | 24 | 8 | alphanumeric / binary |
| ACBADD2 | Additions 2 | 45-48 | 2C | 4 | alphanumeric / binary |
| ACBADD3 | Additions 3 | 49-56 | 30 | 8 | alphanumeric |
| ACBADD4 | Additions 4 | 57-64 | 38 | 8 | alphanumeric |
| ACBADD5 | Additions 5 | 65-72 | 40 | 8 | alphanumeric / binary |
| ACBCMDT | Command Time | 73-76 | 48 | 4 | binary |
| ACBUSER | User Area | 77-80 | 4C | 4 | not applicable |

### ACB Fields

The content of the control block fields and buffers must be set before an Adabas command (call) is issued. Adabas also returns one or more values or codes in certain fields and buffers after each command is executed.

We recommend that you set unused ACB fields to binary zeros before the direct call is initiated.

Each of the fields in the ACB is described in this section, in the order they appear in the **ACB format**. The descriptions are valid for most Adabas commands; however, some Adabas commands use some control block fields for purposes other than those described here. For complete information about how these fields are used by each Adabas command, read *Commands*.

#### Call Type (ACBTYPE)

The first byte of the Adabas control block (ADACB) is used by the Adabas API to determine the processing to be performed.

The values for logical requests are:

| Hex | Indicates ... |
|---|---|
| 00 | a 1-byte file number (file numbers between 1 and 255) or DBID. |
| 30 | a 2-byte file number (file numbers between 1 and 65535) or DBID. |

All other values in the first byte of the ADACB are reserved for use by Software AG.

Because an application can reset the value in the first byte of the ADACB on each call, it is possible to mix both one- and two-byte file number (DBID) requests in a single application. In this case, you must ensure the proper construction of the file number (ACBFNR) and response code (ACBRSP) fields in the ADACB for each call type. See the discussions of these fields for more information.

Software AG recommends that an application written to use two-byte file numbers always places hex 30 in the first byte of the ADACB, the logical database ID in the ACBRSP field, and the file number in the ACBFNR field. The application can then treat both the database ID and file number as 2-byte binary integers, regardless of the value for the file number in use.

Applications written in Software AG's Natural language need not include this first byte of the Adabas ACB because Natural supplies appropriate values.

### Command Code (ACBCMD)

The command code defines the command to be executed, and comprises two alphanumeric characters (for example, OP, A1, BT).

### Command ID (ACBCID)

The command ID field is used by many Adabas commands to identify logical read sequences, search results, and (optionally) decoded format buffers for use by subsequent commands. You can specify alphanumeric or binary command IDs as you choose or you can request the generation of new binary command IDs by Adabas. See the section *Using Command IDs* for more information about command IDs. For ET, CL, and some OP commands, Adabas returns a binary transaction sequence number in the command ID field.

**Note:** Internally, the command ID field is treated as binary, even though alphanumeric values are often stored in this field. For cross-platform calls this means that there is no EBCDIC - ASCII conversion, and that the bytes are swapped if the integer arithmetic is different on the client and nucleus platforms. However, for those users who specify alphanumeric values, a 4 byte blank value is also considered to be an empty command ID, like a binary 0 value. This is valid for both ASCII and EBCDIC blanks; in the following, both ASCII blanks (0x20202020) and EBCDIC blanks (0x40404040) are considered as blank ID values.

### File Number (ACBFNR)

The file number may be one or two bytes.

For an application program issuing Adabas commands for file numbers between 1 and 255 (single byte), build the control block as follows:

| Position | Action |
|---|---|
| 1 | Place hex 00 in the first byte of the ADACB. |
| 9 | Place the file number in the low-order byte of the ACBFNR field of the ADACB. The high-order byte of the ACBFNR field is used to store the logical (database) ID or number. |

Adabas permits the use of file numbers greater than 255 on logical requests. For an application program issuing Adabas commands for file numbers between 256 and 5000 (two-byte), build the control block as follows:

| Position | Action |
|---|---|
| 1 | Place hex 30 in the first byte of the ADACB. |
| 9 | Use both bytes in ACBFNR for the file number, and use the two bytes in ACBRSP for the database (logical) ID. |

### Response Code (ACBRSP)

The response code field is used for two-byte database IDs.

It is also always set to a value when the Adabas command is completed. Successful completion is normally indicated by a response code of zero. For repeatable commands that process sequences of records or ISNs, other response codes indicate end-of-file or end-of-ISN-list. Non-zero response codes are defined in the *Adabas Messages and Codes*.

### ISN (ACBISN)

The ISN field both specifies a required four-byte Adabas ISN value required by the command and, where appropriate, returns either the first ISN of a command-generated ISN list, or an ISN of the record read by the command.

### ISN Lower Limit (ACBISL)

ISN lower limit specifies the starting point in an ISN list or range where processing is to begin. For OP commands, an optional user-specific non-activity timeout value can be specified in this field. An OP command also returns Adabas release information in this field (see also the Additions 5 field description).

### ISN Quantity (ACBISQ)

The ISN quantity is a count of ISNs returned by a command. The count can be a total of all ISNs in an ISN list, or the total ISNs entered into the ISN buffer from a larger pool of ISNs by this operation. The OP command uses this field to specify an optional user-specific transaction time limit; it returns system and call type information flags in the ISN quantity field (see also the Additions 5 field description).

### Buffer Length: Format, Record, Search, Value, and ISN (ACBFBL, ACBRBL, ACBSBL, ACBVBL, and ACBIBL)

The format, record, search, value, and ISN buffer length fields specify the size of the related buffers. A buffer's size usually remains the same throughout a transaction. In some ISN-related operations, the ISN buffer size value determines how a command processes ISNs; for example, specifying a zero ISN buffer length causes some commands to store a resulting ISN list in the Adabas work area. If a buffer is not needed for an Adabas command, the corresponding length value should be set to zero. In some cases (multifetch option, as an example), there is a limit on the length of the buffer; see the specific command descriptions for more information.

### Command Option 1 and Command Option 2 (ACBCOP1 and ACBCOP2)

The Command Option 1 and 2 fields allow you to specify processing options (ISN hold, command-level prefetching control, returning of ISNs, and so on).

### Additions 1 (ACBADD1)

The Additions 1 field sometimes requires miscellaneous command-related parameters such as qualifying descriptors for creating ISN lists.

### Additions 2 (ACBADD2)

The Additions 2 field returns compressed record length in the last two bytes and decompressed length of record buffer-selected fields in the first two bytes for all An, Ln, Nn, and S1/2/4 commands. OP (open) and RE (read ET data) commands return transaction sequence numbers in this field. If Entire Net-work is installed, some response codes return the node ID of the "problem" node in the last two bytes of the Additions 2 field.

If a command results in a nucleus response code, the addition 2 field's first two bytes (47 and 48) can contain a hexadecimal subcode to identify the cause of the response code. Response codes and their subcodes (as decimal equivalents) are described in the *Adabas Messages and Codes*.

### Additions 3 (ACBADD3)

The Additions 3 field is for providing a user`s password for accessing password-protected files. If the file containing the field is actually password-protected, the password in this field is replaced with spaces (blanks) during command execution before Adabas returns control to the user program.

### Additions 4 (ACBADD4)

On Mainframes, the Additions 4 field must be set to the cipher code of a file, if a command reads or writes records of an encrypted (ciphered) Adabas file. On UNIX and Windows platforms this field is not used.

### Additions 5 (ACBADD5)

Instead of using the command ID as a format buffer ID, Additions 5 can be used to store the format buffer ID for a command separately. Please refer to the section "Using Format Buffer IDs" for further information.

#### Command Time (ACBCMDT)

The command time field is used by Adabas to return the elapsed time that was needed by the nucleus to process the command. In contrast to the mainframe, where this field is always filled by Adabas, it is only filled on open systems platforms if Command Logging is switched on or if the nucleus is started with the environment variable ADA_CMD_TIME set (the value is irrelevant).

#### User Area (ACBUSER)

The user area field is reserved for use by the user program. When making logical user calls, the user area is neither written nor read by Adabas.

For compatibility with future Adabas releases, Software AG recommends that you set unused control block fields to null values corresponding to the field's data type.

## Extended Adabas Control Block (ACBX)

The extended Adabas control block, the ACBX, supports the increase in the buffer sizes in Adabas commands. It is 192 bytes in length (versus the 80 bytes used by the ACB). 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 Version 6.1 (or later), 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 introduced with Adabas Version 6.1.

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

- ACBX Format
- ACBX Fields

### ACBX Format

The following table describes the format of the ACBX. We recommend that you set unused ACBX fields to binary zeros before the direct call is initiated.

| Name | Field | Control Block Position | Offset | Length (in bytes) | Format |
|---|---|---|---|---|---|
| ACBXTYP | Call Type | 1 | 00 | 1 | binary |
| ACBXRSV1 | Reserved 1 | 2 | 01 | 1 | binary |
| ACBXVER | Version Indicator | 3-4 | 02 | 2 | binary |
| ACBXLEN | ACBX Length | 5-6 | 04 | 2 | binary |
| ACBXCMD | Command Code | 7-8 | 06 | 2 | alphanumeric |

| Name | Field | Control Block Position | Offset | Length (in bytes) | Format |
|---|---|---|---|---|---|
| ACBXRSV2 | Reserved 2 | 9-10 | 08 | 2 | binary |
| ACBXRSP | Response Code | 11-12 | 0A | 2 | binary |
| ACBXCID | Command ID | 13-16 | 0C | 4 | binary |
| ACBXDBID | Database ID | 17-20 | 10 | 4 | numeric |
| ACBXFNR | File Number | 21-24 | 14 | 4 | numeric |
| ACBXISN | ISN | 25-32 | 18 | 8 | binary |
| ACBXISL | ISN Lower Limit | 33-40 | 20 | 8 | binary |
| ACBXISQ | ISN Quantity | 41-48 | 28 | 8 | binary |
| ACBXCOP1 | Command Option 1 | 49 | 30 | 1 | alphanumeric |
| ACBXCOP2 | Command Option 2 | 50 | 31 | 1 | alphanumeric |
| ACBXCOP3 | Command Option 3 | 51 | 32 | 1 | alphanumeric |
| ACBXCOP4 | Command Option 4 | 52 | 33 | 1 | alphanumeric |
| ACBXCOP5 | Command Option 5 | 53 | 34 | 1 | alphanumeric |
| ACBXCOP6 | Command Option 6 | 54 | 35 | 1 | alphanumeric |
| ACBXCOP7 | Command Option 7 | 55 | 36 | 1 | alphanumeric |
| ACBXCOP8 | Command Option 8 | 56 | 37 | 1 | alphanumeric |
| ACBXADD1 | Additions 1 | 57-64 | 38 | 8 | alphanumeric/ binary |
| ACBXADD2 | Additions 2 | 65-68 | 40 | 4 | binary |
| ACBXADD3 | Additions 3 | 69-76 | 44 | 8 | alphanumeric/ binary |
| ACBXADD4 | Additions 4 | 77-84 | 4C | 8 | alphanumeric |
| ACBXADD5 | Additions 5 | 85-92 | 54 | 8 | alphanumeric/ binary |
| ACBXADD6 | Additions 6 | 93-100 | 5C | 8 | alphanumeric/ binary |
| ACBXRSV3 | Reserved 3 | 101-104 | 64 | 4 | binary |
| ACBXERRA | Error Offset in Buffer | 105-112 | 68 | 8 | binary |
| ACBXERRB | Error Character Field | 113-114 | 70 | 2 | alphanumeric |
| ACBXERRC | Error Subcode | 115-116 | 72 | 2 | binary |
| ACBXERRD | Error Buffer ID | 117 | 74 | 1 | alphanumeric |
| ACBXERRE | Reserved | 118 | 75 | 1 | binary |
| ACBXERRF | Error Buffer Sequence Number | 119-120 | 76 | 2 | binary |
| ACBXSUBR | Subcomponent Response Code | 121-122 | 78 | 2 | binary |
| ACBXSUBS | Subcomponent Response Subcode | 123-124 | 7A | 2 | binary |
| ACBXSUBT | Subcomponent Error Text | 125-128 | 7C | 4 | alphanumeric |
| ACBXLCMP | Compressed Record Length | 129-136 | 80 | 8 | binary |
| ACBXLDEC | Decompressed Record Length | 137-144 | 88 | 8 | binary |
| ACBXCMDT | Command Time | 145-152 | 90 | 8 | binary |

| Name | Field | Control Block Position | Offset | Length (in bytes) | Format |
|---|---|---|---|---|---|
| ACBXUSER | User Area | 153-168 | 98 | 16 | not applicable |
| ACBXRSV4 | Reserved 4 | 169-193 | A8 | 24 | *do not touch* |

## ACBX Fields

The content of the control block fields and buffers must be set before an Adabas command (call) is issued. Adabas also returns one or more values or codes in certain fields and buffers after each command is executed.

We recommend that you set unused ACBX fields to binary zeros before the direct call is initiated.

Each of the fields in the ACBX is described in this section, in the order they appear in the ACBX format. The descriptions are valid for most Adabas commands; however, some Adabas commands use some control block fields for purposes other than those described here.

### Call Type (ACBXTYP)

The first byte of the Adabas control block (ADACBX) is used by the Adabas API to determine the processing to be performed.

When issuing an Adabas command, set this field to binary zeros. This indicates that a logical user call is being made (ACBXUSER equate).

Applications written in Software AG's Natural language need not include this first byte of the Adabas ACBX because Natural supplies appropriate values.

### Reserved 1 (ACBXRSV1)

This field is reserved. Set this field to zero.

### Version Indicator (ACBXVER)

The version indicator identifies whether the Adabas control block uses the new ACBX or the classic ACB format. If this field is set to a value starting with the letter "F" (for example "F2"), Adabas treats the Adabas control block as though it is specified in the ACBX format. If this field is set to any other value, Adabas treats the control block as though it is specified in the classic ACB format.

### ACBX Length (ACBXLEN)

The ACBX length field should be set to the length of the ACBX structure passed to Adabas (the ACBXQLL equate, currently 192).

### Command Code (ACBXCMD)

The command code defines the command to be executed, and comprises two alphanumeric characters (for example, OP, A1, BT).

### Reserved 2 (ACBXRSV2)

This field is reserved. Set this field to zero.

### Response Code (ACBXRSP)

This field gets set to a value when the Adabas command is completed. Successful completion is normally indicated by a response code of zero. For repeatable commands that process sequences of records or ISNs, other response codes indicate end-of-file or end-of-ISN-list. Non-zero response codes are defined in the Adabas Messages and Codes documentation.

### Command ID (ACBXCID)

The command ID field is used by many Adabas commands to identify logical read sequences, search results, and (optionally) decoded format buffers for use by subsequent commands. You can specify alphanumeric or binary command IDs as you choose or you can request the generation of new binary command IDs by Adabas. See the section *Using Command IDs* for more information about command IDs. For ET, CL and some OP commands, Adabas returns a binary transaction sequence number in the command ID field.

**Note:** Internally, the command ID field is treated as binary, even though alphanumeric values are often stored in this field. For cross-platform calls this means that there is no EBCDIC - ASCII conversion, and that the bytes are swapped if the integer arithmetic is different on the client and nucleus platforms. However, for those users who specify alphanumeric values, a 4 byte blank value is also considered to be an empty command ID, like a binary 0 value. This is valid for both ASCII and EBCDIC blanks; in the following, both ASCII blanks (0x20202020) and EBCDIC blanks (0x40404040) are considered as blank ID values.

### Database ID (ACBXDBID)

Use this field to specify the database ID. The Adabas call will be directed to this database.

This field is a four-byte binary field, but at this time only two-byte database IDs are supported. Therefore, the database ID should be specified in the low-order part of the field, with leading binary zeros.

### File Number (ACBXFNR)

Use this field to specify the number of the file to which the Adabas call should be directed.

This field is a four-byte binary field, but the file number should be specified in the low-order part of the field, with leading binary zeros.

### ISN (ACBXISN)

The ISN field specifies any required Adabas ISN value required by the command and, where appropriate, returns either the ISN of the record read by the command , or the first ISN of an ISN list generated by the command.

The ACBXISN field is an eight-byte field, which is not yet used, but only 4-byte values are allowed. The high-order part of the ACBXISN field must contain binary zeros.

### ISN Lower Limit (ACBXISL)

The ISN Lower Limit field specifies the starting point in an ISN list or range where processing is to begin.

For OP commands, an optional user-specific non-activity timeout value can be specified in this field. The OP command also returns Adabas release information in this field.

The ACBXISL field is an eight-byte field, which is not yet used, but only 4-byte values are allowed. The high-order part of the ACBXISL field must contain binary zeros.

### ISN Quantity (ACBXISQ)

The ISN Quantity field is the count of ISNs returned by a search (S*x*) command. The count can be a total of all ISNs in an ISN list, or the total ISNs entered into the ISN buffer segment from a larger pool of ISNs by this operation.

For an OP command, an optional user-specific transaction time limit may be specified in this field. The OP command returns system and call type information in this field.

The ACBXISQ field is an eight-byte field, which is not yet used, but only 4-byte values are allowed. The high-order part of the ACBXISQ field must contain binary zeros.

### Command Options 1 through 8 (ACBXCOP1 through ACBXCOP8)

The Command Option 1 - 8 fields allow you to specify processing options (ISN hold, command-level prefetching control, returning of ISNs, and so on). In Adabas Version 6.1, only the Command Option 1 and Command Option 2 field are supported. However, the other Command Option fields are provided for potential expansion in future Adabas releases.

### Additions 1 (ACBXADD1)

The Additions 1 field sometimes requires miscellaneous command-related parameters such as qualifying descriptors for creating ISN lists, or the second file number of a coupled file pair.

### Additions 2 (ACBXADD2)

OP (open) and RE (read ET data) commands return transaction sequence numbers in this field.

The other values for Additions 2, as described under Additions 2 in the ACB, are also provided, but the ACBX contains other fields for these values; it is recommended that you use the new fields if you want to access these values with the ACBX interface.

### Additions 3 (ACBXADD3)

The Additions 3 field is for providing a user's password for accessing password-protected files. This field is always reset to blanks during command execution.

### Additions 4 (ACBXADD4)

On Mainframes, the Additions 4 field must be set to the cipher code of a file, if a command reads or writes records of an encrypted (ciphered) Adabas file. On UNIX and Windows platforms this field is not used.

### Additions 5 (ACBXADD5)

Instead of using the command ID as a format buffer ID, Additions 5 can be used to store the format buffer ID for a command separately. Please refer to the section "Using Format Buffer IDs" for further information.

### Additions 6 (ACBXADD6)

This field is not used at this time. It must be set to binary zeros.

### Reserved 3 (ACBXRSV3)

This field is reserved. This field must be set to binary zeros.

### Error Offset in Buffer (ACBXERRA)

The Error Offset in Buffer specifies the offset in the buffer, if any, where the error is detected during the direct call.

The ACBXERRx fields are only set when a response code is returned from a direct call. The ACBXERRA, ACBXERRD, and ACBXERRE fields are only set when the response code is related to buffer processing.

### Error Character Field (ACBXERRB)

This field identifies the two-byte Adabas short name of the field, if any, that was being processed when the error was detected.

The ACBXERRx fields are only set when a response code is returned from a direct call.

### Error Subcode (ACBXERRC)

This field stores the subcode of the error that occurred during direct call processing.

The ACBXERRx fields are only set when a response code is returned from a direct call. If Entire Net-work is installed, some response codes return the node ID of the problem node in this field.

### Error Buffer ID (ACBXERRD)

This field contains the ID (from the ABDID field) of the buffer referred to by the ACBXERRA field, so that the buffer causing the error can be identified, when multiple buffers are involved.

The ACBXERRx fields are only set when a response code is returned from a direct call. The ACBXERRA, ACBXERRD, and ACBXERRE fields are only set when the response code is related to buffer processing.

### Error Buffer Sequence Number (ACBXERRF)

This field contains the sequence number of the buffer referred to by the ACBXERRA and ACBX-ERRD fields.

The ACBXERRx fields are only set when a response code is returned from a direct call. The ACBXERRA, ACBXERRD, and ACBXERRE fields are only set when the response code is related to buffer processing.

### Subcomponent Response Code (ACBXSUBR)

This field contains the response code from any error that occurred when an Adabas add-on product intercepts the Adabas command.

### Subcomponent Response Subcode (ACBXSUBS)

This field contains the response subcode from any error that occurred when an Adabas add-on product intercepts the Adabas command.

### Subcomponent Error Text (ACBXSUBT)

This field contains the error text of any error that occurred when an Adabas add-on product intercepts the Adabas command.

### Compressed Record Length (ACBXLCMP)

This field returns the compressed record length when a record was read or written.

This is the length of the compressed data processed by the successful Adabas call. If the logical data storage record spans multiple physical data records, the combined length of all associated physical records may not be known. In this case, Adabas returns high values in the low-order word of this field.

### Decompressed Record Length (ACBXLDEC)

This field returns the decompressed record length. This is the length of the decompressed data processed by the successful call. If multiple record buffer segments are specified, this reflects the total length across all buffer segments.

#### Command Time (ACBXCMDT)

The command time field is used by Adabas to return the elapsed time that was needed by the nucleus to process the command. In contrast to the mainframe, where this field is always filled by Adabas, it is only filled on open systems platforms if Command Logging is switched on or if the nucleus is started with the environment variable ADA_CMD_TIME set (the value is irrelevant).

#### User Area (ACBXUSER)

The user area field is reserved for use by the user program. When making logical user calls, the user area is neither written nor read by Adabas.

#### Reserved 4 (ACBXRSV4)

This field is reserved for use by Adabas. Your user program should set this field to binary zeros before the first Adabas call using this ACBX and then leave it unmodified thereafter.

### Differences between the ACB and the ACBX

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

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

#### Control Block Length

The ACBX is 192 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. When using the ACBX direct call interface, they are instead provided in the individual Adabas buffer descriptions (ABDs). So the ACBX contains no buffer fields corresponding to the ACBFBL, ACBIBL, ACBRBL, ACBSBL, and ACBVBL found in the ACB; the ABDs associated with the call are used instead. One ABD represents an individual Adabas buffer segment. They are described in *Adabas Buffer Descriptions*.

### Command Options, Additions, and Reserved Fields

The number of command option, additions, and reserved control block fields are larger 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.

### Field Length Differences

The lengths of many control block fields are larger in the ACBX than in the ACB, but note that the value length that is really supported is often smaller than the actual field length - this was done in order to enable larger values in future Adabas versions without having to change the interface. The following table summarizes these changes:

| Field Title | Length | | |
|---|---|---|---|
| | ACB | ACBX | |
| | | Field | Value |
| File Number | 2 (with call type 0x30 in File Number) | 4 | 2 |
| Database ID | 2 (with call type 0x30 in Response Code) | 4 | 2 |
| ISN | 4 | 8 | 4 |
| ISN Lower Limit | 4 | 8 | 4 |
| ISN Quantity | 4 | 8 | 4 |
| Compressed Record Length | 2 (in Additions 2) | 8 | 2 |
| Decompressed Record Length | 2 (in Additions 2) | 8 | 4 |
| Command Time | 4 | 8 | 8 |
| Format Buffer Length | 2 | 8 (in the ABD) | 4 |
| Record Buffer Length | 2 | 8 (in the ABD) | 4 |

| Field Title | Length | | |
|---|---|---|---|
| | ACB | ACBX | |
| | | Field | Value |
| Search Buffer Length | 2 | 8 (in the ABD) | 4 |
| Value Buffer Length | 2 | 8 (in the ABD) | 4 |

### Additional Fields in ACBX

The following additional fields are available in the ACBX:

| ACBX 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 hex 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). |
| 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 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 additional fields in the ACBX:

- In the ACB, the Response code field (ACBRSP) is used to store the database ID for hex 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, 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.

| Offset | ACB Field Name | ACBX 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) | ACBXISN (8-Byte ISN) |
| 1A | ACBRBL (Record buffer length) | |
| 1C | ACBSBL (Search buffer length) | |
| 1E | ACBVBL (Value buffer length) | |
| 20 | ACBIBL (ISN buffer length) | ACBXISL (8-Byte ISN Lower Limit) |
| 22 | ACBCOP1 (Command option 1) | |
| 23 | ACBCOP2 (Command option 2) | |
| 24 | ACBADD1 (Additions 1) | |

| Offset | ACB Field Name | ACBX Field Name |
|---|---|---|
| 28 | *(ACBADD1 continued)* | ACBXISQ (8-Byte ISN Quantity) |
| 2C | ACBADD2 (Additions 2) | |
| 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) |
| 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 Descriptions (ABDs)

If an Adabas call using the ACBX interface is made that requires buffer specifications, Adabas buffer descriptions (ABDs) *must* be used. ABDs *must not* be used when specifying an Adabas call using the classic ACB interface; if an Adabas call using the ACB interface is made that requires buffer specifications, specify the buffers or pointers to the buffers directly in the Adabas call itself. For more information about the ACBX and ACB interface direct calls, read *Calling Adabas*.

As the 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 in an ACBX call is not 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.

In UNIX and Windows applications, the addresses of ABDs are specified in the ABD list associated with the call; in mainframe system applications, the addresses of ABDs are specified directly in the Adabas call.

This section describes the structure of an ABD and ABD lists:

### Available ABD Types

Using ABDs in an ACBX interface direct call, the buffers used in a direct call can be contiguous or discontiguous. You can define ABDs for the following types of buffers:

- Format buffers
- Record buffers
- Multifetch buffers
- Search buffers
- Value buffers
- ISN 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.

In an ACBX interface call, there is a one-to-one correspondence between ABD and buffer specifications; each buffer you want to specify must have a corresponding ABD. The buffer can be specified in the ABD itself or referenced by indirect reference.

ABDs can be specified in any sequence in an ACBX interface direct call. However, if an ABD requires a matching ABD of another type, Adabas will match them sequentially. For example, if three format buffer ABDs and three record buffer ABDs are included in the call, the first format buffer ABD in the call is matched with the first record buffer ABD in the call, the second format buffer ABD is matched with the second record buffer ABD, and the third format buffer ABD is matched with third record buffer ABD.

If unequal numbers of match-requiring ABDs are specified, Adabas will generate a dummy ABD (with a buffer length of zero) for the missing ABD. For example, if three format buffer ABDs are specified, but only two record buffer ABDs are specified, a dummy record buffer ABD is created for use with the third format buffer ABD. If you would prefer that the dummy record buffer ABD be used for the second format buffer ABD instead, you must specify the dummy record buffer ABD yourself prior to the record buffer ABD to be used by the third format buffer ABD.

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. This applies to only a few commands; the most prominent of them is OP.

For information about the relationships between different buffer types, read *Understanding the Different Buffer Types*.

## ABD Structure

The following table describes the structure of the ABD.

| Name | Field | Control Block Position | Offset | Length (in bytes) | Format |
|---|---|---|---|---|---|
| ABDLEN | ABD length | 1-2 | 00 | 2 | binary |
| ABDVER | Version indicator | 3-4 | 02 | 2 | binary |
| ABDID | Buffer Type ID | 5 | 04 | 1 | alphanumeric |
| ABDRSV1 | Reserved 1 | 6 | 05 | 1 | binary |
| ABDLOC | Buffer location flag | 7 | 06 | 1 | alphanumeric/binary |
| ABDRSV2 | Reserved 2 | 8 | 07 | 1 | binary |
| ABDRSV3 | Reserved 3 | 9 | 08 | 4 | binary |
| ABDRSV4 | Reserved 4 | 13 | 0C | 4 | binary |
| ABDSIZE | Buffer size (allocated length) | 17-24 | 10 | 8 | binary |
| ABDSEND | Data length to send | 25-32 | 18 | 8 | binary |
| ABDRECV | Data length received | 33-40 | 20 | 8 | binary |
| ABDRSV5 | Reserved 5 | 41-44 | 28 | 4 | binary |

| Name | Field | Control Block Position | Offset | Length (in bytes) | Format |
|---|---|---|---|---|---|
| ABDADR | Indirect address pointer (if ABDLOC= C'I') | 45-48 | 2C | 4 | alphanumeric |
| --- | Buffer (if ABDLOC=C' ' or X'00') | 49-n | 30 | user-defined | not applicable |

## ABD Field Descriptions

Each of the fields in the ABD is described in this section, in the order they appear in the ABD structure.

### ABD Length (ABDLEN)

Required. Use this field to specify the length of the ABD. Currently, the value of this field must be 48.

### Version Indicator (ABDVER)

Required. This field identifies the version of the ABD structure. A value of C'G2' in this field indicates that the buffer definition is in the new, extended ABD structure.

### Buffer Type ID (ABDID)

Required. Use this field to identify the type of buffer described by the ABD, as shown in the following table:

| ID Setting | Type of Buffer |
|---|---|
| C'F' | Format |
| C'I' | ISN |
| C'M' | Multifetch |
| C'R' | Record |
| C'S' | Search |
| C'V' | Value |

#### Reserved 1 (ABDRSV1)

This field is reserved and must be set to binary zeros.

#### Buffer Location Flag (ABDLOC)

Required. Use this field to identify whether the location of the buffer is defined at an indirect address or is defined at the end of the ABD itself. If this field is set to "I" (C'I'), Adabas assumes indirect addressing is specified and will use the address specified in the indirect address pointer field (ABDADDR). In this case the buffer must reside in 31-bit addressable storage in the primary address space.

If this field is blank (C' ') or contains hexadecimal zeros, the buffer must immediately follow the ABD.

#### Reserved 2 (ABDRSV2)

This field is reserved and must be set to binary zeros.

#### Reserved 3 (ABDRSV3)

This field is reserved and must be set to binary zeros.

#### Reserved 4 (ABDRSV4)

This field is reserved and must be set to binary zeros.

#### Buffer Size (ABDSIZE)

Required. Use this field to specify the size of the buffer (in bytes), as it is allocated. A size of zero indicates a dummy buffer, which is treated as if it was not specified at all.

#### Data Length to Send (ABDSEND)

Required. Use this field to specify the length of the data (in bytes) to be sent to Adabas. The maximum value of this field cannot exceed the value set for the buffer size field (ABDSIZE). A buffer is sent to Adabas only if it is an input buffer for the type of command being issued.

**Note:** At this time, you must specify the same value for this field as you specify for the maximum buffer size (ABDXSIZE field). This is a temporary limitation of Adabas Version 6.1, that will be resolved in a future release.

### Data Length Received (ABDRECV)

This field specifies the length of the data (in bytes) returned to Adabas. The Adabas router sets this value at the end of call processing. The maximum value of this field will not exceed the value set for the buffer size field (ABDSIZE). A buffer is received from Adabas only if it is an output buffer for the type of command being issued.

### Reserved 5 (ABDRSV5)

This field is reserved and must be set to binary zeros.

### Indirect Address Pointer (ABDADR)

If you set the buffer location flag field (ABDLOC) to C'I' (indirect buffer), specify the address of the actual buffer in this field. More than 32 KB of data can now be specified in an Adabas buffer.

### Actual Buffer

If you set the buffer location flag field (ABDLOC) to C' ' (blanks), this field should contain the actual buffer. More that 32 KB of data can be specified in an Adabas buffer using the ACBX interface. For complete information on defining buffers, read *Defining Buffers*.

## ABD Lists

An ABD list is a file containing a list of pointer references to the Adabas buffer descriptions (ABDs) used for a direct call. ABD lists are used only for open systems ACBX direct calls. In the list, one ABD pointer is required for every buffer segment that is needed for the direct call.

ABD lists can include pointers to the ABDs for the following types of buffers: format, record, multifetch, search, value and ISN. Multiple ABDs of the same type can be specified in an ABD list.

ABDs can be specified in the list in any sequence. However, if an ABD requires a matching ABD of another type, Adabas will match them sequentially. For example, if three format ABDs and three record ABDs are included in the list, the first format ABD in the list is matched with the first record ABD in the list, the second format ABD is matched with the second record ABD, and the third format ABD is matched with third record ABD. If unequal numbers of match-requiring ABDs are listed (for example, if three format ABDs are listed, but only two record ABDs), Adabas will generate a dummy ABD for the missing ABD (in this case a dummy record ABD will be created).

For complete information about the relationships between the different types of ABD or buffer specifications, read *Understanding the Different Buffer Types*.

## Defining Buffers

If your direct calls use the *ACB direct call interface*, you can define five different types of buffers: format, record, search, value, and ISN buffers. These buffers are specified elsewhere in your application and are indirectly referenced in the ACB direct call (via pointer references).

If your direct calls use the ACBX direct call interface, you can define different types of buffer segments using Adabas buffer descriptions (ABD) and their associated buffer definitions: format, record, multifetch, search, value and ISN buffers. Each Adabas buffer segment is represented by a single ABD, although you can define multiple ABDs of some types in the same program. (For example, you can define multiple format ABDs for use by the same program.) A single buffer definition is associated with each ABD -- either indirectly by pointer reference or directly in the ABD itself. For detailed information about ABDs, including their structure, read *Adabas Buffer Descriptions (ABDs)*.

This section covers the following topics:

| | |
|---|---|
| *Understanding the Different Buffer Types* | Describes the different buffer types and the relationships between them, and correspondingly, the relationships between their associated ABDs (if you are making ACBX interface direct calls). |
| *Format and Record Buffers* | Describes format and record buffers and their syntax, together with examples |
| *Multifetch Buffers* | Describes multifetch buffers and their syntax. |
| *Search and Value Buffers* | Describes search and value buffers and their syntax, together with examples |
| *ISN Buffers* | Describes ISN buffers and their syntax. |

## Understanding the Different Buffer Types

The following syntax depicts the relationships between the different types of buffers that can be specified for a direct call. It should assist you in determining which buffer specifications are dependent on the presence of others.

**Notes:**

1. If you are specifying an ACBX interface direct call, corresponding Adabas buffer descriptions (ABDs) must also be specified. In addition, in ACBX interface direct calls when buffer specifications require the presence of other buffer specifications (for example, a format buffer requires the presence of a record buffer), Adabas pairs the buffers in the sequence in which they are specified (for example, the first specified format buffer ABD with the first specified record

buffer ABD). The syntax below can assist you in determining the sequence in which the ABDs should be listed in the call or in the ABD list.

2. If you are specifying an ACB interface direct call, the multifetch buffer listed in this syntax do not apply. In addition, buffers must be specified in this sequence: format, record, search, value, and ISN. If an earlier buffer in the sequence is not needed, but a later one is, all of the buffers up to the needed buffer must be specified, even if they are blank. For example, if an ACB interface direct call requires an ISN buffer but none of the other buffers, dummy format, record, search, and value buffers must be specified before the ISN buffer.

```
[format-buffer record-buffer... [multifetch-buffer]]...
[search-buffer value-buffer]
[ISN-buffer]
```

The following table describes the elements in this syntax:

| Element | Description | Conditions |
|---|---|---|
| *format-buffer* | A format buffer segment to use for the call. Each format buffer segment must end with a period and be a complete and valid standalone format buffer. | Required only if you need to specify the fields to be processed during the execution of an Adabas read or update command.<br><br>When required, multiple format buffers can be specified for an ACBX interface direct call. Only one format buffer can be specified in an ACB interface direct call.<br><br>If a format buffer is specified in the call, a corresponding record buffer must also be specified.<br><br>Optionally, in an ACBX interface direct call,a corresponding multifetch buffer can also be specified. |
| *ISN-buffer* | An ISN buffer segment to use for the call. | Required only if you need to set aside an area in storage to store ISNs or (in the case of an ACB interface direct call) an area to store the record descriptor elements (RDEs) of multifetched records.<br><br>When required, only one ISN buffer should be specified for the call. |
| *multifetch-buffer* | A multifetch buffer segment to use for the ACBX interface call. This buffer is only available for ACBX interface direct calls. | Used only by ACBX interface direct calls and required only if you need to set aside an area in storage to store the record descriptor elements (RDEs) of multifetched records.<br><br>When required, multiple multifetch buffers can be specified for an ACBX interface direct call.<br><br>If a multifetch buffer is specified, corresponding format and record buffers must also be specified. |

| Element | Description | Conditions |
|---|---|---|
| *record-buffer* | A record buffer segment to use for the call. | Required only if you need to set aside an area of storage to store record data required or collected for the call.<br><br>When required, multiple record buffers can be specified for an ACBX interface direct call. Only one record buffer can be specified in an ACB interface direct call.<br><br>If a record buffer is specified in the call, a corresponding format buffer must also be specified.<br><br>Optionally, in an ACBX interface direct call,a corresponding multifetch buffer can also be specified. |
| *search-buffer* | A search buffer segment to use for the call. | Required only if search criteria are required to select records for the call.<br><br>If a search buffer is specified in the call, a corresponding value buffer must also be specified. Only one search and value buffer pair can be specified in a single direct call. |
| *value-buffer* | A value buffer segment to use for the call. | Required only if search criteria are required to select records for the call.<br><br>If a value buffer is specified in the call, a corresponding search buffer must also be specified. Only one search and value buffer pair can be specified in a single direct call. |

## Format and Record Buffers

The format buffer specifies the fields to be read/updated during the execution of an Adabas read/update command.

For read commands, the values of the fields specified in the format buffer are returned by Adabas in the record buffer.

```
Format Buffer  AA,BB.               names of the fields to be read
```

```
Record Buffer  value(AA) value(BB)  field values returned by Adabas
```

For add/update commands, the new values for the fields specified in the format buffer are provided by the user in the record buffer.

```
Format Buffer  XX,YY.              names of the fields to be updated
```

```
Record Buffer  value(XX) value(YY)  field values provided by user
```

## Format Buffer Syntax

This section describes the syntax used to construct the format buffer.

The field names used in the examples in this section are based on the two file definitions contained in *Appendix A* in this manual.

**Note:** There are several restrictions for format buffers, which are described in *Messages and Codes*, *Nucleus Response Codes*, response 41.

The syntax of the format buffer is as follows:

```
{[{nX|'literal'}, ...] field_definition [segment | {,length}] [,format] ↩
[,edit_mask][,#'char_set']}, ... .
```

A comma must be used to separate adjacent format buffer entries. One or more spaces may be present between entries. The last entry may not be followed by a comma. The format buffer must be terminated with a period.

The following special option is available: a format buffer with the value `C.' will return the compressed record in the record buffer.

The terms nX, literal, field_definition, segment, length, format, edit_mask and #'char_set', which are used in the syntax of the Format Buffer, are described below.

### nX

For READ commands, nX specifies that n spaces are to be inserted in the record buffer by Adabas immediately before the next field value. The maximum allowed value of n is 253.

```
Format Buffer  AA,5X,BB.  5 blanks are to be inserted between
                          values for fields AA and BB
```

For UPDATE commands, nX causes n positions in the record buffer to be ignored by Adabas.

```
Format Buffer  AA,5X,BB.  5 positions between values for fields
                          AA and BB are to be ignored
```

```
Record Buffer  value(AA) 5 bytes value(BB)
```

**'literal'**

For READ commands, the character string contained within the quotation marks is to be inserted in the record buffer immediately before the next field value. The character string provided may be 1 - 254 bytes in length and may contain any alphanumeric character except a quotation mark.

```
Format Buffer  AA,'text',BB.  'text' is to be inserted between
                              values for AA and BB
```

```
Record Buffer  value(AA)text value(BB)
```

**field_definition**

The field_definition field indicates the elementary field, multiple-value field or periodic group to be used. Ranges of adjacent elementary fields can also be specified. For multiple value fields (MU) and periodic groups (PE), index ranges can be specified. The permissible combinations are as follows, where the field name is indicated by `name':

```
name [mu_pe_index]
name A [mu_pe_index]
name [pe_index]C
name S
name-name
name L[mu_pe_index]
name D[mu_pe_index]
```

where mu_pe_index is one of:

```
mu_index
pe_index
pe_and_mu_index
```

mu_index specifies an MU index or a range of MU indicies for an MU field. pe_index specifies a PE index for a PE or a range of PE indices for a PE. For mu_index and pe_index the following values may be specified:

**i**
: MU index for MU field or PE index for PE

**i-j**
: Range of MU indices or PE indices

**N**
: highest MU index for MU field or PE index for PE

**1-N**
: Range of all MU indices or PE indices (not permitted with update commands)

**Note:** If you specify 1-N, and no value exists, because the periodic group or the multiple-value field is empty, no occurrence is displayed. This is an inconsistency to mainframe, where at least one occurrence is displayed.

pe_and_mu_index specifies a PE index or a range of PE indices, and an MU index or a range of MU indices for an MU field in a PE. The following values may be specified:

**i(m)**
i = PE index, m = MU index for MU field in PE

**i(m-n)**
i = PE index, m-n = range of MU values for MU field in PE

**i(N)**
i = PE index, N = highest MU index for MU field in PE

**i(1-N)**
i = PE index, range of all MU indices for MU field in PE (not permitted with update commands)

**N(m)**
N = highest PE index, m = MU index for MU field in PE

**N(m-n)**
N = highest PE index, m-n = range of MU indices for MU field in PE

**N(N)**
Highest PE index and highest MU index for MU field in PE

**N(1-N)**
Highest PE index, range of all MU values for MU field in PE (not permitted with update commands)

**i-j(m)**
i-j = range of PE indices, m = MU index for MU field in PE

**i-j(m-n)**
i-j = range of PE indices, m-n = range of MU indices for MU field in PE

**i-j(N)**
i-j = range of PE indices, N = highest MU index for MU field in PE

**i-j(1-N)**
i-j = range of PE indices,1-N = range of all MU values for MU field in PE (not permitted with update commands)

If you specify a range both for the MU indices and for the PE indices, the corresponding sequence of record buffer elements starts with all specified elements for the lowest specified PE index, and ends with all specified elements for the highest specified PE index.

**Example:**

`AA1-2(3-4)` Is equivalent to AA1(3),AA1(4),AA2(3),AA2(4).

These combinations are described in detail in the following.

**name [mu_pe_index]**

The following combinations are permitted for name [mu_pe_index]:

- `name`

  The name of the field (or group) for which the value (or values) is requested, or a new value (or values) is being provided.

  The name specified must be two characters in length and must be present in the field definition table of the file being read/updated.

  The name may refer to an elementary field, a group or a multiple-value field. The field or group must not belong to a periodic group. The first occurence of a multiple-value field without MU index references the first value of the multiple-value field, the second occurence to the second value of the multiple-value field, etc. The same multiple-value field should not be specified without MU index and with MU index in the same format buffer; the results of a command using such a format buffer are undefined.

  A subdescriptor or superdescriptor name may be specified for an access command if no parent field of the descriptor is a multiple field or a field within a periodic group. Phonetic or hyper-descriptors must not be used. For the L9 command, any descriptor other than a phonetic descriptor is allowed.

  For UPDATE commands, the same name may not be used more than once (except in the case of multiple-value fields as explained below).

  A name which refers to a group results in all of the fields within the group being referenced.

  `GA` Refers to group GA (equivalent to the specification AA,AB).

  A group name may not be used if the group contains a multiple-value field.

  The use of group names will result in a significant reduction in the time required to process the command.

  Specifying multiple-value fields without an index in an UPDATE command allows you to replace all old multiple-value field values by exactly the number of fields specified in the format buffer. In this way, the multiple-value field count of an MU field without the NU option can be reduced again.

- `name mu_or_pe_index`

  The user must specify which occurrence is to be referenced, if one of the following is to be referenced:

  - a periodic group;
  - a field within a periodic group that is not a multiple-value field;
  - a multiple-value field;
  - a subdescriptor or superdescriptor with a field within a periodic group as parent field, which does not have a multiple-value field as parent field;
  - a subdescriptor or superdescriptor with a multiple-value field as parent field, which does not have a field within a periodic group as parent field.

  This is done by appending a numeric subscript (leading zeros permitted) to the name.

  **Note:** Subdescriptors or superdescriptors with fields within a periodic group or a multiple-value fields as parent field in the format buffer are not supported with Adabas versions < 5.1.

| | |
|---|---|
| `GB3` | The third occurrence of the periodic group GB is referenced (fields BA3, BB3, BC3). |
| `BB6` | The field BB in the 6th occurence of the periodic group GB is referenced. |
| `MF02` | The second value of the multiple-value field MF is referenced. |

N refers to the last occurence of a periodic group, a field in the last occurence of a periodic group or a multiple-value field for a read command; for an update command, a new occurence will be appended. A periodic group name must not be used if the periodic group contains a multiple-value field.

A range of occurrences of a periodic group (or a field within a periodic group) may be referenced by specifying the first and last occurrence number to be referenced (connected by a hyphen) after the name. A multiple-value field may also be referenced. A descending range may not be specified. A periodic group name must not be used if the periodic group contains a multiple-value field.

| | |
|---|---|
| `GB2-4` | The second through fourth occurrences of the periodic group GB are referenced (BA2, BB2, BC2, BA3, BB3, BC3, BA4, BB4, BC4). |

| | |
|---|---|
| `BA2-4,BC2-4` | The second through fourth occurrences of BA and BC are referenced (BA2, BA3, BA4, BC2, BC3, BC4). |
| `MF1-3` | The first 3 values of the multiple-value field are referenced. |
| `GB2-GB4` | Invalid, incorrect syntax. |
| `GB4-2` | Invalid, descending range. |

name 1-N means the first to the last occurrence of the field.

- `name mu_and_pe_index`

  If a multiple-value field contained within a periodic group or a subdescriptor or superdescriptor where a parent field is a field within a periodic group and a parent field is a multiple-value field (they can be the same – a multiple-value field within a periodic group) is to be referenced, the periodic group occurrence number (i), followed by the desired multiple-value field value (m) or range of values (m-n), must be specified.

  **Note:** Subdescriptors or superdescriptors with fields within a periodic group or a multiple-value fields as parent field in the format buffer are not supported with Adabas versions < 5.1.

| | |
|---|---|
| `CB2(5)` | The fifth value of the multiple-value field CB in the second occurrence of the periodic group GC is to be referenced. |
| `CB2(1-5)` | The first five values of the multiple-value field CB in the second occurrence of the periodic group GC are to be referenced. |

  N means the last occurrence of either the last occurrence of the element in a periodic group or the last value of a multiple-value field or both. In an update command, this means an append of a value.

  If a range of multiple-value field values within a range of periodic group occurrences is to be referenced, the range of the periodic group occurrences (i-j), followed by the range of the multiple-value field values (m-n) must be specified.

| | |
|---|---|
| `CB1-2(1-4)` | The first four values of the multiple-value field CB in the first occurrence of the periodic group GC are referenced, followed by the first four values of CB in the second occurrence of GC. |

**name A [mu_pe_index]**

Appending an `A' to the name of an elementary field, a multiple-value field, a field within a periodic group, or a multiple-value field within a periodic group indicates the add option. The permitted combinations are the same as for name [mu_pe_index].

If this option is used, a value can be added to a field instead of the value in the field being overlayed. This saves some Adabas calls, e.g. the sequence L4-A1 can be reduced to a single A1 call.

The following formats are supported: unpacked (U), packed (P), fixed point (F) and floating point (G).

For all commands other than A1, the A suffix is ignored.

**name [pe_index]C**

The highest occurrence number of a periodic group, or the number of existing values of a multiple-value field, or the number of existing values of a multiple value field in a PE for one occurrence or a range of occurrences of the PE may be obtained by appending the literal `C' to the periodic group or multiple-value field name.

`GBC` The highest occurrence number of the periodic group GB is referenced.

`MFC` The number of existing values of the multiple-value field MF is referenced.

For UPDATE commands, a `C' element is ignored and its corresponding count within the record buffer is skipped.

The count is returned in the record buffer as a one-byte binary number, unless an explicit length and/or format is specified by the user (see length and format parameters).

The count of the existing values of a multiple-value field within a periodic group may be obtained by indicating the periodic group occurrence with a one or two digit index and the literal `C' immediately following the multiple-value field name.

`CB4C` The count of the existing number of values of the multiple-value field CB in the fourth occurrence of the periodic group GC is referenced.

For UPDATE commands, a `C' element is ignored and its corresponding count within the record buffer is skipped. This is one byte for count fields which are specified without a length override.

The user cannot directly update the contents of multiple-value field or periodic group count fields. These count fields are automatically updated by Adabas when multiple-value field values and periodic group occurrences are added/deleted.

`MFNC` Specifies the count of existing values of the multiple-value field in the final periodic group.

**name S**

The `S' element is used when reading or updating the SQL null–value attribute of fields which have SQL null-value support. The `S' element is formed by appending the literal `S' to the field name.

This element can only be applied to elementary fields that are not in a periodic group, with the NC option (SQL null value supported) specified. If no other format or length is specified, it is 2 bytes long and the format is fixed point.

For UPDATE commands, the SQL null value will be stored in the field if the corresponding value in the record buffer is –1. Any other negative values result in a response 52. Positive values and 0 are ignored.

The `S' element is only required for UPDATE commands if the elementary field is being changed to the null value; otherwise the `S' element is not required but may be used. The `S' element and name element can be anywhere within the format buffer. When an `S' element updates a field's value to null then a following name element is ignored.

Several format buffer elements which refer to the same field cannot be contained within a format buffer for an UPDATE command.

For READ commands, the following information is returned in the `S' element:

| Value in S element | Meaning |
|---|---|
| -1 | Field contains the SQL null value. |
| 0 | Field is significant and the value is not truncated. |
| 1 | Field is significant and the value is truncated, the length of the value does not fit into the S element (for example, with default format F and default length 2: value length is greater than 32767). |
| >1 | Field is significant and the value is truncated: length of the value. |

**name-name**

The notation name-name may be used to reference a series of consecutive fields (as ordered in the field definition table). The user specifies the beginning and ending field names connected by a hyphen. No multiple-value field or periodic group may be contained within the series, and no length or format override is permitted.

A name which refers to a group may not be specified as the beginning or ending name, but a group may be embedded within the series.

Standard format and length is in effect for all the fields within the series.

| | |
|---|---|
| `AA-AC` | The fields AA, AB and AC are referenced. |
| `AA-GC` | Invalid, the series may not contain a multiple-value field or periodic group. |
| `GA-AC` | Invalid, the series may not begin/end with a group. |
| `AA,5,U,-AD` | Invalid, a length and/or format override is not permitted with a series notation. |

**name L[mu_pe_index]**

The format buffer indicator, L, can be used to retrieve or specify the actual length of any alphanumeric or Unicode field value that is either a multiple-value field or a field within a periodic group. This format buffer element is referred to as the *length indicator*.

The length indicator is specified using the field name followed by the character L (for example, FB='ACL.' would return the length of the AC field). By default, the compressed field length is returned in four-byte binary format.

**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.

Consider the following examples.

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

   ```
   FB='CBL2(5).'
   ```

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

   ```
   FB='MFL1-10.'
   ```

3. The following example for the multiple-value field MF is illegal as the length indicator does not support MU fields without an occurrence index:

   ```
   FB='MFL.'
   ```

**name D[mu_pe_index]**

Using the D element (daylight saving time indicator) in the format buffer is only allowed for fields with DT and TZ option. It indicates if daylight saving time was active for the date/time value. The default format for a D element is F with length 2 bytes. Its value is the number of seconds to be added to the standard time in order to get the current local time; this is usually 0 when daylight saving time is not active and 3600 when daylight saving time is active. For update format buffers, the value of the D element is subtracted from the value specified for the field in the record buffer in order to get the standard time. Using the D element also allows you to specify time values during the hour in which the daylight saving time is switched off again. If you specify date/time values in this hour, which occurs twice in add/update commands without a D element, it is assumed the values belonging to standard time are meant. In add/update commands for each D element there must be a 1:1 relationship between the D element and the corresponding date/time edit mask element. If your format buffer contains a D element, the value must be a correct daylight saving time indicator value valid for the time specified; otherwise you get a response code 55 with subcode 31.

In add and update commands, it is possible to specify the moment when the standard time or the daylight saving time ends, both as standard time and as daylight saving time value, but if you read such a value again, it is always displayed as the beginning of the new time period.

In the following example, the local time zone is MET (Middle European Time)., and field values for a field defined with the edit mask E(DATETIME) are read:

| Internal Value (UTC) | Field Values for Edit Mask E(DATETIME) | Value of D Element |
|---|---|---|
| 20080121135000 | 20080121145000 | 0 |
| 20091025003000 | 20091025023000 | 3600 |
| 20091025013000 | 20091025023000 (1) | 0 |

(1) In add or update commands it is also possible to specify 20091025030000 with D element 3600.

#### length, format

The length and format parameters are used to indicate that a field value is to be returned or provided in a length and/or format different from the standard length and/or format of the field.

The length specified must be large enough to contain the value in the chosen format and must not exceed the maximum length permitted (see following figure).

### Asterisk (*) Length Notation

For alphanumeric and Unicode fields you can specify an asterisk (*) instead of a length in the format element. The presence of an asterisk indicates that the amount of space available for the field value in the record buffer is variable and depends on the actual value of the field. However, unlike the zero length specification setting, *no* four-byte length field precedes the field value in the record buffer; the record buffer area corresponding to the format element only contains the value of the field. The actual 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). Instead, 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.

The format specified must be compatible with the standard format of the field (see the following table of length/format parameters). The processing rules for format conversion are shown later in this section.

These parameters may only be specified for an elementary or multiple-value field.

If the length and/or format parameters are omitted, the field value is returned/must be provided according to the standard length and/or format of the field. If a length of zero is specified, or name refers to a field which is defined as a variable-length field (no standard length), the value returned by Adabas will be preceded by a binary field which contains the length of the value (including the length indicator). For UPDATE commands, this length indicator must be provided by the user. The length of the length indicator is

- 4 bytes if the field has the L4 option;
- 2 bytes if the field has the LA option;
- 1 byte if the field has neither of these options.

| Source Format | Max. Length | Explanation | Destination Formats |
|---|---|---|---|
| A | 16381 (if the field has the option L4 or LA and is not a descriptor), or 1114 (if the field has the L4 or LA option and is a descriptor), else 253 | Alphanumeric | A |
| B | 126 | Binary (unsigned) | A, F, P, U |
| F | 8 | Fixed point (signed) | A, B, P, U |
| G | 8 | Floating point | G |
| P | 15 | Packed decimal<br>Signed, `+' = nA, nC, nE, nF<br>`–' = nB or nD in least significant byte, where `n' is the least significant digit of a given value | A, B, F, U |
| U | 29 | Unpacked decimal<br>Signed, `+' = 3n in least significant byte<br>`–' = 7n in least significant byte, where `n' is the least significant digit of a given value (as in zoned format) | A, B, F, P |
| W | For the internal value stored in UTF-8: 16381 (if the field has the option L4 or LA, and is not a descriptor), 1144 (if the field has the option L4 or LA, and is a descriptor), else 253. The external values, which may use different encodings, may be larger, if the field has the option LA or L4, but without one of these options the external values are also limited to 253 bytes. | Unicode | W |

All lengths are in bytes. The signs in P and U formats are hexadecimal in nibbles or bytes.

Conversion from BINARY is limited to values between 0 and 2**64 - 1; conversion to BINARY is limited to values between 0 and 2**80 -1.

Conversion from or to FIXED POINT is limited to values between -(2**63) and 2**63 - 1.

Conversion from a numeric format to alphanumeric results in an unpacked value, left–justified, without leading zeros and with trailing blanks. For example, the three–byte packed value `10043F' would be converted to `31303034333202020'. Value truncation is possible with this type of conversion. Floating point fields (G format) cannot be converted to another format. Conversions from 4 byte G format to 8 byte G format and vice versa are also not allowed.

**Note:** Starting with Adabas Version 6.3 SP2, for source format U, the sign bytes on EBCDIC machines, converted to ASCII, are also allowed as input for the sign bytes (see following table):

| Digit | EBCDIC, Hex | Display | ASCII, Hex |
|---|---|---|---|
| +0 | C0 | { | 7B |
| +1 | C1 | A | 41 |
| +2 | C2 | B | 42 |
| +3 | C3 | C | 43 |
| +4 | C4 | D | 44 |
| +5 | C5 | E | 45 |
| +6 | C6 | F | 46 |
| +7 | C7 | G | 47 |
| +8 | C8 | H | 48 |
| +9 | C9 | I | 49 |
| -0 | D0 | } | 7D |
| -1 | D1 | J | 4A |
| -2 | D2 | K | 4B |
| -3 | D3 | L | 4C |
| -4 | D4 | M | 4D |
| -5 | D5 | N | 4E |
| -6 | D6 | O | 4F |
| -7 | D7 | P | 50 |
| -8 | D8 | Q | 51 |
| -9 | D9 | R | 52 |

**Using the Length Indicator and Asterisk 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.

For Unicode fields, the length indicator currently always returns the internal length, i.e. the length of the value in UTF-8 encoding. This may be changed in a future version of adabas, so that the length of the value in another encoding is returned. It is, therefore, strongly recommended that you currently use the length indicator for reading W fields only if the default encoding for the Adabas session is UTF-8 and if no encoding is specified in a corresponding format buffer element with asterisk length. Otherwise you may receive different results with a future version of Adabas.

If character LB field L1 (format A) contains a 40,000-byte 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:

```
0x00009C40
```

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.

If a field in the format buffer is specified with its corresponding length indicator (for example, `FB='L1L,4,B,L1,*.'`), 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. 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.

If a field in the format buffer is specified with its corresponding length indicator (for example, `FB='L1L,4,B,L1,*.'`), and if the field is null suppressed (the NU option is specified for the field in the FDT) and the field value is all blanks, the returned field value length is zero; if the field is not null suppressed, the returned field value length is the length of one blank (one byte for alphanumeric fields and UTF-8 Unicode fields).

**Using the Length Indicator and Asterisk Length in Store/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.

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,*.'
```

#### Format Buffer: segment

Specifying segment instead of length allows you to process only a part of a field, which is of particular interest for LOB fields. You can only specify segment for fields defined with the format A. segment has the following syntax:

```
(byte_number, length [, length2])
```

For MU fields in PE, segment can only be specified if both the PE and MU index have been specified. If no segment is specified, the complete field is processed.

byte_number specifies the byte number of the first byte of the field segment to be processed; byte_number may be either a number or ‘*’.

If a number is specified, it denotes the position of the byte within the value where the segment to be processed start; the first (leftmost) byte of the value has byte number 1.

If you specify byte_number = '*', it is called the *-position. If you specify a *-position, it means that the segment starts at the current position. For an L1, L4 or A1 command with command option 2='L', the current position is specified in the field ISN LOWER LIMIT in the control block as the number of bytes preceding the segment to be processed; by specifying a *-position you can process different segments of a field using the same format buffer. For other commands, the current position is always the first (leftmost) byte of the field value. In a read command, only one field value can be processed with *-position; therefore, more than one format buffer element with *-position must not be specified in the format buffer.

**Notes:**

1. If the current position is past the end of the value, you get response code 3 for L1 and L4 commands with the L option. Otherwise, Adabas returns the old current position and length in the ISN LOWER LIMIT field. This means that you can process the complete value by a series of L1 or L4 commands, until you get response code 3, without having to change the control block and format buffer.
2. The current position to be specified in the ISN LOWER LIMIT field is one less than the explicit byte number to be specified in the format buffer for the same segment. Examples: L1(1,1000) is equivalent to L1(*,1000) with ISN LOWER LIMIT 0, and L1(1001,1000) is equivalent to L1(*,1000) with ISN LOWER LIMIT 1000.

length is the length in bytes of the field segment to be processed. The specified number of bytes must be provided at the corresponding position in the record buffer.

length2 is optional and relevant only for update (A1) commands; for other commands length2 is ignored. It specifies the length of the area in the old value that is replaced by the new segment; if specified, currently the value of length2 must be equal the value of length. If not specified, all remaining data of the old field value starting from position byte_number is replaced by the data in the record buffer.

**Notes:**

1. If a segment that is read contains bytes past the end of the field value, these bytes are filled up with blanks.
2. If a segment does not start with the first byte in an N1/N2 command, the field value is constructed by preceding the segment with blanks. The resulting value is compressed if the field does not have the NB option.
3. If a segment starts past the end of the old value for an A1 command, the new value is constructed by filling up the old value with blanks and adding the segment from the record buffer. The resulting value is compressed if the field does not have the NB option.
4. If a segment in an A1 command does not start past the end of the old value, but ends past the end of the old value, the value is created by adding the segment after the part of the old value before the segment. The resulting value is compressed if the field does not have the NB option.

5. If a segment in an A1 command ends before the end of the old value, the resulting value depends on length2: if it is specified, length2 bytes of the old value are replaced by the segment. If not specified, the rest of the old value is removed, and the resulting value is compressed if the field does not have the NB option.

### Format Buffer: edit_mask

edit_mask is one of:

```
numeric_edit_mask
E(date_time_edit_mask_name)
```

### Format Buffer: numeric_edit_mask

Numeric edit masks are used according to the standard edit mask rules as used in the COBOL programming language.

A numeric edit mask may only be specified for numerically-defined fields. All data that is being returned by Adabas to an edited field is converted to unpacked decimal format, regardless of the standard format of the field. The maximum number of digits (other than edit characters) which may be returned to an edited field is 15.

The user must ensure that the length parameter for the field for which an edit format has been specified is sufficiently large to contain the field value plus all required edit characters, otherwise Adabas will return a response code.

| Format | Default Length | Numeric Edit Mask Generated Maximum Length |
|---|---|---|
| E1 | 15 | zzzzzzzzzzzzzzz |
| E2 | 16 | zzzzzzzzzzzzzz9- |
| E3 | 17 | zzzzzzzzz99.99.99 |
| E4 | 17 | zzzzzzzzz99/99/99 |
| E5 | 20 | z.zzz.zzz.zzz.zzz,zz |
| E6 | 20 | z,zzz,zzz,zzz,zzz,zz |
| E7 | 21 | z,zzz,zzz,zzz,zz9.99- |
| E8 | 21 | z.zzz.zzz.zzz.zz9.99- |
| E9 | 21 | *,***,***,***,**9.99- |
| E10 | 21 | *.***.***.***.**9.99- |
| E11 | - | Reserved |
| E12 | - | Reserved |
| E13 | - | Reserved |
| E14 | - | Reserved |
| E15 | - | Reserved |

**Examples of Numeric Edit Mask Usage:**

```
Format Buffer  Field Value    Edited Value

XC,15,E1.      009877         bbbbbbbbbbb9877
XC,8,E4.       301177         30/11/77
XB,5,E7.       -366           3.66-
XB,7,E9.       542            **5.42b
```

**Format Buffer: E(date_time_edit_mask_name)**

For date/time fields defined with a date/time edit mask you can specify a date/time edit mask. While numeric edit masks are only allowed for read commands, date/time edit masks are also allowed for update commands. The edit masks are the same edit masks that are used in the file definition. The following combinations of edit masks in the field definition and edit masks in the format buffer are allowed (columns=format buffer, rows=FDT):

| | DATE | TIME | DATETIME | TIMESTAMP | NATTIME | NATDATE | UNIXTIME | XTIMESTAMP |
|---|---|---|---|---|---|---|---|---|
| DATE | A/A | -/- | F/T | F/T | F/T | C/C | F/T | F/T |
| TIME | -/- | A/A | -/- | -/- | -/- | -/- | -/- | -/- |
| DATETIME | T/F | X/- | A/A | F/T | F/T | T/F | C/C | F/T |
| TIMESTAMP | T/F | X/- | T/F | A/A | T/F | T/F | T/F | C/C |
| NATTIME | T/F | X/- | T/F | F/T | A/A | T/F | T/F | F/T |
| NATDATE | C/C | -/- | F/T | F/T | F/T | A/A | F/T | F/T |
| UNIXTIME | T/F | X/- | C/C | F/T | F/T | T/F | A/A | F/T |
| XTIMESTAMP | T/F | X/- | T/F | C/C | T/F | T/F | T/F | A/A |

The first value specifies the behaviour during read commands; the second the behaviour during update commands:

| Value | Meaning |
|---|---|
| -/- | Not allowed. |
| A | Allowed, no conversion required. |
| C | Conversion required; source and target value have the same precision. |
| F | Fill up the value with 0; the target value has a higher precision. Depending on the edit masks, an additional conversion may be required. |
| T | Truncate the value; the target value has a lower precision. Depending on the edit masks, an additional conversion may be required. |
| X/- | Extract the time component from the value for read operations; the target value does not contain date information. Depending on the edit masks, an additional conversion may be required. Only allowed for read operations. |

**Notes:**

- Adabas allows you to add a date/time edit mask to the field definition of an existing field of formats B, F, P and U. This can result in values in the database that are not correct values for this date/time edit mask.
- The following rules apply to the usage of date/time edit masks in the format buffer:
  - If you don’t specify a date/time edit mask for a field without the TZ option in the format buffer, no date/time conversions and checks are performed.
  - If you don’t specify a date/time edit mask for a field with the TZ option in the format buffer, the format buffer is treated as the date/time edit mask of the field definition specified.
  - If you don’t specify length or format together with a date/time edit mask in the format buffer, the default format or length of the field is used - ensure that you specify length or format if the default does not match the date/time edit mask specified.
  - If a date/time edit mask specified is not compatible with the date/time edit mask in the field definition, you get a response code 41.
  - If you specify a date/time edit mask for a field defined without the DT option, a check is made to see whether the value is a valid value for this edit mask - if it isn't, you get a response code 55.
  - If you specify a date/time edit mask that is only allowed only for read commands in the format buffer for an update or add command, you get a response code 44.
  - If you specify a date/time edit mask in the format buffer for an update or add command, a check is made to ensure that the value is a correct date/time value - you get a response code 55 if the value is not compatible with the edit mask.
  - If you specified a date/time edit mask in the format buffer for a read command, and the field contains an invalid date/time value, or the length of the field is not sufficient to store the value, you get a response code 55.

#### #'char_set'

This format buffer element may be specified only for W fields, and is used to specify the character set of the corresponding field in the record buffer. You must specify an encoding name that is listed in **http://www.iana.org/assignments/character-sets** - most of the character sets listed there are supported by ICU, which is used by Adabas for internationalization support.

**Example for Character Set Specification**

```
#'UTF-16BE'
```

## Format Buffer Examples

This section provides examples of format buffer and record buffer construction. All the examples in this section refer to the sample Adabas files in Appendix A.

**Example 1: Using elementary fields (standard length and format).**

```
Format Buffer : AA,5X,AB.
```

```
Record Buffer : AA value(8 bytes alphanumeric)
                5 spaces
                AB value(2 bytes packed)
```

**Example 2: Using elementary fields (length and format override).**

```
Format Buffer :  AA,4,5X,AB,3,U,W1,50,#'UTF-16BE'.
```

```
Record Buffer :  AA value (4 bytes alphanumeric)
                 5 spaces
                 AB value (3 bytes unpacked)
                 W1 value (50 bytes = 25 characters in UTF-16BE encoding)
```

**Example 3: A reference to a periodic group.**

```
Format Buffer : GB1.
```

```
Record Buffer : BA1 value (1 byte binary)
                BB1 value (5 bytes packed)
                BC1 value (10 bytes alphanumeric)
```

**Example 4: The first two occurrences of periodic group GB are referenced.**

```
Format Buffer : GB1-2.
```

```
Record Buffer : BA1 value (1 byte binary)
                BB1 value (5 bytes packed)        GB1
                BC1 value (10 bytes alphanumeric)
                BA2 value (1 byte binary)
                BB2 value (5 bytes packed)        GB2
                BC2 value (10 bytes alphanumeric)
```

**Example 5: The sixth value of the multiple-value field MF is referenced.**

```
Format Buffer : MF6.
```

```
Record Buffer : MF value 6 (3 bytes alphanumeric)
```

**Example 6: The first two values of the multiple-value field MF are referenced.**

```
Format Buffer : MF01-02.
```

```
Record Buffer : MF value 1 (3 bytes alphanumeric)
                MF value 2 (3 bytes alphanumeric)
```

**Example 7: The highest occurrence number of the periodic group GC and the existing number of values for the multiple-value field MF are referenced.**

```
Format Buffer : GCC,MFC.
```

```
Record Buffer : Highest occurrence count for GC (1 byte binary)
                Value count for MF (1 byte binary)
```

## Format Buffer Performance Considerations

Performance improvements may be achieved by using the following guidelines during format buffer construction:

- Use group names wherever possible rather than referring to elementary fields individually. The use of group names reduces the time required by Adabas to interpret the format buffer. The use of the field series notation does not result in performance improvements. A field series notation is converted by Adabas into a series of elementary fields;
- Use length and format overrides only when necessary. Using overrides requires additional processing time when interpreting the format buffer and when processing the field;
- If the same fields of a record are to be read and then updated, the same format buffer should be used for the read and update commands.

## Record Buffer

The record buffer is used primarily with the read (L1-L6, L9, S1/S2/S4 with read option) and the update (A1, N1/N2) commands.

For read commands, Adabas returns the requested field values in this buffer. The field values are returned in the order specified in the format buffer. A value is returned in the standard length and format defined for the field, unless a length and/or format override was specified in the format buffer. If the value is a null value, it is returned in the format in effect for the field:

| Format | Empty Value Format |
|---|---|
| ALPHANUMERIC | blanks |
| BINARY | binary zeros |
| FIXED POINT | binary zeros |
| FLOATING POINT | binary zeros |
| UNPACKED DECIMAL | unpacked decimal zeros |
| PACKED DECIMAL | packed decimal zeros |

Adabas returns the number of bytes equal to the combined lengths (standard or overridden) of all requested fields.

For update commands, the user provides the values for the fields to be updated in this buffer. If an empty value is being provided, it must be provided according to the format in effect as described above.

The record buffer is also used to transfer information to or from Adabas in the following commands:

- End transaction (ET), or close user session (CL). The user provides the data to be stored in an Adabas system file. Storing user data with any of the above commands is optional;
- Read ET data (RE). Adabas returns the user data stored in the Adabas system file;
- Write user data to the protection log (C5). The user provides the data to be written to the Adabas data protection log;
- Open user session (OP). The user indicates the type of updating to be performed (exclusive control or ET logic) together with the files to be accessed/updated. User data stored in the Adabas system file is returned by Adabas (optional);
- Read field definitions (LF). Adabas returns the field definitions for the file.

## Search and Value Buffers

The search and value buffers are used together to define the search criterion to be used to select a set of records using a find command (S1, S2, S4). They are also used read logical sequential (L3/L6) and the read descriptor values (L9) commands to indicate the starting value for a given sequential pass of a file.

The user provides the search expression(s) in the search buffer and the values which correspond to the search expressions in the value buffer.

The search and value buffer are also used with the update record (A1) and delete record (E1) commands; the search buffer syntax for these commands is provided in *Adabas Commands*.

## Search Buffer Syntax for S1/S2/S4 Commands

The search buffer as used with the S1, S2 and S4 commands is constructed according to the following syntax:

```
search expression [,connecting operator, search expression]... .
```

A comma must be used to separate all search buffer entries. One or more spaces may be present between entries. The last entry may or may not be followed by a comma. The search buffer must be terminated with a period.

The syntax for a search expression is:

```
{[name[i]D[,length][,format],] name[i][,length] [,format] ↩
[,E(date_time_edit_mask_name)] [,C]
 [,#'char_set'][,comparator]} | {nameS[,length] [,format]} | (command_id)
```

The terms used in this syntax are described in the following sections.

```
name[i|S]
```

The name of the field or derived descriptor to be used in the search expression. The name must refer to a field, subdescriptor, superdescriptor, collation descriptor, hyperdescriptor or phonetic descriptor.

If you search for a field within a periodic group, a superdescriptor or collation descriptor derived from a field within a periodic group, or a hyperdescriptor with the PE option, a subscript (leading zeros permitted) may be appended to the name in order to limit the search to only those values located in the occurrence specified. If no subscript is provided, the values in all occurrences will be searched.

If you search for a multiple-value field, or a subdescriptor, a superdescriptor or a collation descriptor derived from a multiple-value field, all values of the field will be used in the search. For this type of fields, it is not possible to restrict the search results by specifying a subscript to only one occurrence of the multiple field.

If the field is defined with the NU option (null value suppression), null values are not stored in the inverted lists; therefore, a search for all the records which have the null value will always result in no records found (even if there are records in Data Storage which contain a null value for the field). This rule also applies to subdescriptors or collation descriptors. A superdescriptor value is not stored if at least one field from which it is derived is defined with the NU option and the value for that field is null.

A search can be made for all of the records that have the SQL null value, by appending an upper case `S' to the field name if the field is defined with the NC option (SQL null value) or with the NU option (null value suppression, here the Adabas null value is interpreted as an SQL null value).

```
length
format
```

The length and format of the field value as provided in the value buffer may be explicitly stated with these parameters. If the length and/or format parameter is omitted, the value must be provided in accordance with the standard length and format of the field.

**Note:** For a superdescriptor, in general, the full length should be explicitly specified or the default length must be used. Exceptions to this rule are, for example, superdescriptors that have only alphanumeric parent fields.

Two figures showing the possible formats which can be used and the processing rules which apply to each format are provided at the end of this section.

```
E(date_time_edit_mask_name)
```

Date/time edit masks can be used in the same way as date/time edit masks in the format buffer; please refer to the description of date/time edit-masks in the format buffer for further information.

```
name[i]D
```

If you want to specify a date/time value with the daylight saving time indicator, the daylight saving time indicator must be specified first, followed by the field specification including the date/time edit mask. Name and, if specified, the PE index for the daylight saving time indicator must be the same as in the following date/time field specification. The default length and format for the daylight saving time indicator are 2,F.

```
C
```

The C option may be specified only for collation descriptors without the HE option. When the option is specified, the corresponding value in the value buffer is not the collating key, but the value of the parent field.

If you don't specify the C option for a collation descriptor without HE option, because you want to specify collation keys that you created yourself with ICU, please note that Adabas uses ICU version 3.2. ICU keys created by a different ICU version may be incompatible with the collation keys used by Adabas.

```
#'char_set'
```

This search buffer element may be specified only for W fields, and is used to specify the character set of the corresponding field in the value buffer. You must specify an encoding name that is listed at **http://www.iana.org/assignments/character-sets** - most of the character sets listed there are supported by ICU, which is used by Adabas for internationalization support.

### Example for Character Set Specification

```
#'UTF-16BE'
```

```
comparator
```

A comparator may be used as the last entry of a search expression to indicate the logical operation to be performed between the preceding field and its corresponding value in the value buffer.

The following comparators may be specified:

| Comparator | Meaning |
|---|---|
| EQ | equal to |
| NE | not equal to |
| GE | greater than or equal to |
| GT | greater than |
| LE | less than or equal to |
| LT | less than |

For the first operand of an S operator, only the comparators GE and GT are allowed - EQ is also accepted and handled like GE. For the second operand of an S operator, only the comparators LE and LT are allowed - EQ is also accepted and handled like LE.

If no comparator is specified, an `equal to' operation is assumed.

**Example:**

| | |
|---|---|
| `AA.` | AA equal to the value specified in the value buffer. |
| `AA,LT.` | AA less than the value specified in the value buffer. |
| `AA,GE.` | AA greater than or equal to the value specified in the value buffer. |

```
Command_id
```

A search expression may consist of a command ID value (enclosed within parentheses) which identifies a list of ISNs resulting from a previous find command in which the SAVE ISN LIST option was used. An ISN list resulting from an S8 or S9 command may also be used.

### Connecting Operators

A connecting operator may be used to connect search expressions. The permissible connecting operators (in order of increasing precedence) are:

| Connecting Operator | Meaning | Examples |
|---|---|---|
| R | Specifies that the results of two search expressions are to be combined using a logical OR operation. | `AA,R,AB.` |
| D | Specifies that the results of two search expressions are to be combined using a logical AND operation. | `AA,D,AB.` |
| O | Specifies that the results of two search expressions are to be combined using a logical OR operation. The O operator may only be used to connect search expressions which use the same descriptor. | `AA,O,AA.  Valid`<br><br>`AA,O,AB.  Invalid` |
| S | Expresses a FROM/TO range which involves two search expressions. The same descriptor must be used in both search expressions. Depending on the comparators defined for the operands, records for which the the field value is equal to the lower or upper bound can be included in the search result or omitted. | `AA,S,AA.`<br>`AA,GE,S,AA,LE.`<br><br>`Range including start and end ↩`<br>`values`<br><br>`AA,GT,S,AA,LT.`<br><br>`Range excluding start and end ↩`<br>`values` |
| N | Excludes a single value or a range of values from a previous FROM/TO range. This operator may only be specified in conjunction with the `S' operator. | `AA,S,AA,N,AA.        Valid`<br>`AA,S,AA,N,AB.        Invalid`<br>`AA,S,AA,N,AA,S,AA.   Valid`<br>`AA,S,AA,N,AA,N,AB.   Invalid` |

### Range Construction

There are two possibilities for constructing a range like NAME between A and Z:

1. NAME greater equal A `AND' NAME less equal Z

```
SB := NA,1,GE,D,NA,1,LE.
VB := AZ
```

2. NAME from A to Z

```
SB := NA,1,S,NA,1.
VB := AZ
```

### Precedence of Connecting Operators

If different connecting operators are used within a single search buffer, the operators are processed in the following order:

- evaluate all O/S/N operations, if necessary
- evaluate D operations, if necessary
- evaluate R operations, if necessary

Example: the search buffer: AA,S,AA,N,AA,O,AA,D,BB,R,CC,D,FF.

is evaluated as:(((((AA,S,AA),N,AA),O,AA),D,BB),R,(CC,D,FF))

### Search Buffer Syntax (Using Soft Coupling)

The search buffer for a search in which soft coupling is to be used is constructed according to the following syntax:

```
(mfile, mfield, sfile, sfield [,mfile, mfield, sfile, sfield],...)
/file/search expression [,operator, search-expression],... .
```

```
mfile
```

The main file. This file must also be specified in the file number field of the Adabas control block. The final resulting ISN list will contain ISNs contained in the main file only.

```
mfield
```

The field in the main file which is to be used as the soft coupling link field. This field must be a descriptor, subdescriptor, superdescriptor or hyperdescriptor. It may not be contained within a periodic group.

```
sfile, sfield
```

For each ISN selected from this sfile (according to the search criterion), the field specified as sfield will be read. The value of the field will then be used to determine which ISNs in the main file have a matching value.

sfield must be a field, either a non-descriptor or a descriptor, but not a subdescriptor, super-descriptor, hyperdescriptor collation descriptor or phonetic descriptor. It must have the same format as mfield. The standard length may be different.

The field may not be contained within a periodic group.

A maximum of 42 soft coupling criteria may be specified.

Please refer to the section *Search/Value Buffer Examples* in this chapter for examples of search criteria for soft coupling.

### Search Buffer Syntax for L3/L6/L9 Commands

The search buffer as used with the L3, L6 and L9 commands is constructed according to the following syntax:

```
{L3_SB_element [,comparator].} | {L3_SB_element,S, L3_SB_element.}
```

The first form of the search buffer is used if the value buffer contains only the starting value or ending value for the processing sequence. The entry in the command option 2 field and the comparator used determines whether the value specified in the value buffer is interpreted as the starting descriptor value or as the ending descriptor value.

The second form of the search buffer is used if the value buffer contains both the starting value and the ending value for the processing sequence. The first value in the value buffer specifies the lower limit of the range, and the second value specifies the upper limit of the range. Whether it is the lower limit or the upper limit that determines the starting value or the ending value depends on whether ascending or descending sequence has been specified. This makes it possible for an application program to change the direction of processing by simply modifying the entry in the command options 2 field in the control block.

The syntax of an L3_SB_element is:

```
[name [i] D [,length] [,format] ,] name [i]  [,length] [,format] , [,C] ↩
```

where

**name D**
: If you want to specify a date/time value with the daylight saving time indicator, the daylight saving time indicator must be specified first, followed by the field specification including the date/time edit mask.

**name**
: The name of the descriptor to be used for sequence control. The name specified must be the same as that specified in the Additions 1 field in the control block.

**i**
: If the descriptor for which values are to be returned is contained in a periodic group, this option can be used to specify the occurrence number for which values are to be returned. The value specified in the ISN field of the control block will be ignored if both the search buffer and the value buffer are specified. All of the occurrences of a given value will be returned if no index is specified.

**Note:** This option can only be used with the L9 command.

**length**

The length of the value as provided in the value buffer. If the length is not specified, it is assumed that the value is being provided using the default length of the descriptor as specified in the FDT.

**format**

The format of the value as provided in the value buffer. If the format is not specified, it is assumed that the value is being provided using the default format of the descriptor as specified in the FDT.

**comparator**

The comparators GE (greater than or equal to), GT (greater than), LE (less than or equal to), and LT (less than) can be used. GE is assumed if no comparator is specified.

**C**

The C option may only be specified for collation descriptors without the HE option. When this option is specified, the corresponding value in the value buffer is not the collating key, but the value of the parent field.

If you don't specify the C option for a collation descriptor without HE option, because you want to specify collation keys that you created yourself with ICU, please note that Adabas uses ICU version 3.2. ICU keys created by a different ICU version may be incompatible with the collation keys used by Adabas.

## Value Buffer

The user specifies the values for each descriptor specified in the search buffer in the value buffer.

The values provided must be in the same sequence as the corresponding search expressions specified in the search buffer.

All values provided must correspond to the standard length and format of the corresponding descriptor, unless the user has explicitly overridden the standard length or format in the search buffer.

Note that for phonetic descriptors and for hyperdescriptors with the HE option, you must specify not the internal search value, but rather a corresponding parent field value.

If the search expression consists of a command ID, no corresponding entry is made in the value buffer. However, a non–zero value buffer length must be specified.

If the search expression contains a field name FN followed by the S option, the value buffer must contain the either the hexadecimal value FFFF (select all records with the null value in the field FN) or the hexadecimal value 0000 (select all records that do not have a null value in the field FN).

Intervening blanks or other characters such as a comma must not be inserted between the values in the value buffer. No period is required to terminate the value buffer.

## Search/Value Buffer Examples

This section contains examples of search and value buffer construction. All examples refer to the sample Adabas files in Appendix A. The values for the value buffer are shown in character and/or hexadecimal notation.

**Example: A search which uses a single search expression.**

```
Search Buffer : AA.
Value Buffer  : 12345bbb
                0x3132333435202020
```

Result: This search returns the ISNs of all the records in file 1 which contain the value 12345 for field AA. The same search may be performed using AA,5. in the search buffer and the value 12345 (without trailing blanks) in the value buffer.

**Example: A search which uses two search expressions connected by the AND operator.**

```
Search Buffer : AA,D,AB.
Value  Buffer : 0x3132333435363738002C
```

Result: this search returns the ISNs of all the records in file 1 which contain the value 12345678 for the field AA and the value +2 for the field AB.

```
Search Buffer : AA,D,AB,3,U.
Value Buffer  : 12345678002
                0x3132333435363738303032
```

Result: this search produces the same result as the preceding search. It shows the use of the length and format override in the search buffer.

**Example: A search which uses three search expressions connected by the OR operator.**

```
Search Buffer : XB,3,U,O,XB,3,U,O,XB,3,U.
Value Buffer  : 284285290
                0x323834323835323930
```

Result: this search returns the ISNs of all the records in file 2 which contain any of the values 284, 285, or 290 for the field XB. Length and format overrides are also used.

**Example: A search which uses two search expressions connected by the FROM/TO operator.**

```
Search Buffer : XB,S,XB.
Value Buffer  : 0x020C030C
```

Result: this search returns the ISNs of all the records in file 2 which contain any value within the range +20 to +30 for the field XB.

**Example: A search which uses three search expressions connected by the FROM-TO and BUT-NOT operators.**

```
Search Buffer : XB,S,XB,N,XB.
Value Buffer  : 0x020C030C027C
```

Result: this search returns the ISNs of all the records in file 2 which contain any of the values in the range +20 to +30 but not +27 for the field XB.

**Example: A search where two ranges of descriptor values are connected by the OR operator.**

```
Search Buffer : XB,S,XB,O,XB,S,XB.
Value Buffer  : 0x001C200C500C600C
```

Result: this search returns the ISNs of all the records in file 2 which contain any of the values in the range +1 to +200 or +500 to +600 for the field XB.

**Example: A search in which a multiple-value field is used.**

```
Search Buffer : MF.
Value Buffer  : ABC
                0x414243
```

Result: this search returns the ISNs of all the records in file 1 which contain the value ABC for any value of the multiple-value field MF.

```
Search Buffer : MF2.  Invalid.
```

**Example: A search in which a descriptor within a periodic group is used.**

```
Search Buffer : BA.
Value Buffer  : 0x04
```

Result: this search returns the ISNs of all the records in file 1 which contain the value 4 in any occurrence of the descriptor BA (which is contained in a periodic group).

```
Search Buffer : BA3.
Value Buffer  : 0x04
```

Result: this search returns all the records in file 1 which contain the value4 in the third occurrence of the descriptor BA (which is contained within a periodic group).

**Example: A search which uses a subdescriptor. SA is a subdescriptor derived from the first four bytes of the field RA.**

```
Search Buffer : SA.
Value Buffer  : 0x41424344
```

Result: this search returns the ISNs of all the records in file 2 which contain the value ABCD for the subdescriptor SA.

**Example: A search which uses a superdescriptor with ALPHANUMERIC format. SB is a superdescriptor derived from the first eight bytes of the field RA and the first four bytes of the field RB.**

```
Search Buffer : SB.
Value Buffer  : 0x414243444546474831323334
```

Result: this search returns the ISNs of all the records in file 2 which contain the value ABCDEFGH1234 for the superdescriptor SB.

**Example: A search which uses a superdescriptor with BINARY format. SC is a superdescriptor derived from the fields XB and XC.**

```
Search Buffer :  SC.
Value Buffer  :  0x020F313233343536
```

Result: this search returns the ISNs of all the records in file 2 which contain the value +20 for the field XB and the value 123456 for the field XC.

**Example: A search which uses previously-created ISN lists (identified by their command IDs).**

```
Search Buffer :  (CID1),D,(CID2).
Value Buffer  :   not used
```

Result: this search returns the ISNs present in both ISN lists identified by the command IDs CID1 and CID2.

```
Search Buffer : (CID1),D,AB,3,U.
Value Buffer  : 123
                0x313233
```

Result: this search returns the ISNs of all the records in file 1 for which an ISN is present in the ISN list identified by CID1 and which contain the value +123 for the field AB.

**Example: A search in which a value operator is used.**

```
Search Buffer : AB,3,U,GT.
Value Buffer  : 100
                0x313030
```

Result: this search returns the ISNs of all the records in file 1 which contain a value greater than +100 for the field AB.

**Example: A search in which both value and connecting operators are used.**

```
Search Buffer : AB,3,U,GT,D,AA,1,GT.
Value Buffer  : 100A
                0x31303041
```

Result: this search returns the ISNs of all the records in file 1 which contain a value greater than +100 for the field AB and a value greater than A for the field AA.

**Example: A search which uses two search expressions of two different descriptors connected by OR; one of the search expressions is a complex expression that uses an AND operator.**

```
Search Buffer : AB,R,AC,3,NE,D,MF.
Value Buffer  : 0x001C414243414243
```

Result: this search returns the ISNs of all records in file 1 that contain the value 0x001C in the filed AB or don't contain the value 0x414243 in the first three bytes of the field AC and contain this value in the field MF.

**Example: A search based on SQL null value.**

**Note:** The following example assumes that the field ZZ is an alphanumeric field with the NC option. The field ZZ does not appear in the sample listings in Appendix A.

```
Search Buffer : AA,4,A,D,ZZS.
Value Buffer : 0x41424344FFFF
```

Result: records are selected with field AA = "ABCD" and a null value in field ZZ.

```
Search Buffer : AA,4,A,D,ZZS.
Value Buffer : 0x414243440000
```

Result: records are selected with field AA = "ABCD" and a non–null value in field ZZ.

**Example: Using a single soft coupling criterion and single search criterion.**

```
File Number  :  4
Search Buffer:  (4,AB,1,AC)/1/AB,S,AB.
Value Buffer :  ...........
```

1. Search file 1 for AB=value as provided in the value buffer;
2. For each resulting ISN in file 1, read the field AC and internally match the value with the corresponding value list for file 4;
3. The resulting ISN list from file 4 is provided in the ISN buffer.

**Example: Using a single soft coupling criterion and multiple search criteria.**

```
File Number  :  1
Search Buffer:  (1,AA,2,AB)/1/AC,D,AE,D,/2/AF,S,AF.
Value Buffer :  ...........
```

1. Search file 2 for AF=... through ... (values provided in the value buffer);
2. For each resulting ISN in file 2, read the field AB and internally match the value with the corresponding value list for file 1;
3. Search file 1 for AC=... and AE=... (values provided in the value buffer);
4. Match resulting ISN lists from steps 2 and 3;
5. ISNs resulting from step 4 are provided in the ISN buffer.

**Example: Using multiple soft coupling criterion and multiple search criteria.**

```
File Number  :  1
Search Buffer:  (1,AA,2,AB;1,AA,5,BA)
                /1/AC,D,AE,D,/2/AF,S,AF,D,/5/BC,S,BC.
```

1. Search file 2 for AF=... through ... (values provided in the value buffer);
2. For each resulting ISN in file 2, read the field AB and internally match the value with the corresponding value list for file 1;
3. Search file 5 for BC=... through ... (values provided in the value buffer);
4. For each resulting ISN in file 5, read the field BA and internally match the value with the corresponding value list for file 1;
5. Search file 1 for AC=... through ... (values provided in the value buffer);
6. Match the resulting ISN lists from steps 2, 4 and 5;

7. ISNs resulting from step 6 are provided in the ISN buffer.

**Example: A search using the NOT EQUAL operator**

```
Search Buffer : AA,4,A,NE.
Value Buffer  : ABCD
                0x41424344
```

Result: this selects all records with field AA not equal to "ABCD".

**Example: A search using the R operator ("OR" with different field names)**

```
Search Buffer : AA,4,A,R,BB,LT.
Value Buffer  : ABCD3000
                0x4142434433303030
```

Result: this selects all records with field AA equal to "ABCD" or field BB less than "3000".

```
Search Buffer : AA,4,A,R,(TEST).
Value Buffer  : ABCD
                0x41424344
```

Result: this selects all records with field AA equal to "ABCD" or which are members of the ISN list "TEST".

## Multifetch Buffers

Multifetch buffers are needed *only* for some Adabas commands run using the ACBX direct call interface; they are not needed for any ACB interface direct calls.

A multifetch buffer defines an area in storage to which Adabas can return the record descriptor elements (RDEs) of multifetched records. This buffer is only required by Adabas commands for which the multifetch option has been activated (by setting Command Option 1 to "M"). RDEs are each 16 bytes long.

When the multifetch option *M* is set in the Command Option 1 field of an ACBX command, Adabas returns all records being read in the specified record buffer segments, based on the format specifications in the corresponding format buffer segments. For each record buffer segment, the corresponding multifetch buffer segment contains multifetch headers describing the records in the record buffer segment.

For BT or ET commands, a multifetch buffer is *not* needed if Command Option 1 is set to "M". In this case, the ISN buffer is used to store the ISNs that need to be removed from the hold queue.

When a multifetch buffer is required, a corresponding format and record buffer are expected as well. If they are not provided, Adabas will create dummy format and record buffers (with length zero) to pair with the multifetch buffer. For complete information about the relationships between the different types of ABD or buffer specifications, read *Understanding the Different Buffer Types*.

## ISN Buffer

Adabas returns the ISNs of the records which satisfy the search criteria in this buffer.

Each ISN is provided as a four-byte binary number. The ISNs are provided in ascending sequence. For the S2 or S9 command, the ISNs are provided according to the user–specified sort sequence.

If the ISN buffer is not large enough to contain the entire resulting ISN list, Adabas will store (if requested) the overflow ISNs on the Adabas temporary working space. These overflow ISNs may be retrieved at a later time (see *Programming Considerations, ISN List Processing* for more detailed information).

**Note:** Since the ACB interface does not support multifetch buffers, for calls that use the ACB interface the ISN buffer is used instead of the multifetch buffer if the mutlifetch option is set.

## Summary of Adabas Format Conversion

The following table describes how values in the record/value buffer are converted for update (including add)/search commands, and how the values in the data record are converted for read commands.

| **Target Format**<br>**- Standard format of the field specified* for update/search commands**<br>**- Format specified in the format buffer for read commands** | **Source Format**<br>**- Format specified in the Format/Search Buffer for update/search commands**<br>**- Standard format of the field specified for read commands** | **Adabas Processing** |
|---|---|---|
| ALPHANUMERIC | ALPHANUMERIC | Only the length is adapted. |
| | BINARY,<br>PACKED,<br>UNPACKED | The value is converted to UNPACKED, leading zeros are removed, the value is left justified with trailing spaces added if necessary. |
| BINARY/FIXED | BINARY/FIXED | Only the length is adapted. |
| | PACKED,<br>UNPACKED | The value is converted to BINARY/FIXED POINT. |
| FLOATING POINT | FLOATING POINT | No conversion required. Different lengths cause a response code 41/61. |
| PACKED | PACKED | Only the length is adapted. |
| | BINARY,<br>UNPACKED,<br>FIXED | The value is converted to PACKED format. |
| UNPACKED | UNPACKED | Only the length is adapted. |
| | BINARY,<br>PACKED,<br>FIXED | The value is converted to UNPACKED format. |
| UNICODE | UNICODE | The value in the encoding specified as charset in the format/value buffer or in the OP command is converted to UTF-8. |

All other format combinations result in a response code 41 (format buffer) or 61 (search buffer).

* Subdescriptor format is the same format as the field from which it is derived.
If all of the fields from which a superdescriptor is derived are unpacked, the superdescriptor standard format is alphanumeric, unpacked or binary, depending on the definition of the superdescriptor. If not all of the fields from which it is derived are unpacked, the default superdescriptor format is alphanumeric if at least one of the fields from which it is derived is alphanumeric, otherwise it is binary.

### Sign Handling

Binary values are treated as unsigned numbers. Fixed point, floating point, unpacked and packed values are treated as signed numbers.

Valid signs which may be provided are:

| | |
|---|---|
| `Fixed point` | The sign is contained in the high-order bit<br><br>0 = positive<br>1 = negative (two's complement)<br><br>0x00000005 = +5<br>0xFFFFFFFB = -5 |
| `Floating point` | The sign is contained in the most-significant bit of the 4 or 8 bytes. |
| `Unpacked` | The sign is contained in the four high-order bits of the low-order byte (zoned format).<br><br>0x313233 = +123<br>0x313273 = -123 |
| `Packed` | The sign is contained in the four low-order bits of the low-order byte. |

```
Input          Output
-----          ------
A,C,E,F        C=positive
B,D            D=negative
```

0x123C = +123
0x123D = -123

If a search value is being provided for a superdescriptor which is derived from a packed field, an `A', `C', `E', `F' for positive or a `B', `D' for negative must be provided in the low nibble of the last byte.

## Calling Adabas from Application Servers

In application servers, multiple clients access the database through one server.

Due to the fact that the server issues the Adabas calls on behalf of the client, the server is the only virtual Adabas user that can be seen by the nucleus. The intention is to issue calls only from the server, but for the nucleus to recognize each user.

The Adabas user identification is made up from the node name, the user name, the environment-specific identification (process ID) and a timestamp.

The user identification information is set up on the client, which then provides the server with the information, or it is generated on the server.

In order to obtain the user identification information, the application has to issue a call lnk_get_adabas_id(). The interface for this routine is as follows:

```
#include "adabas.h"

int lnk_get_adabas_id (int buffer_length, unsigned char *buf);
```

The first argument buffer_length describes the length of a buffer that is addressed by a second argument buf. This buffer will be filled with the user identification by lnk_get_adabas_id(...). The buffer length that is supplied must be sufficient to accommodate all of the information that is returned. The structure that is returned is as follows:

```
struct adaid {
    unsigned short s_level;      //security: equivalent structure level: 3
    unsigned short s_size;       //size of the data structure
    unsigned char s_node[8];     //Adabas node name
    unsigned char s_user[8];     //Adabas user name
    unsigned int s_pid;          //process identification, 4 bytes
    long long s_timestamp        //microseconds since 1970
};
```

For the current version, the structure level is 3, and the returned buffer length is 32 bytes. This user identification information must be sent to the server. The server has to issue a function call lnk_set_adabas_id() in order to make the user identification available to Adabas. The interface for this routine is as follows:

```
#include "adabas.h"

int lnk_set_adabas_id (unsigned char *buf);
```

The argument is a pointer to a buffer that contains the user identification structure that is returned by lnk_get_adabas_id(...) on the client, or is generated on the server. ADALNK checks the input structure using the length field and the structure level, and makes the information available to Adabas. The user identification information has to be written by the server each time the client changes.

There are the following requirements for server-generated user identifications: neither the ASCII node name nor the ASCII user name nor the process ID may be filled with binary zeroes.

**Notes:**

1. The caller's node name, user name and process identification are used by default for the user identification structure if you don't call lnk_set_adabas_id. If you call lnk_set_adabas_id, you can set these fields to any values that you want.
2. When the Adabas user identification is displayed by utilities, for example ADADBM DISPLAY=UQ, the node name and the user name are displayed as 8 characters. Therefore, it is

strongly recommended to specifiy only printable characters for the node name and the user name; values provided as null-terminated strings with a string length of less than 8 bytes should be padded with blanks.

3. For compatibility reasons with previous Adabas versions, structure level 2 is also supported. Structure level 2 means that the timestamp is not contained in the adaaid structure; this reduces the size of the adaaid structure to 24 bytes. Therefore, structure level 2 is used in lnk_get_adabas_id, if lnk_get-adabas_id is called with the parameter buffer_length = 24. Using structure level 2 is only possible if no Adabas call has been performed before the first call to lnk_get_adabas_id or lnk_set_adabas_id. A program must not mix calls with structure level 2 and 3. However, an Adabas user identification received by a program with structure level 2 may be used in another program in lnk_set_adabas_id with structure level 3; you must set the time stamp to 0 in this case.

**Caution:** If you still use Adabas version 6.1 with an SP < 11 or Adabas version 6.2 or Net-Work version 7.3, the timestamp is not used for Adabas sessions where these versions are used. In such cases you have to take the following into consideration: if you want to create a new Adabas session with lnk_set_adabas_id, it is your responsibility to ensure that the combination (node, user, pid) in the adaid structure specified is not used for another Adabas session. Because the timestamp information is not yet used as part of the user identification, it is not sufficient that there are different timestamps in the adaid structure. If an existing user identification without a timestamp is used, unexpected errors will occur since both clients share the same Adabas session, which is not what is intended - a typical error in such a case is an Adabas response code 153 if both clients issue an Adabas call at the same time.

**Caution:** If an application uses lnk_set_adabas_id, it is no longer possible for Adabas to recognize whether an Adabas call is the first call for an Adabas session. Therefore, transaction consistency following an Adabas nucleus restart is only guaranteed if the nucleus is started with the option OPEN_REQUIRED.

**Note:** Because Net-Work also uses lnk_set_adabas_id, you should always use the option OPEN_REQUIRED if you use Net-Work.

### Multi-threaded Applications

If you write multi-threaded applications, you should use adalnkx.

The standard way to call Adabas in multi-threaded applications is for each thread that needs to execute Adabas calls to have its own Adabas session; each thread first makes an open command, then the Adabas calls to access the database, and finally a close command.

It is also possible to use one Adabas session in different threads by using the functions lnk_get_adabas_id and lnk_set_adabas_id. If you have an active Adabas session in thread 1, and you want to continue it in thread 2, you must first call lnk_get_adabas_id in thread 1 and then call lnk_set_adabas_id in thread 2.

**Note:** Older multi-threaded applications that were developed before adalnkx was available might still call the old interface adalnk. This is possible if you use the following rules:

- the adalnk calls must be synchronized - only one thread at a time may call adalnk;
- while adalnkx by default creates a separate Adabas session for each thread, adalnk by default only creates one Adabas session for the whole process. If you want to have more sessions, you must use lnk_set_adabas_id.

## Calling Adabas with Authentication

The application is responsible for setting the user credentials prior to opening a database session.

The following Adabas Client functions are provided to manage client sessions and set credentials:

| Steps | Function | Description |
|---|---|---|
| 1 (optional) | lnk_set_adabas_id() | Set the session identification. |
| 2 | lnk_set_uid_pw() | Set the authentication credentials for a specific database. |

Step 1 is optional. It is only required in application servers, where multiple clients access the database through one server. The Adabas session identification must then be set *prior* to setting the credentials.

It is mandatory, when accessing a secured database that the credentials are set *prior* to the start of each Adabas session and must be set for each database ID that is to be accessed.

The credentials are valid for the duration of the database session. Attempts to set credentials during an open database session will result in a nucleus response 9 “SE”: Security Violation. Open database transactions are backed out.

Database sessions with either invalid credentials or that are initiated without credentials will result in a “Security Violation”: Nucleus response 200 with subcode 31.

**Example**

Refer to the file **security_example.c** in Appendix D - Example Files in the Adabas Kit for further information.

```
#include "adabasx.h"


int           dbid;
static char  *uid;
static char  *passwd;

/*
** Set database-specific credentials
**
** This must be done -
** a) for each Adabas Session (Prior to start of session)
** b) for each dbid, which is to be accessed
**
*/

lnk_set_uid_pw(dbid, uid, passwd);
```

### Transition Mode

The Adabas nucleus user exit 21 could be used to set authentication credentials.

The purpose of this routine is to ease the transition of applications to use Adabas authentication. This routine is not intended as a replacement for setting credentials in the application and should be used as briefly as possible.

# 5 Programming Considerations

## Using Command IDs

The command ID field of the control block serves an important function during Adabas command execution. This section provides a summary of the uses of this field and describes the procedures to be followed when using command IDs.

A command ID is associated with the following information:

- If you perform read sequential commands (L2/L5, L3/L6, L9), Adabas needs information to identify the record to be processed with the next command. For this purpose, Adabas has a table for all active sequential command sequences, and the command ID identifies the entry in this table.
- If you perform search commands, Adabas needs information on the ISN list returned by the search commands. For this purpose, Adabas has a table for all active ISN lists, and the command ID identifies the entry in this table.
- If a format buffer is associated with a read, S1/S2/S4 search or update command, Adabas converts the format buffer to an internal format buffer that is used for record compression or decompression. Adabas can keep these internal format buffers in memory in order to reuse them for further commands with the same format buffer, which avoids a new conversion of the format buffer. The command ID can be used as a format buffer ID to identify an internal format buffer that already exists.

### Command IDs Used with Read Sequential Commands

The read sequential (L2/L5, L3/L6, L9) commands require that a non–blank, non–zero command ID be specified. The command ID is required by Adabas to return the records to the user in the correct sequence. These command IDs are maintained in the table of sequential commands.

The command ID is released by Adabas when an end-of-file condition (Adabas response code 3) is detected during read sequential processing (only from the table of sequential commands).

The command ID value provided with these commands is also entered and maintained in the internal format-buffer pool. It will only be released by an explicit RC command.

### Command IDs Used with ISN Lists

If a non-blank, non-zero command ID is specified for any command which results in an ISN list (S1, S2, S4, S8, S9), the command ID value may be used to identify the list at a later time.

If the SAVE ISN LIST option is used for an Sx command, a non-blank, non-zero command ID must be provided. The SAVE ISN LIST option causes the entire ISN list to be stored on the Adabas temporary working space. ISNs from the list may be subsequently retrieved by an Sx command or by using the GET NEXT option of the L1/L4 command.

If the SAVE ISN LIST option is not used and an ISN buffer-overflow condition occurs (the entire ISN list cannot be inserted in the ISN buffer), the overflow ISNs will be stored on the Adabas temporary working space only if a non-blank, non-zero command ID value was used. In this case, the command ID will be released by Adabas (only from the ISN list table) and the ISN list it identifies will be released by Adabas when all the ISNs have been returned to the user by subsequent Sx commands, or by L1 commands with the GET NEXT option when an end-of-file condition (Adabas response code 3) is detected.

### Automatic Command ID Generation

Automatic command ID generation is invoked by specifying a command ID of hexadecimal FFFFFFFF. This causes the Adabas nucleus to generate command IDs automatically, starting with hexadecimal 00000001 and incrementing by 1 for each new command ID.

### Releasing Command IDs

The user may release a command ID and its associated entries (or ISN list) with an RC or CL command. The RC command contains options which allow the user to release only those command IDs contained in the internal format–buffer pool, the table of sequential commands or the table of ISN lists.

The CL command causes all of the command IDs that are currently active for the user to be released.

Command IDs are not released at the end of a global transaction.

### Internal Identification of Command IDs

Each command ID entry is identified by Adabas using the internal user number and the command ID value.

A user need not be concerned with the command ID values in use by other users. The user should, however, exercise care when using the same command ID value for different commands, particularly for command IDs used for sequential read (L2/L5, L3/L6, L9) commands and Sx commands. The notation Sx command as used in this manual refers to any search command (S1, S2, S4, S8, S9).

### Empty Command IDs

You may sometimes want to perform Adabas commands without storing a command ID in the table of sequential commands, in the ISN list table or in the internal format buffer pool. This can be achieved by storing one of the following values in the command ID field:

- Binary zero.
- ASCII blanks (0x20202020). This is because character values are often used as command ID, although the command ID field is defined as binary.
- EBCDIC blanks (0x40404040). This is to enable mainframe users performing cross-platform calls to also specify blanks as an empty command ID.

Specifying one of these values in the command ID field has the following consequences:

- The command ID is not stored in the table of sequential commands, in the ISN list table and inthe internal format buffer pool.
- No RC command is required to release the command ID.
- The format buffer is re-translated for each call (unless you have specified a format buffer ID in Additions 5, as decribed in the next section) .
- It is not possible to perform subsequent commands for this command.

### Command ID Usage Examples

This section contains examples of command ID usage.

**Example 1: Find/Read processing**

A set of records is to be selected and read. The same format buffer is to be used for each record.

```
Find (S1)   CID=EX1B
Read (L1)   CID=EX1B
Read (L1)   CID=EX1B
```

**Example 2: Find/Read using the Get Next option**

A set of records is to be selected and read using the GET NEXT option of the L1/L4 command.

```
Find (S1)    CID=EX2A
Read (L1)    CID=EX2A
Read (L1)    CID=EX2A
Read (L1)    CID=EX2A
...
```

**Example 3: Read/Update processing**

A file is to be read and updated in logical sequence. The same format buffer is to be used for reading and updating.

```
Read Log Seq (L6)  CID=EX3A
Update (A1)         CID=EX3A
Read Log Seq (L6)  CID=EX3A
Update (A1)         CID=EX3A
...
```

**Example 4: Read/Find processing**

A file is to be read in logical sequence. A find command is to be issued to a second file using the value of a field read from the first file, and the records which result from the find command are then to be read using the GET NEXT option.

```
Read Log Seq (L3)  CID=EX4A
Find (S1)          CID=EX4B
Read (L1)          CID=EX4B
Read (L1)          CID=EX4B (RSP 3)
Read Log Seq (L3)  CID=EX4A
Find (S1)          CID=EX4B
Read (L1)          CID=EX4B
```

## Using Format Buffer IDs

Format buffer translations to an internal format buffer are time-consuming. For this reason, Adabas can save the internal format buffers in memory for reusage by other Adabas commands, and format buffer IDs were introduced to identify the internal format buffers to be reused.

Usually the command ID is used as the format buffer ID, but it is also possible to use Additions 5 to specify a format buffer ID separately:

- If there is more than one active read-sequential or search command using the same format buffer, a separate local format buffer ID may be useful. In this case “local” means user-specific.
- If several users run the same application program doing Adabas calls in parallel, a separate global format buffer ID may be useful. In this case “global” means valid for all Adabas users.

### Using a Command ID and Separate Local Format Buffer ID

Separate values can be used for command IDs and format buffer IDs. As long as the first byte of the Additions 5 field is not alphanumeric, the command ID is also used as the format buffer ID. If, however, the first byte of the Additions 5 field is a lower case character, the bytes 5 to 8 of the Additions 5 field are used as the (local) format buffer ID (see example 2 below). The format buffer ID must not start with hexadecimal FF.

### Using a Global Format Buffer ID

In many cases, numerous users who use the same program read or update the same fields of a file and therefore use identical format buffers. Defining a global format buffer ID for each of these programs means that Adabas does not have to store the same format buffer in the internal format buffer pool for each such program. If this option is used, the format buffer for each user is identified by the format buffer ID only, rather than by the format buffer ID and the internal communication ID. The result of this option is that numerous users can `share' a single format buffer, which in turn means that Adabas does not have to overwrite entries in the format buffer pool.

**Caution:** If a format buffer contains W fields, and the character set is not specified explicitly in the format buffer, Adabas implicitly uses the character set specified in the Adabs OP command via the parameter WCHARSET (default: 'UTF-8') . The information about the character set is included in the internal format buffer. This has the consequence that you may only use a global format buffer ID for such a format buffer when the same WCHARSET is specified in the OP comand of all Adabas sessions in which this global format buffer ID is used. Otherwise it may happen that the W fields are processed with the wrong character set.

The global format buffer ID option is set by setting the first byte of the Additions 5 field to a digit or upper case letter (see example 3 below).

### EBCDIC Characters in Additions 5

Adabas must also be able to process commands issued on mainframes. In this case, Additions 5 may contain EBCDIC characters. Because it may also contain binary data, no EBCDIC-ASCII conversion is performed for Additions 5. Therefore, EBCDIC lower-case characters (hex 8x, 9x) in first byte of Additions 5 indicate a separate local format buffer ID, EBCDIC upper-case characters (hex Cx, Dx) and EBCDIC digits (hex Fx) indicate a separate global format buffer ID, and EBCDIC blank (hex 40) indicates no separate format buffer ID.

The behavior of Adabas is undefined if you specify anything else in the first byte of Additions 5.

### Examples of Command ID and Format Buffer ID Usage

**Example 1: Using the same command ID and format buffer ID**

```
Additions 5: The first byte of the Additions 5 field is blank or binary zero
```

Result: the command ID is used as the format buffer ID.

**Example 2: Using separate local format buffer ID**

```
Additions 5: The first byte of the Additions 5 field is set to 'a'
```

Result: bytes 5 to 8 of the Additions 5 field are used as the format buffer ID (the format buffer ID must not start with hexadecimal FF).

**Example 3: Using global format buffer ID**

```
Additions 5: The first byte of the Additions 5 field is set to 'A'
```

Result: the Additions 5 field (8 bytes) is used as the global format buffer ID (usable by several users in parallel).

**Example 4: Using a global format buffer ID on mainframe**

```
Additions 5: The first byte of the Additions 5 field is set to hex D1
```

Result: Hex D1 = EBCDIC ‘J’ -> the Additions 5 field (8 bytes) is used as the global format buffer ID (usable by several users in parallel).

### Format Buffer IDs Used with Read, Search and Update Commands

The read commands (L1-L6, L9) and update commands (A1, N1/N2) require a format buffer which specifies the fields to be read or updated. A format buffer can also be specified for an S1/S2/S4 search command to read the first result record. This format buffer must be interpreted and converted into an internal format buffer by Adabas. The same format buffer ID may be used to avoid repetitive interpretation and conversion during successive commands which use the same format buffer. A read or update command in which a format buffer ID is used causes Adabas to check whether it is in the internal format buffer pool. If the format buffer ID is present, the internal format buffer for the format buffer ID is used, and no format buffer interpretation is required. The user may, therefore, achieve a significant decrease in the processing time required for read and update commands by using a format buffer ID when reading or updating a series of records in which the same format buffer is used. If the user is reading and updating the same fields (for example, L5 followed by A1), it is also recommended that the same format buffer ID be used for both commands (see the A1 and N1/N2 commands for restrictions on using the same format buffer for reading and updating).

If the internal format buffer pool is full, and a command is received in which a format buffer ID not in the pool is present, Adabas will overwrite the least active entry in the pool with the new format buffer ID. If a command is subsequently received which uses the deleted command ID, this command ID will in turn replace the least active entry in the pool. The format buffer must be newly interpreted and converted into an internal format buffer whenever the above occurs. For this reason, the format buffer should not be modified between successive read or update commands in which the same format buffer ID is used.

**Caution:** Please take care to ensure that you always specify a correct pair of format buffer ID and format buffer:

- If, for example, you use a new format buffer but forget to specify a new command ID, you may read the fields specified in the format buffer associated to the old command ID. This may lead to wrong results in the record buffer.
- If you don’t specify the correct format buffer for subsequent commands with the same format buffer ID, the program may work correctly for a long time, but suddenly it may happen that the corresponding internal format buffer has been overwritten in the format buffer pool. Then a new format buffer conversion is required, which may fail, or will be invalid if the correct format buffer has not been specified for the command.

## ISN List Processing

This section discusses the procedures used to retrieve ISNs from the Adabas temporary working space. If the GET NEXT option of the L1/L4 command is to be used to read the records which correspond to the ISNs contained in the ISN list, ISN handling as discussed in this section is performed automatically by Adabas, and the user need not make use of these procedures.

### Storing ISN Lists

Adabas stores ISNs on the Adabas temporary working space under the following conditions:

- An Sx command is issued, a non-blank non-zero command ID is specified and the SAVE ISN LIST option is specified. The entire resulting ISN list is stored in this case;
- An Sx command is issued, a non-blank non-zero command ID is specified, the SAVE ISN LIST option is not specified, and the resulting ISN list contains more ISNs than can be inserted in the ISN buffer. Only the overflow ISNs are stored in this case.

If an Sx command is issued with blanks or binary zeros in the command ID field, Adabas does not store any ISNs on the Adabas temporary working space.

### Retrieving ISN Lists

The user may retrieve ISNs stored on the Adabas temporary working space by issuing an Sx command in which the same command ID value is used as that used for the initial Sx command. When an Sx command with an active command ID value is issued, Adabas uses this as an indicator that the user is requesting ISNs from an existing ISN list. Adabas locates the ISN list identified by the command ID specified and inserts the next group of ISNs in the ISN buffer. As many ISNs are returned as can be inserted in the ISN buffer.

If the SAVE ISN LIST option was specified with the Sx command used to create the ISN list, Adabas uses the ISN specified in the ISN lower limit field to determine the next group of ISNs to be returned. The next group begins with the first ISN which is greater than the ISN specified in ISN lower limit. If binary zeros are specified, the next group begins with the first ISN in the list. If a value is specified which is greater than any ISN in the list, response code 3 is returned. Such an S1 call without an ISN buffer can be used for repositioning within the current ISN list.

If the ISN list was created using an S2/S9 command, the ISN specified must be present in the ISN list, otherwise response code 25 is returned. Using the SAVE ISN LIST option thus allows the user to move forwards and backwards within an ISN list. This is useful for programs which must perform forward and backward screen paging.

If the SAVE ISN LIST option was not specified with the Sx command used to create the ISN list, Adabas returns the ISNs in the order in which they are positioned in the list and deletes each group from the temporary working space when it has been inserted in the user's ISN buffer. The command ID used to identify the list is released when the last group of ISNs has been returned to the user. The ISN lower limit field is not used in this case.

The user may determine when all of the ISNs in a list have been retrieved by using the ISN quantity field of the control block. Adabas returns in this field, as a result of an initial Sx command, the total number of records which satisfy the search criteria. Adabas returns, as the result of a subsequent Sx command used to retrieve ISNs from the Adabas temporary working space, the number of ISNs which were inserted in the ISN buffer.

### Handling of ISNs that no longer exist

When an ISN list is processed, it may occur that ISNs found by the search operation no longer exist when the ISN is processed, for the following reasons:

- The ISN has been deleted after the ISN was included in the search result.
- The search operation processes a previously generated ISN list, and the ISN in that ISN list has already been deleted before the search operation started.

Please note that Adabas generally does not check whether the ISNs still exist when they are included in the search result. An exception is an initial Sx command, where a format buffer has also been specified. In this case, Adabas reads the first ISN of the resulting ISN list - if it no longer exists, the ISN is removed from the resulting ISN list. For subsequent Sx commands with a format buffer, you get a response code 113 if the first ISN in the ISN list to be processed does not exist. This is because the command ID is deleted after the last ISN has been returned. If you did not get all ISNs of the ISN list, it could be that the user assumes that another subsequent Sx command is performed, but in fact a new initial Sx command is performed.

If you process the ISN list using L1 commands with GET NEXT option, ISNs that no longer exist are skipped. Therefore, it may occur that you get an end-of-list condition (response code 3), although the number of ISNs read is less than the ISN quantity returned in the initial Sx command.

## Examples of ISN List Processing

**Example 1: Initial Sx call using SAVE ISN LIST option**

```
COMMAND = Sx
COMMAND ID = SX01
COMMAND OPTION 1 = H
ISN LOWER LIMIT = 0
ISN BUFFER LENGTH = 20
CALL ADABAS ...
```

Resulting ISN quantity = 7

Resulting ISN list:
(all ISNs are stored) 8 12 14 15 24 31 33

Resulting ISN buffer: 8 12 14 15 24

*Subsequent Sx call*

```
COMMAND = Sx
COMMAND ID = SX01
ISN LOWER LIMIT = 24
ISN BUFFER LENGTH = 20
CALL ADABAS ...
```

Resulting ISN quantity = 2

Resulting ISN buffer: 31 33 14 15 24

*Subsequent Sx call*

```
COMMAND = Sx
COMMAND ID = SX01
ISN LOWER LIMIT = 0
ISN BUFFER LENGTH = 20
CALL ADABAS ...
```

Resulting ISN quantity = 5

Resulting ISN buffer: 8 12 14 15 24

**Example 2: With ISN overflow handling**

*Initial Sx call (SAVE ISN LIST option not used)*

```
COMMAND = Sx
COMMAND ID = SX02
COMMAND OPTION 1 = blank
ISN LOWER LIMIT = 0
ISN BUFFER LENGTH = 20
CALL ADABAS ...
```

Resulting ISN quantity = 7

Resulting ISN list: (only ISNs 31 and 33 are stored)
8 12 14 15 24 31 33

Resulting ISN buffer: 8 12 14 15 24

*Subsequent Sx call*

```
COMMAND = Sx
COMMAND ID = SX02
ISN LOWER LIMIT  (not used)
ISN BUFFER LENGTH = 20
CALL ADABAS ...
```

Resulting ISN quantity = 2

Resulting ISN buffer: (ISNs 31 and 33 are deleted from the Adabas temporary working space, command ID SX02 is released).
31 33 14 15 24

A subsequent Sx call with command ID SX02 will be processed as an initial Sx call since SX02 was released after the second call which resulted in the last ISN being returned to the user.

**Example 3: Without ISN overflow handling**

*Initial Sx call with blank or zero command ID*

```
COMMAND = Sx
COMMAND ID = blanks or binary zeros
COMMAND OPTION 1 = blank
ISN LOWER LIMIT = 0
ISN BUFFER LENGTH = 20
CALL ADABAS ...
```

Resulting ISN quantity = 7

Resulting ISN list:
(no ISNs are stored) 8 12 14 15 24 31 33

Resulting ISN buffer: 8 12 14 15 24

A subsequent Sx call with command ID equal to blanks or binary zeros and ISN lower limit equal to 0 will result in the re-execution of the same find command with the same result as the initial call.

A subsequent call with command ID equal to blanks or binary zeros and ISN lower limit = 24 will re-execute the Sx command. The result of this command will be ISN quantity = 2 and ISNs 31 and 33 in the ISN buffer.

**Example 4: Reading ISN list with GET-NEXT option**

*Inital Sx command*

```
COMMAND = Sx
COMMAND ID = SX01
COMMAND OPTION 1 = blank
ISN LOWER LIMIT = 0
ISN BUFFER LENGTH = 4
FORMAT BUFFER = <not empty>
CALL ADABAS ...
```

Resulting ISN quantity = 3

Resulting ISN = 44

Resulting ISN list:
44 321 344 (only 321 and 344 are stored)

Resulting ISN buffer: 44

Resulting record buffer: field values of ISN 44

*Subsequent L1/L4 call*

```
COMMAND = L1 or L4
COMMAND ID = SX04
COMMAND OPTION 2 = N
FORMAT BUFFER = <unchanged>
CALL ADABAS ...
```

Resulting response code = 0

Resulting ISN = 321

ISN 321 is deleted from the Adabas temporary working space.

Resulting record buffer: field values of ISN 321

*Subsequent L1/L4 call*

```
COMMAND = L1 or L4
COMMAND ID = SX04
COMMAND OPTION 2 = N
FORMAT BUFFER = <unchanged>
CALL ADABAS ...
```

Resulting response code = 0

Resulting ISN = 344

ISN 344 is deleted from the Adabas temporary working space, the remaining ISN list is now empty

Resulting record buffer: field values of ISN 344

*Subsequent L1/L4 call*

```
COMMAND = L1 or L4
COMMAND ID = SX04
COMMAND OPTION 2 = N
FORMAT BUFFER = <unchanged>
CALL ADABAS ... ↩
```

Resulting response code = 3 (end of ISN list)

Command ID SY04 is released.

## Using the Multifetch Feature

The Multifetch feature is used to retrieve more than one record with one call. This leads to a considerable reduction in the communications overhead.

### Retrieval

The Multifetch feature is invoked by specifying the value `M' in the Command Option 1 field for any of the commands L1/L4, L2/L5, L3/L6 and L9. If, in addition, the Return option is required for these commands, the value `O' should be specified instead.

If you use the ACBX interface, multifetch buffers must also be defined within the user program.

If you use the ACB interface, an ISN buffer must also be defined within the user program.

Adabas uses the corresponding buffers as storage for data that describe the records returned in the record buffer. The first 4 bytes contain the number of records returned by the call. Each record in the record buffer is represented by a 16-byte entry in the ISN/multifetch buffer:

| Byte | Usage |
|---|---|
| 1-4 | Length of record |
| 5-8 | Adabas response code |
| 9-12 | ISN of record (L9 command: occurrence count if descriptor is within a periodic group) |
| 13-16 (L9) | ISN quantity |
| 13-16 (L3/L6) | Index in periodic group if descriptor used is within periodic group |

The ISN Lower Limit field in the command block can be used to limit the number of records to be returned. If this field is set to zero, the maximum number of records returned depends on the size of the record buffer and/or the size of the ISN/multifetch buffer.

If an ISN is in hold status for another user or if the hold queue limit is reached, no further records are returned to the record buffer. If the first record is held by another user and the Command Option 1 field is set to `M', an implicit wait is performed. If a record other than the first record is held by another user, Adabas immediately returns the records up to but not including that record to the user. If the Command Option 1 field is set to `O' and a record is held by another user, Adabas response code 145 is returned. For the first record, the response code is within the control block, and for subsequent records it is in the ISN/multifetch buffer.

If an error is detected while a record is being processed, the response code will be returned in the corresponding structure in the ISN/multifetch buffer.

The ISN returned in the control block is the ISN of the first record returned.

If the ISN/multifetch buffer length is less than 20 bytes, or if the record buffer is too small to process at least one record, an Adabas response 53 will be returned in the Control Block.

If an error is detected while the first record is being processed, the error response is returned in the Response Code field of the control block. If an error is detected while a record other than the first record is being processed, Adabas returns the records that were processed successfully and puts the the response code for the unsuccessful record in the corresponding structure in the user-defined ISN/multifetch buffer.

### Transaction Control

The Multifetch feature is invoked by specifying the value 'M' in the command option 1 field of the ET/BT command. This specifies that the list of ISNs provided by the user in the ISN buffer are to be released from hold status. The first 4 bytes contain the number of ISNs to be released. Each ISN is represented by an 8-byte entry in the ISN buffer:

| Byte | Usage |
|---|---|
| 1-4 | File number |
| 5-8 | ISN of record |

If all of the records are to be kept in hold status, the first 4 bytes of the ISN buffer must be set to zeros. In this case, the TT time limit is not restarted.

If a given combination of file number and ISN is not in hold status for the user in question, Adabas response code 144 is returned.

This response is given after the entire ISN buffer is processed.

If the ISN buffer length is less then 4 + (number of ISNs * 8), the Multifetch feature will be ignored (i.e. the call will be processed as a standard call) and an Adabas response 53 will be returned.

**Note:** For transaction control, the ACBX interface also uses ISN buffers, rather than Multifetch buffers, for the Multifetch feature.

## ACB versus ACBX Functionality

The Adabas functionality that is available with the ACB interface is a subset of the functionality available with the ACBX interface, with one exception - the MC command is only available via the ACB interface.

All ACB calls are transformed into an ACBX call by the Adabas interface routine (e.g. adalnkx), and the results are transformed back to the ACB interface. The following rules apply:

- All fields that are contained both in the ACB and in the ACBX are copied to the ACBX. Some fields are defined with larger sizes in the ACBX interface; this means that you must use the ACBX interface if you want to specify values that do not fit into an ACB field, but that do fit into the corresponding ACBX field. With Adabas Version 6.2, these larger lengths are not yet used.
- The ABDs required in the ACBX interface are generated from the Adabas buffer lengths. If a command requires buffers >= 64 KB, you must use the ACBX interface.
- As described in the previous section, the ISN buffer is sometimes used in the ACB interface when the ACBX interface uses a multifetch buffer. In such cases, the ABD for the ISN buffer of the ACB call is generated as the ABD for a multifetch buffer.
- Fields that are only contained in the ACBX are set to their default values; if you want to provide other values for these fields, you must use the ACBX interface, but for Adabas Version 6.2, there are no commands that use input from ACBX-only fields.
- The resulting values in fields that belong to fields contained both in the ACB and in the ACBX are copied back to the ACB. With Adabas Version 6.1, it is not possible that these fields have

values that do not fit into the ACB fields, if the fields are defined with a larger size in the ACBX than in the ACB.

- Information that is returned in ACBX-only fields is lost in ACB calls. In Adabas Version 6.2, only some diagnostic information for the command executed is returned in ACBX-only fields.

With this knowledge about the relationship between ACB and ACBx calls, it is not necessary to provide separate descriptions about how commands are executed with the ACB interface and with the ACBX interface.

## System Generated Fields

If a file contains system generated fields, only the behaviour of store and update commands changes - the behaviour of read and search commands remains unchanged.

The values for system generated fields are generated automatically by Adabas. It doesn't matter whether a record buffer for store and update commands contains values for system generated fields or not - field values for system generated fields in the record buffer are normally ignored, with the exception of using a last update time stamp field for optimistic locking as described below.

If the system generated fields are defined with option CR, the values are generated only during a store command and are never updated.

If the system generated fields are defined without option CR, the values are generated during store commands, and up-dated during update commands. If the field is a multiple-value field, a new first value is inserted and the old fields are shifted one element to the right - if this results in field values with MU index > file parameter SYFMAX, these fields are deleted.

### Using Last Update Time Stamp for Optimistic Locking

If you specify command option 2='T' (check time stamp) for an A1 command, you can use a last-update-time-stamp system generated field for optimistic locking:

- When you read the record to be updated, you either do not lock the record at all, with the disadvantage that you can see uncommitted updates (dirty read), or you lock it with option 'C', which guarantees you that you don't see uncommitted updates but doesn't keep the lock beyond the end of the read command. The read command reads a last-update-time-stamp system generated field (field with option SY=TIME without option CR).
- The update is performed by an A1 command with command option 'T' and must include the last-update-time-stamp field in the format and record buffer; the field value must not have been changed since the read command.
- If the last update time stamp is stored in an MU field, the field value with index 1 must be specified in the format buffer.

- If option 'T' was specified and no last-update-time-stamp system generated field is contained in the format buffer, then you receive response code 44, subcode 13.
- A check is made to see whether the field value specified in the record buffer is the same as the current value in the database record:
  - If the value in the database is the same value, and the current time in the precision of the field with option SY=TIME is still the same, the command fails with response code 22, subcode 42
  - If it is the same value, and the current time in the precision of the field with option SY=TIME has changed, the record is placed in hold status; if the record is held by another user, the behaviour depends on the usage of the return option:
    - If the return option (option 'R' or 'U') is specified, response code 145 is returned.
    - Otherwise the command waits until the record is available again; then the check is repeated.
  - If the system generated value has been changed in the meantime, you receive a response code 48, subcode 34. If you have specified option 'L' or 'U', the modified record is read into the record buffer.

**Note:** If the system generated field with option SY=TIME is not a high-precision time stamp, it is only guaranteed that no updates are lost if all updates on this file are performed with option 'T'. If you aren't sure, you should only use this kind of update with high-precision time stamps.

## Read Integrity

If you read a record with an L1, L2, L3 or S1 command, you perform a dirty read, i.e. you also see uncommitted updates. In order to see only the committed states of records, Adabas provides shared locks for the records. Depending on your requirements, you can keep the record in shared hold status for different times:

- The C option does not keep the record in shared hold status beyond the current read operation, unless it was already locked before.
- The Q option keeps the record in shared hold status until the next record of a read loop is read. It can be used to perform more than one consistent read operation on the same record.
- The S option keeps the record in shared hold status until the current transaction is committed or rolled back.
- The RI command enables you to release a record from shared hold status before the record would normally be removed from shared hold status.

**Note:** In order to use the C, Q or S options, you must change the commands to L4, L5, L6 or S4.

If you read different parts of a record with more than one read command, for example, if you read a LOB field piece-meal, it is strongly recommended to ensure that the record is not updated between the read operations. The following possibilities are available for this purpose:

- Place the record in shared or exclusive hold status with command option 3 = Q, S or blank when you perform the first read command for the record.
- Read a high-precision, system-generated last update time stamp with each read command, and check that it was not changed between the read commands. The advantage of this solution is that you can also read the record when somebody else has put the record in exclusive hold status - but you must be aware that this is still a dirty read, i.e. you can see uncommitted updates.
- The application guarantees the read integrity; for example, you can read a record without locking it, when another user is only allowed to update the record after locking another record, which you have already locked. Note that you should only use this kind of integrity preservation if you can really be sure that all programs updating the file apply these rules.

## Features in the Adabas Command Interface for Large Object (LOB) Support

The following features have been introduced to the Adabas command interface, for LOB support in particular:

### Large Buffers with the ACBX Interface

With the old ACB interface, the size of record buffers was limited to 64 KB, which is too small for LOB support. This limitation was removed with the new ACBX interface, which allows record buffer sizes of up to 2 GB.

### More than one Format/Record Buffer Pair with the ACBX Interface

When you access LOB fields and other fields with one Adabas call, the LOB value often has a variable length, and you want to be able to allocate the space in a special area, while the other fields have a fixed length, and which are located in a predefined area. The ACBX interface allows you to specify both areas as different record buffers for the same Adabas call.

### 4-Byte Length Indicator for LOB Fields when reading with variable Length

If you specify 0 as the length in the format buffer to indicate variable length for LOB fields (fields with field option LB), a 4-byte length indicator is used in the record buffer in order to support larger length values.

### Length Indicator in Format/Record Buffer

When you want to read a LOB value, you often don't know the length of the LOB value in advance. The length indicator allows you to retrieve the length of the LOB value first, before you allocate the area for the LOB value.

### Asterisk (*) Length Notation in the Format Buffer

Specifying an asterisk (*) as length reads a value with variable length. The difference to specifying 0 as length is as follows:

- The value is a value without the length in front. It is recommended for read commands and required for insert/update commands that you also specify a length indicator for the LOB field.
- You don't get a record buffer overflow (Adabas response code 53) when you read a value that doesn't fit into the record buffer. If you read the length indicator and the LOB value with * length, you still get the LOB value length even if the LOB value does not fit into the record buffer, and you can reallocate the record buffer with a sufficient size and reread the LOB value.

### Accessing Parts of LOBs with the Segment Notation in the Format Buffer

If the available memory is limited, or if you only need a part of a LOB where you know its position within the LOB value, and you want to access large LOB values, you may want to access the LOB value in pieces. This is possible using the segment notation in the format buffer.

**Note:** You can find Detailed information on the format buffer entries mentioned above in the section *Calling Adabas, Format and Record Buffers.*

# 6 ADABAS COMMANDS

This chapter contains a detailed description of each Adabas command. The commands are presented in alphabetical order for ease of reference.

The user is referred to the *Adabas Messages and Codes* for an explanation of all response codes which may result from Adabas commands.

The information is organized under the following headings:

| | |
|---|---|
| A1 Command: Update Record | Update record(s) (with hold option) |
| BT Command: Back Out Transaction | Remove database updates for ET logic users |
| C1 Command: Write a Checkpoint | Write checkpoint identifier, data protection log number and block number |
| C5 Command: Write User Data to Protection Log | Write user data to PLOG |
| CL Command: Close User Session | End/ET session and update database |
| E1 Command: Delete Record / Refresh File | Delete record or refresh file |
| ET Command: End Transaction | End and save current transaction |
| HI Command: Hold Record | Prevent record update by other users |
| L1 / L4 Command: Read Record | Read record of specified ISN<br>Read and hold option |
| L2 / L5 Command: Read Physical Sequential | Read records in physical order<br>Read in physical order and hold option |
| L3 / L6 Command: Read Logical Sequential | Read records in descriptor value order<br>Read in descriptor value order with hold option |
| L9 Command: Read Descriptor Values | Read the values of a specified descriptor |
| LF Command: Read Field Definitions | Read the characteristics of all fields in a file |
| MC Command: Multi-Call | Reduce interprocess communications between an Adabas database and its application programs |

| | |
|---|---|
| **N1 / N2 Command: Add Record** | Add new database record with ISN assigned by Adabas<br>Add new database record with user-assigned ISN |
| **OP Command: Open User Session** | Open user session |
| **RC Command: Release Command ID or Global Format Buffer ID** | Release one or more command IDs or a global format buffer ID for the issuing user |
| **RE Command: Read ET User Data** | Read ET data for this, another, or all users |
| **RI Command: Release Record** | Release held record and ISN<br>Release all records currently held by user |
| **S1 / S2 / S4 Command: Find Records** | Return count and ISNs of records satisfying the search criterion<br>Return count of records and ISNs in user-specified order |
| **S8 Command: Process ISN Lists** | Combine two ISN lists from the same file with an AND, OR, or NOT operation |
| **S9 Command: Sort ISN List** | Sort ISN list in ascending ISN or descriptor-specified sequence |

Please note that only those control block fields relevant to the command in question are shown in the control block. Also, in order to be compatible with new versions of Adabas, Software AG strongly recommends that you set all unused control block fields to their default values.

If errors occur while using the ACBX interface, Adabas may return additional information in the ACBXERRx fields, although this is not documented in the command documentation. Please refer to the *Nucleus Response Codes* in *Messages and Codes* for more information about this additional information.

If an Adabas call using the ACBX interface is made that requires buffer specifications, Adabas buffer descriptions (ABDs) must be used instead of the ACB fields for format buffer length, record buffer length, search buffer length, value buffer length and ISN buffer length.

# 7 A1 Command (Record Update)

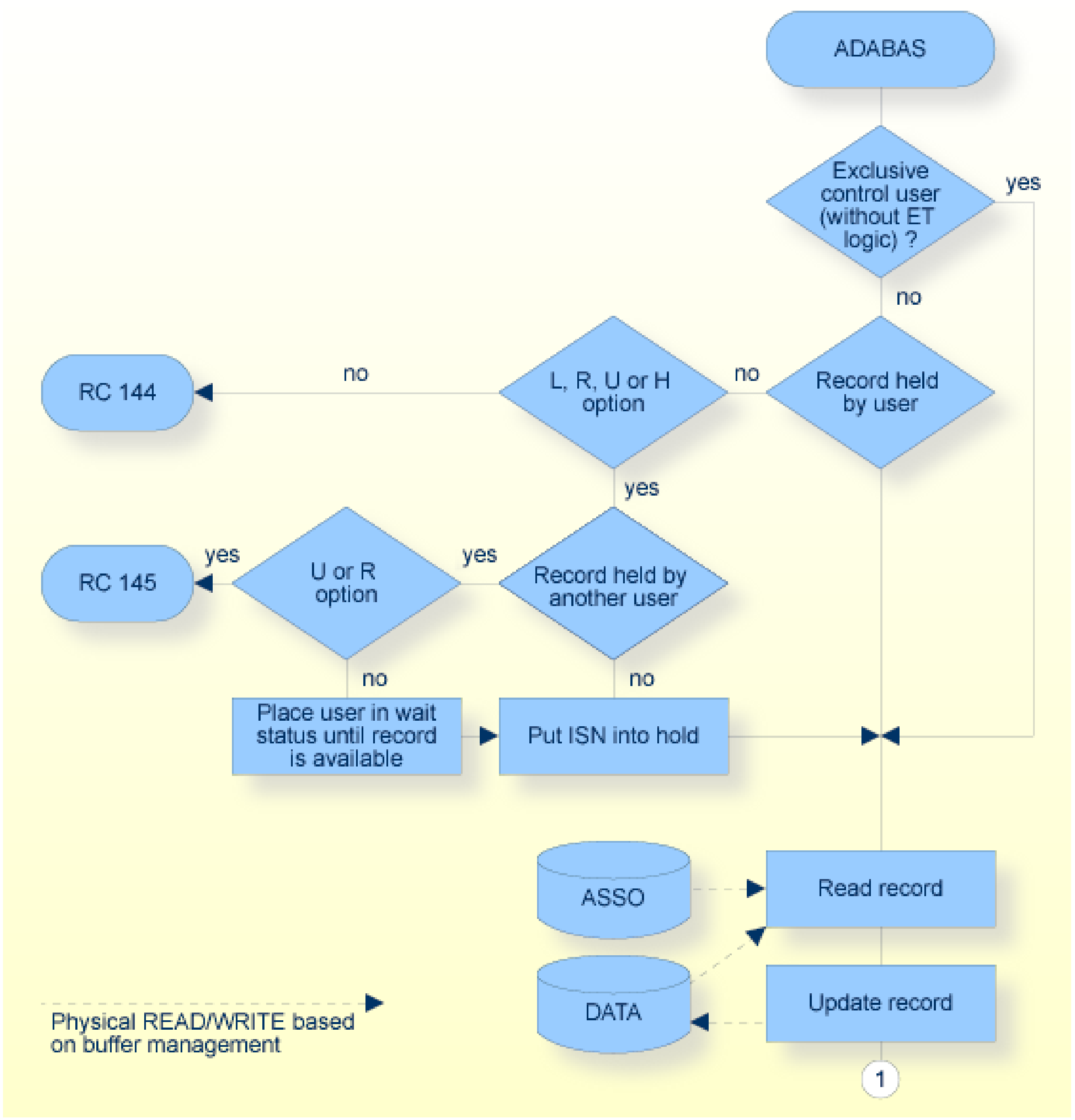
## Function and Use

The A1 command is used to modify the value of one or more fields within a record. The record containing the field (or fields) to be updated is identified by the file number in which it is contained and its ISN. The user specifies the fields to be updated in the format buffer, and provides the values to be used for updating these fields in the record buffer. Only those fields specified and system generated fields defined without the option CR will be modified. All other fields in the record remain unchanged.

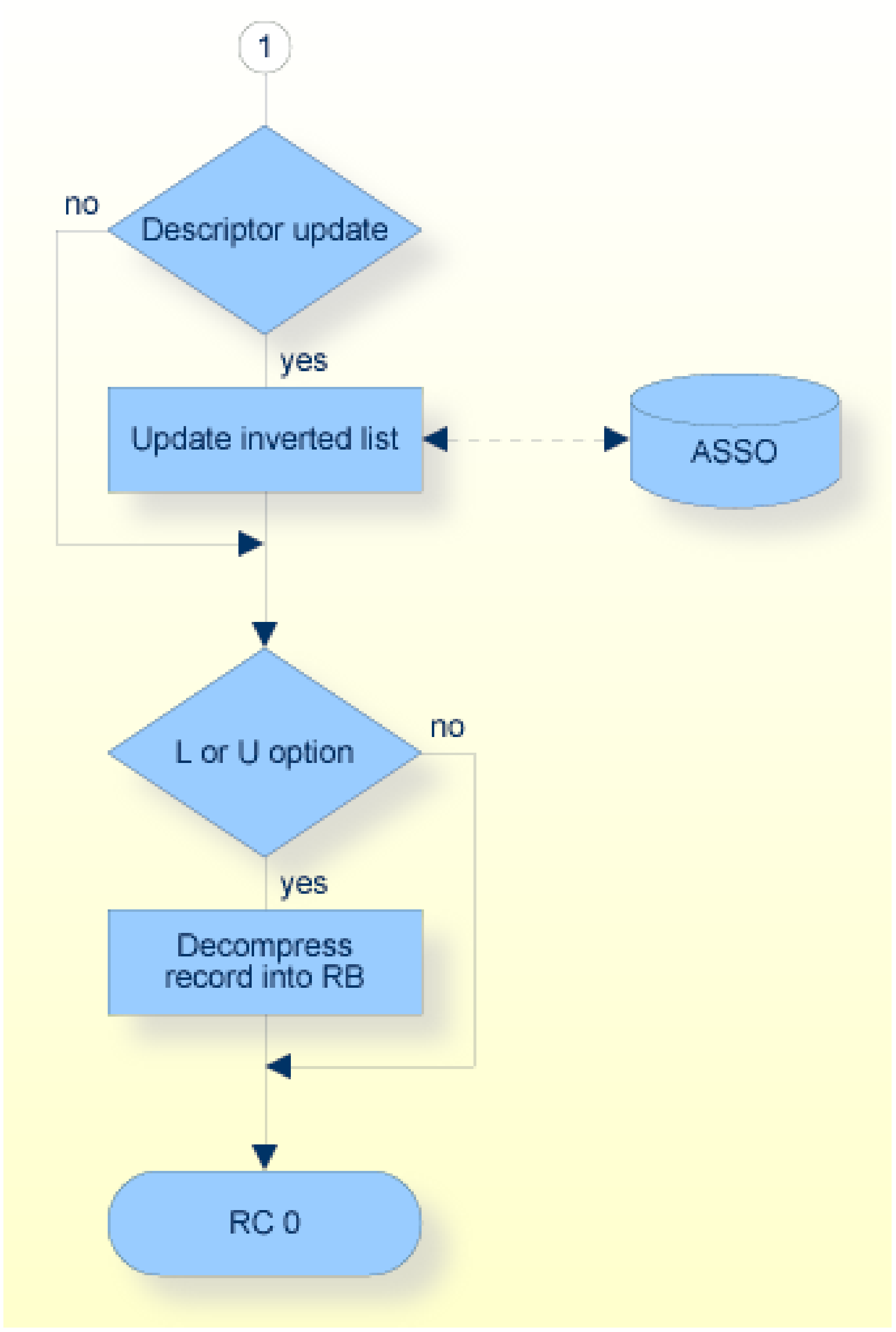
All necessary updating to the Associator and Data Storage is performed by Adabas. If one or more descriptors are updated, Adabas will update the inverted lists to reflect the modifications. If a field which was used to derive a subdescriptor or a superdescriptor is updated, Adabas will update the corresponding subdescriptor or superdescriptor values to reflect the modification.

In case of an ET user, the A1 command will be executed only if the record to be updated is in hold status for the user or when command option 1 = H, L, U or R was specified.

The user can also position to a record via an ADAM (Adabas direct access method) key. In this case, the command option 2 must be set to V, and the search and value buffers must contain the ADAM descriptor name and the ADAM key respectively. If this method is used, the ISN field in the control block becomes an output field.


ADABAS
Exclusive control user (without ET logic) ?
yes
no
Record held by user
no
L, R, U or H option
no
RC 144
yes
Record held by another user
yes
U or R option
yes
RC 145
no
no
Place user in wait status until record is available
Put ISN into hold
ASSO
DATA
Read record
Update record
1
Physical READ/WRITE based on buffer management

**A1 Command, Procedure Flow**



**A1 Command, Procedure Flow (continued)**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN | B | F/U |
| ISN Lower Limit | B | F/A |
| Format Buffer Length (ACB only) | B | F/U |
| Record Buffer Length (ACB only) | B | F/U |
| Search Buffer Length (ACB only) | B | F/U |
| Value Buffer Length (ACB only) | B | F/U |
| Command Option 1 | A | F/U |
| Command Option 2 | A | F/U |
| Additions 2 | A,B | -/A |
| Additions 3 | A | F/A |
| Additions 5 | A | F/U |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | F/U |
| Record Buffer | F/A |
| Search Buffer | F/U (2) |
| Value Buffer | F/U (2) |
| ISN Buffer | –/– |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.
(2) Only required if the V option is used

## Control Block

**Command Code**
A1

**Command ID**
If a series of records is to be updated by using a series of A1 calls, and the same fields are specified in the format buffer for each call (such as when updating a set of records resulting from a FIND command), this field should be set to a non-blank, non-zero value. If the A1 is used in conjunction with the L1/L4, L2/L5, or L3/L6 command, and the same fields within each record are read and updated, the same Command ID which was used for the READ command should be used for the A1 calls. In both cases, this reduces the time required to process each successive A1 call.

If only a single record is to be updated with a single A1 call, or the format buffer is modified between A1 calls, this field should be set to blanks or binary zero.

The high-order byte of this field may not be set to hexadecimal `FF', except when automatic command ID generation is used (see *Programming Considerations, Using Command IDs* for additional information).

**File Number**
The number of the file which contains the record to be updated.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**ISN**
The ISN of the record to be updated.

If the ADAM option is used, Adabas returns the ISN of the record found in this buffer.

**ISN Lower Limit**
If command option ‘L’ is used, the format buffer must contain a segment with *-position, the input value contains the current position of the segment. The value is increased by the length specified for the segment.

**Format Buffer Length (ACB only)**
The format buffer length (in bytes). The format buffer area defined in the user program must be as large as (or larger than) the length specified.

**Record Buffer Length (ACB only)**
The record buffer length (in bytes). The record buffer area defined in the user program must be as large as (or larger than) the length specified.

**Search Buffer Length (ACB only)**
The search buffer length (in bytes). The search buffer area defined in the user program must be as large as (or larger than) the length specified.

**Value Buffer Length (ACB only)**
The value buffer length (in bytes). The value buffer area defined in the user program must be as large as (or larger than) the length specified.

**Command Option 1**
When the user supplies an `H', the record is placed into exclusive hold status before the update command is executed.

When the user supplies an `R', the record is placed into exclusive hold status before the update command is executed. If the ISN is held by another user, a response 145 is returned.

When the user supplies an `L', the current updated record is returned in the record buffer according to the current format buffer. This option implies the H option.

When the user supplies a `U', Adabas attempts to place the record is placed into exclusive hold status. The command will not wait if an ISN conflict is detected (ISN in hold for another user), and a response 145 is returned. The current updated record is returned in the record buffer.

The `L' and the `U' options are only useful when used in conjunction with the add option or system generated fields (without CR) in the format buffer.

When the user supplies an ‘X’, the A1 command is equivalent (but more efficient) to an E1 command for the ISN specified followed by an N2 command for the ISN specified with the format and record buffers specified, with the exception that fields with field options SY and CR remain unchanged.

**Note:** The ‘X’ option is helpful for reducing the number of MU values or PE groups in a record.

**Command Option 2**

When the user supplies an `H', the ISN is placed into the hold queue before the update command is executed. The command waits if the ISN is held by another user unless the `R' option is specified in Command Option 1, in which case a response 145 is returned.

When the user supplies an 'L', the format buffer is only allowed if the format buffer contains a segment with *-position. Then the ISN Lower Limit field is used to determine the current position of the segment with *-position. Without the option 'L', the current position of a segment with *-position is the first (leftmost) byte of the field value.

When the user supplies a 'T', the record is handled according to the rules described in the section *Programming Considerations, System Generated Fields*.

A `V' in this field indicates that the ADAM option is being used. A value (ADAM key) can only be supplied for ADAM descriptor files. The value for the key is given in the value buffer, and the descriptor name is specified in the search buffer. If the ADAM option is used, the ISN field becomes an output field.

A response 52 is returned if no record with specified ADAM key is found.

The following table shows the possible combinations of the command options 1 and 2, together with the hold status required and conflict information:

| CO1 | CO2 | Hold Status | Conflict | Specials |
|---|---|---|---|---|
| ' ' | ' ' | Requires hold | | |
| 'H' | ' ' | Set hold | Wait for ISN | |
| ' ' | 'H' | Set hold | Wait for ISN | |
| 'L' | ' ' | Set hold | Wait for ISN | Read record |
| 'U' | ' ' | Set hold | Rsp 145 | Read record |
| 'R' | ' ' | Set hold | Rsp145 | |
| 'R' | 'H' | Set hold | Rsp 145 | |
| 'R' | 'V' | Set hold | Rsp 145 | ADAM |
| ' ' | 'V' | Requires hold | | ADAM |
| 'H' | 'V' | Set hold | Wait for ISN | ADAM, read record |
| 'L' | 'V' | Set hold | Wait for ISN | ADAM, read record |
| 'U' | 'V' | Set hold | Rsp 145 | ADAM, read record |

**Additions 2**

If the response code is 0, Adabas returns the compressed record length of the updated record in this field. The length is provided in the first two bytes in binary format. If the `L' or `U' option is used, the last two bytes will contain the length of the compressed fields selected by the format buffer in binary format.

**Additions 3**

This field is used to provide a security password.

If the file to be used is not security protected, this field should be set to blanks. If the file is security protected, the user must provide a valid password.

Adabas sets this field to blanks during command processing to protect the integrity of any password provided.

**Additions 5**

This field may be used to provide a separate format buffer ID that is used to identify the internal format buffer used for this command, or to provide a global format buffer ID.

As long as the first byte of the Additions 5 field is not alphanumeric, the value provided in the command ID field will also be used as the format buffer ID.

If the first byte is a lower case character, the bytes 5 to 8 of the Additions 5 field will be used as the separate local format buffer ID.

If the first byte is a digit or an upper case character, the Additions 5 field (8 bytes) will be used as a separate global format buffer ID, which means that the format buffer ID can be used by several users in parallel.

See *Programming Considerations, Using Command IDs* for additional information and examples.

## Format Buffer

The fields to be updated must be specified in this buffer.

The syntax and examples of format buffer construction are provided in *Calling Adabas, Format and Record Buffers*.

## Record Buffer

The values to be used for updating are provided in this buffer according to the length and format as specified in the format buffer.

Examples of record buffer construction are provided in *Calling Adabas, Format and Record Buffers*.

## Search Buffer

If the ADAM option (command option 2=V) is used, the search buffer contains the field specification of the ADAM descriptor. A response 61 is returned if the field name is not an ADAM key.

## Value Buffer

If the ADAM option (command option 2=V) is used, the value buffer contains the value of the ADAM key.

## Additional Considerations

The following additional considerations are applicable for the A1 command:

1. Subdescriptors, superdescriptors, hyperdescriptors and phonetic descriptors may not be updated directly. In order to update any of the above, the field(s) used to derive the subdescriptor, superdescriptor, or phonetic descriptor must be updated. All corresponding subdescriptor, superdescriptor, or phonetic descriptor values will then be updated automatically by Adabas.
2. The maxiumum record length after compression (including record ISN) is the maximum available DATA storage block size - 4.
3. A descriptor value may not exceed 1144 bytes, unless the descriptor is defined with the TR option.
4. If a field is updated using a length override which exceeds the standard length (not permitted if the field is defined with the Fixed Storage option), all subsequent references to this field should specify the length which was used. If a subsequent reference uses the standard length, value truncation for alphanumeric fields or response code 55 for numeric fields may occur.
5. Field names without index may be specified more than once only if they are multiple-value fields. Multiple-value fields and fields in periodic groups with the same index may not be specified more than once. It is also forbidden to specify a group and a field in the group at the same time.
6. Numeric edit masks must not be specified in the format buffer.
7. A multiple-value count field or periodic group count field specified in the format buffer will be ignored by Adabas. The corresponding value in the record buffer will also be ignored. A literal in the format buffer will be ignored by Adabas. The corresponding positions in the record buffer will also be ignored.
8. If a multiple-value field is updated, Adabas automatically updates the multiple value field count, if necessary, according to the following rules:

- For a multiple-value field defined with the NU option, the count field is adjusted to reflect the current number of existing non-null values. Null values are completely suppressed.

```
Field Definition            01,MF,5,A,MU,NU

MF values before update     XXXXX,YYYYY
Format Buffer               MF4
Record Buffer               ZZZZZ
Result after update         XXXXX,YYYYY,ZZZZZ
                            MF count = 3

MF values before update     XXXXX,YYYYY,ZZZZZ
Format Buffer               MF2
Record Buffer               bbbbb (blanks)
Result after update         XXXXX,ZZZZZ
                            MF count = 2

MF values before update     XXXXX,ZZZZZ
Format Buffer               MF1-2
Record Buffer               bbbbbbbbbb (blanks)
Result after update         Values suppressed
                            MF count = 0
```

- For a multiple-value field defined without the NU option, the count is adjusted to reflect the current number of existing values (including null values).

```
Field Definition            01,MF,5,A,MU

MF values before update     XXXXX,YYYYY
Format Buffer               MF4
Record Buffer               DDDDD
Result after update         XXXXX,YYYYY,b(blank),DDDDD
                            MF count = 4

MF values before update     XXXXX,YYYYY,ZZZZZ
Format Buffer               MF3
Record Buffer               bbbbb (blanks)
Result after update         XXXXX,YYYYY,b (blank)
                            MF count = 3
```

9. If you specify MU fields without index, only the values contained in the record buffer are stored in the database; all other MU values contained in the old record are removed. If the current record buffer contains null values and the MU field is defined with the NU option, the null values are suppressed. The count field is adjusted accordingly.

```
Field Definition            01,MF,5,A,MU,NU

MF values before update     XXXXX,YYYYY
Format Buffer               MF
Record Buffer               AAAAA
Result after update         AAAAA
                            MF count = 1

MF values before update     XXXXX,YYYYY
Format Buffer               MF1
Record Buffer               AAAAA
Result after update         AAAAA,YYYYY
                            MF count = 2

MF values before update     XXXXX,YYYYY,ZZZZZ
Format Buffer               MF
Record Buffer               bbbbb (blanks)
Result after update         value suppressed
                            MF count = 0
```

10. If one or more fields contained in a periodic group are updated, Adabas automatically updates the periodic group count, if necessary, according to the following rule:

    - The count is adjusted to reflect the highest occurrence number referenced in the format buffer (provided that this occurrence is higher than the current highest occurrence number).

```
Field Definitions           01,GB,PE
                            02,BA,1,B,DE,NU
                            02,BB,5,P,NU

GB values before update     GB (1st occurrence)
                            BA = 5   BB = 20
                            GB (2nd occurrence)
                            BA = 6   BB = 25
                            GB count = 2
```

```
Format Buffer               GB4
Record Buffer               0x08000000500C
```

```
Result after update         GB (1st occurrence)
                            BA = 5   BB = 20
                            GB (2nd occurrence)
                            BA = 6   BB = 25
                            GB (3rd occurrence)
                            BA = 0   BB = 0
                            GB (4th occurrence)
                            BA = 8   BB = 500
                            GB count = 4

GB values before update     GB (1st occurrence)
                            BA = 5   BB = 20
                            GB (2nd occurrence)
                            BA = 6   BB = 25
                            GB count = 2
```

```
Format Buffer               GB1
Record Buffer               0x00000000000C
```

```
Result after update         GB (1st occurrence)
                            BA = 0   BB = 0
                            GB (2nd occurrence)
                            BA = 6   BB = 25
                            GB count = 2
```

11. If a field defined with variable length (no standard length) is specified in the format buffer, the corresponding value in the record buffer must be preceded by a one byte binary number which represents the length of the value (including the length byte).

```
Field Definitions           01,AA,3,A
                            01,AB,0,A

Format Buffer               AA,AB.
Record Buffer               "3132330631323334 35"
```

Fields AA and AB are to be updated. The new value for AA is 123. The new value for AB (which is a variable length field) is 12345.

## Example

**Note:** The Adabas file definitions in *Appendix A* are used in the examples in this section.

ISN 4 of file 1 is to be updated with the following values:

```
Field AA            1234
Field AB            20
```

**Control Block:**

```
Command Code          A1
```

```
Command ID            bbbb (blanks; only 1 record is to be updated)
```

```
File Number           1
```

```
ISN                   4
```

```
Format Buffer Length  10 (or larger)
```

```
Record Buffer Length  10 (or larger)
```

```
Additions 3           bbbbbbbb (blanks; file is not security protected)
```

**Buffer Areas:**

```
Format Buffer         AA,AB,2,U.
```

```
Record Buffer         0x2132333420202020203230
```

# 8 BT Command (Backout Transaction)

## Function and Use

The BT command is used for two purposes:

- Backout of a logical transaction (command option ‘S’ not specified);
- Backout of subtransactions (command option ‘S’ specified).

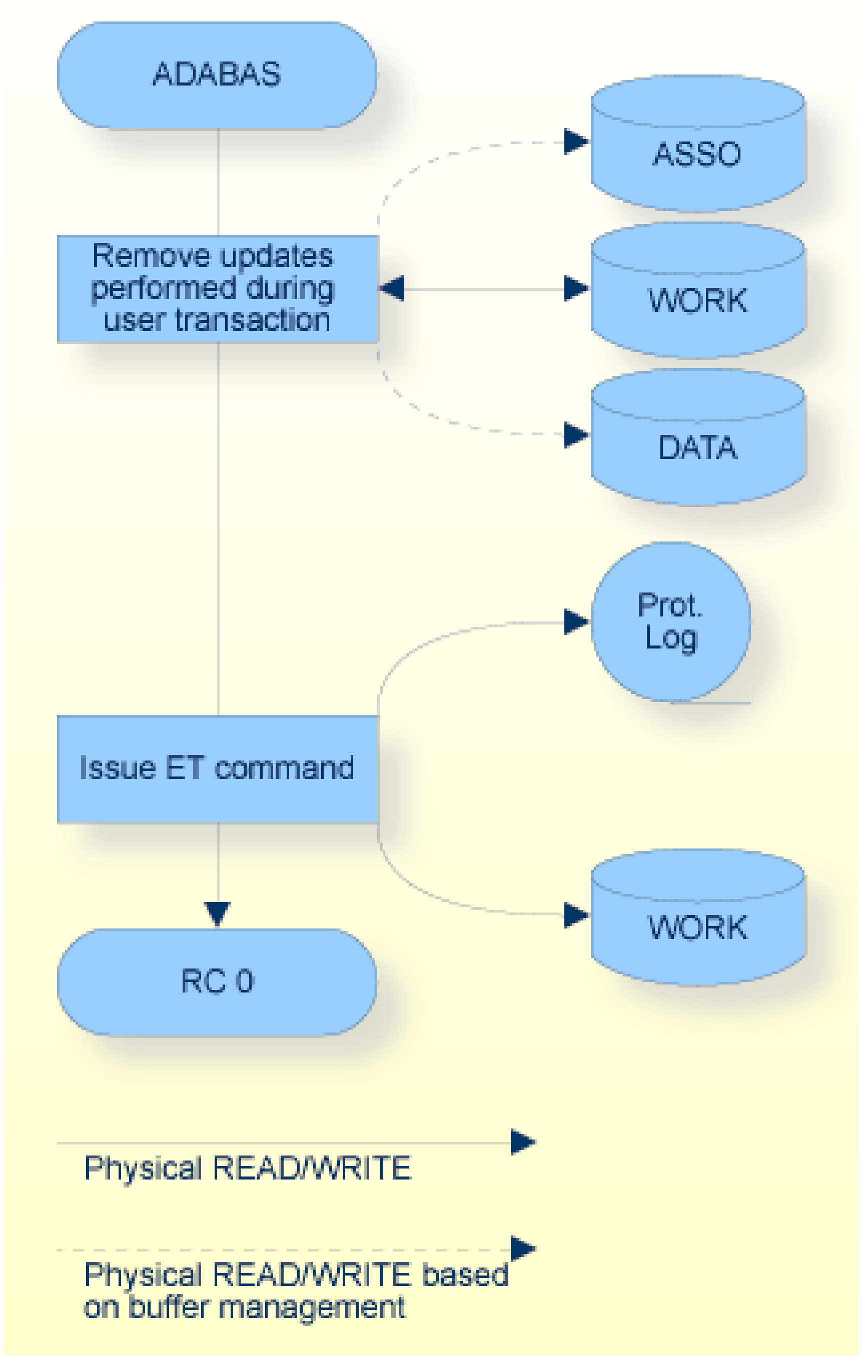
BT commands may only be issued by ET Logic users.

### Backout of a Logical Transaction

The BT command is used to remove all the database modifications (adds, deletes, updates) performed during the user's current logical transaction. This may be necessary because of a program error or when it is determined that the entire transaction cannot be successfully completed.

If command option ‘H’ is used, all exclusive locks of the user are downgraded to shared locks. Otherwise, without the multifetch option all records held by the user are released, or with the multifetch option all records specified in the ISN buffer are released.

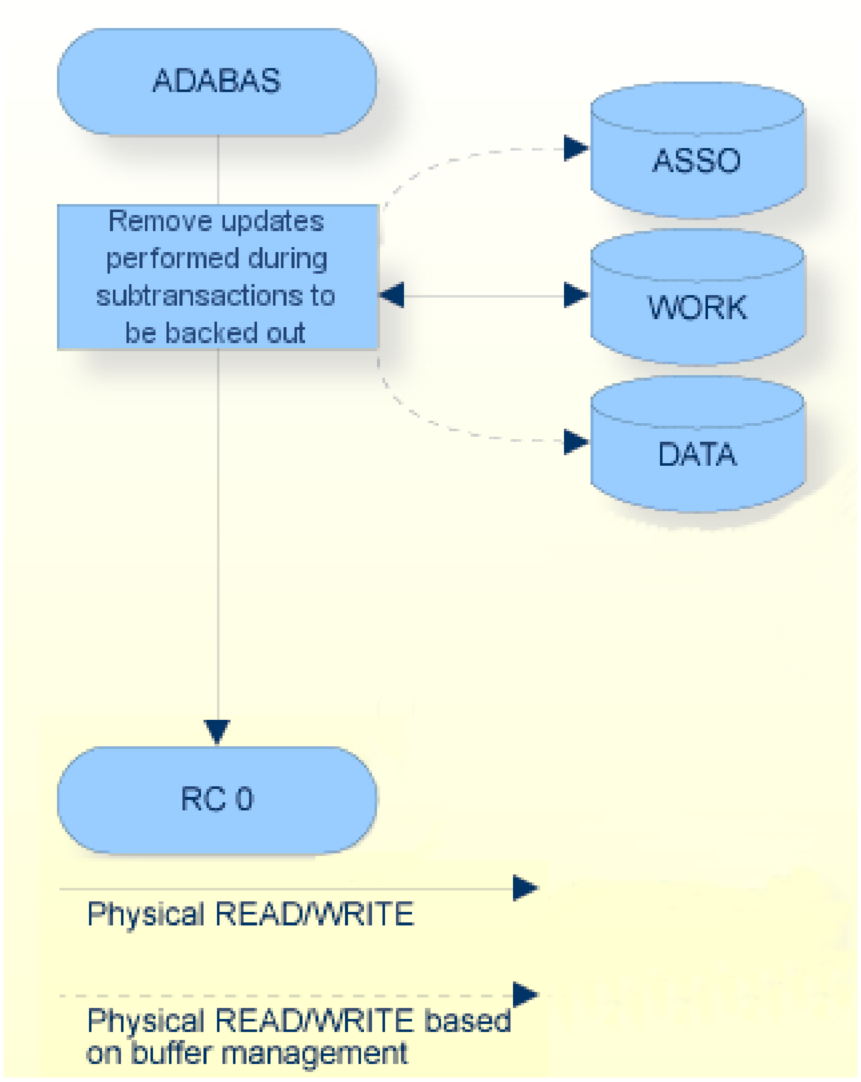
Adabas issues an implicit ET command as the last step in the processing of a BT command. This causes the current data protection block to be physically written to the Adabas work file and the data protection log, and releases all of the records which were held during the transaction.


ADABAS
ASSO
Remove updates performed during user transaction
WORK
DATA
Prot. Log
Issue ET command
WORK
RC 0
Physical READ/WRITE
Physical READ/WRITE based on buffer management

**BT Command, Backout of a Logical Transaction**

## Backout of a Subtransaction

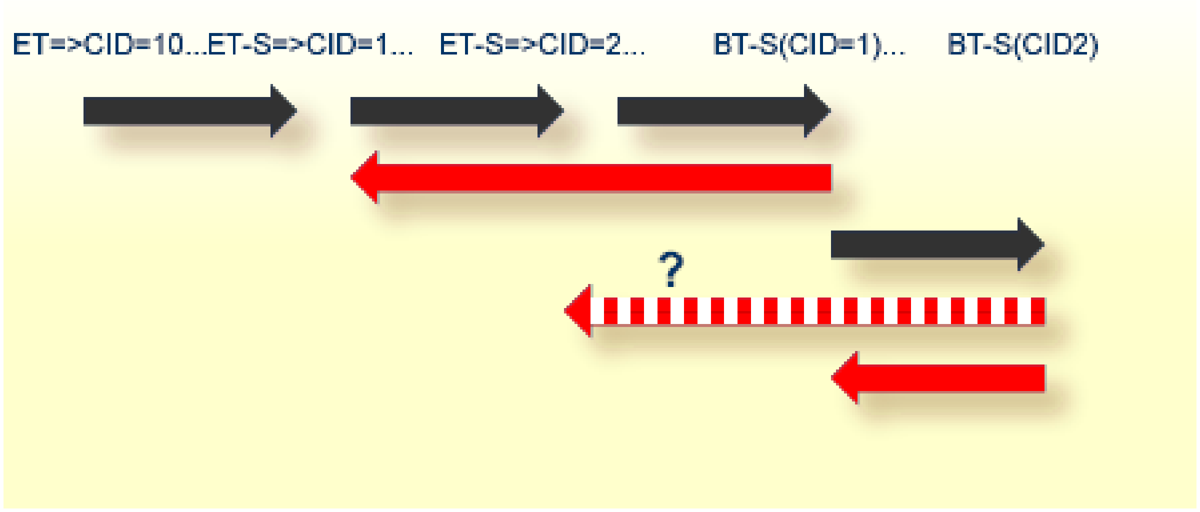
The BT command is used to remove all the database modifications performed during all subtransactions, starting at the savepoint with the savepoint ID specified in the command ID field of the control block.


ADABAS
ASSO
Remove updates performed during subtransactions to be backed out
WORK
DATA
RC 0
Physical READ/WRITE
Physical READ/WRITE based on buffer management

**BT Command, Backout of a Subtransaction**

### Error Situations for the Backout of a Subtransaction

- Command option 'S' is specified for the BT command, but subtransactions are not enabled for the current Adabas user session: response 22 is returned, the first 2 bytes of the Additions 2 field are set to 19.
- The CID specified for a BT subtransaction is not a savepoint ID belonging to the current transaction: response 21 is returned, the first 2 bytes of The Addition 2 field are set to 10.
- The CID specified for a BT subtransaction is a savepoint ID that no longer exists because of a BT subtransaction; as shown in following diagram:



In this case, the actions until the problematic subtransaction backout are rolled back; because of this subtransaction backout, the database content is the same as it was when ET-S=>CID=1 was issued. Because this is not the database content as it was when ET-S=>CID=2 was issued, response 2 is returned, the first 2 bytes of the Additions 2 field are set to 5.

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/A |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN Buffer Length (ACB only) | B | F/U |
| Command Option 1 | A | F/U |
| Command Option 3 (ACBX only) | A | F/U |

| Field | Format | |
|---|---|---|
| Additions 2 | A,B | -/A |
| Command Time | B | -/A |
| User Area | | F/U |

### Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | –/– |
| Record Buffer | –/– |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | F/U |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
BT

**Command ID**
Adabas returns in this field the transaction sequence number of the transaction which has been backed out. The number is returned in binary format. If the user was at ET status or has backed out a transaction without any updates, a 0 will be returned.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**ISN Buffer Length (ACB only)**
The ISN buffer length (in bytes). This buffer is only used in conjunction with the Multifetch feature. The value specified may not be smaller than 4 + (number of ISNs * 8), otherwise the Multifetch feature will be ignored.

**Command Option 1**
When the user supplies an 'S' in this field, a backout of a subtransaction is performed. Otherwise a backout of a logical transaction is performed. An `M' in this field invokes the Multifetch feature. See *Programming Considerations, Using the Multifetch Feature* for additional information about the Multifetch feature.

**Command Option 3 (ACBX only)**
If this field is set to 'H', all locked records remain in hold status, but exclusive locks are downgraded to shared locks.

If this field is set to a blank, all records currently locked are released from hold status.

**Additions 2**
For some response codes, Adabas returns detailed information in this field. See the *Adabas Messages And Codes* for further information.

## ISN Buffer

This buffer contains the ISNs and file numbers of the records that are to be released from hold status. This field is only used in conjunction with the Multifetch feature. If all of the records are to be kept in hold status, the first 4 bytes of this buffer must be set to zeros.

For a detailed layout description of the ISN buffer, see *Programming Considerations, Using the Multifetch Feature (Transaction Control)*.

# 9 C1 Command (Write a Checkpoint)

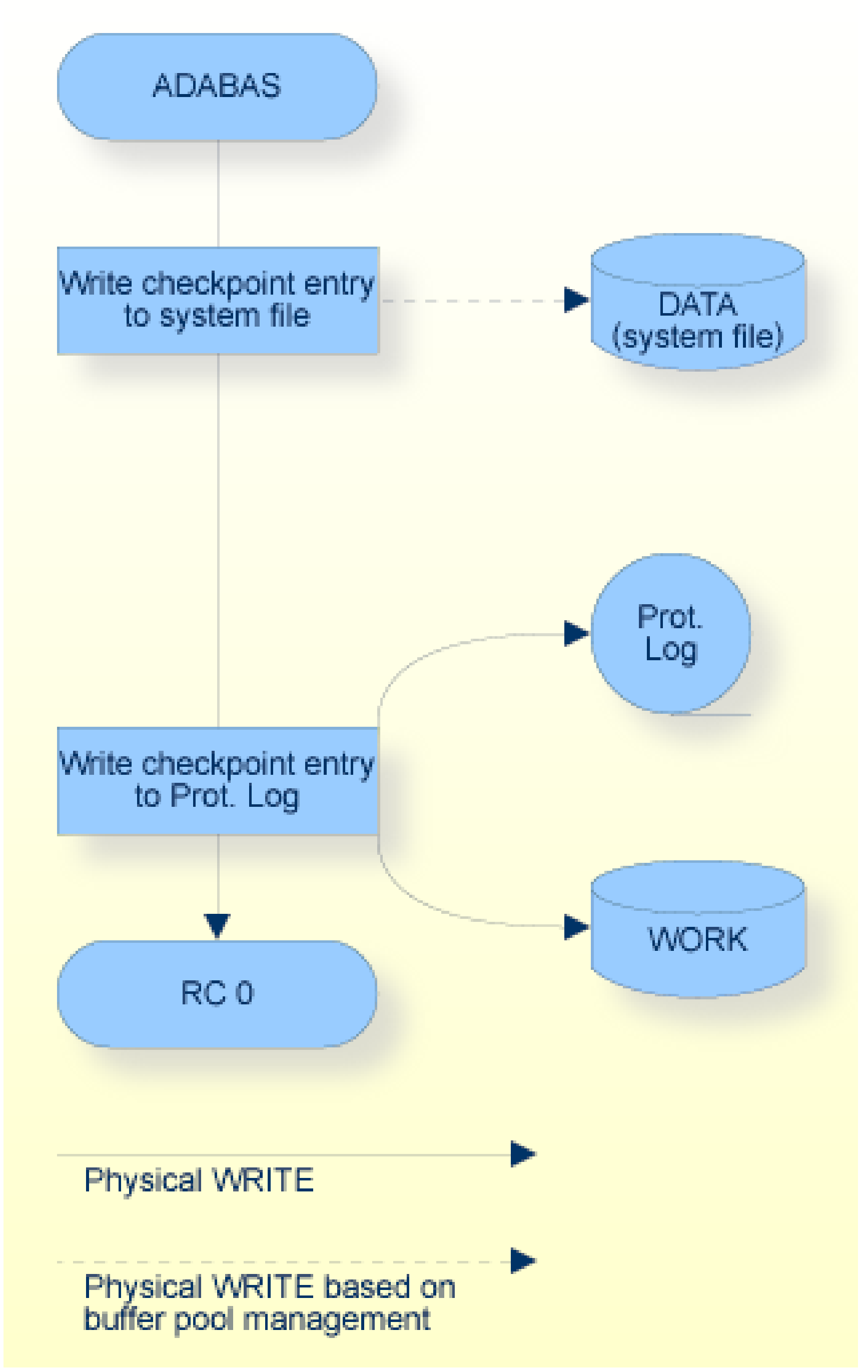
## Function and Use

The C1 command is used to request that a checkpoint be taken.

C1 commands are normally issued only by exclusive control update users who are not using ET Logic. Even if a C1 command is issued by an ET Logic user, it is not subject to ET Logic, which means that its effects will not be wiped out by a BT (backout transaction) command.

Adabas automatically executes a C1 command at the beginning of a user program in which exclusive file control updating has been requested.

The result of the C1 command is a checkpoint entry in the Adabas checkpoint table. This checkpoint entry contains the checkpoint identifier (the value provided by the user in the Command ID field), and the current data protection log and block number. This checkpoint entry may be used to restore the database (or certain files) to the status in effect at the time the checkpoint was taken. This may be necessary before a program that performs exclusive control updating can be rerun or restarted.


ADABAS
Write checkpoint entry to system file
DATA (system file)
Prot. Log
Write checkpoint entry to Prot. Log
WORK
RC 0
Physical WRITE
Physical WRITE based on buffer pool management

**C1 Command, Procedure Flow**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| Additions 2 | A,B | -/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | –/– |
| Record Buffer | –/– |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | –/– |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**

C1

**Command ID**

A non-blank, non-zero value must be entered in this field. This value will serve to identify the checkpoint taken. It is not necessary that each value provided for each checkpoint is unique.

A value of 'SYNx' must not be specified.

**Response Code**

Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**Additions 2**

For some response codes, Adabas returns detailed information in this field. See *Adabas Messages And Codes* for further information.

## Example

The user requests a checkpoint. The checkpoint is to be identified by the value USR4.

**Control Block:**

```
Command Code          C1
```

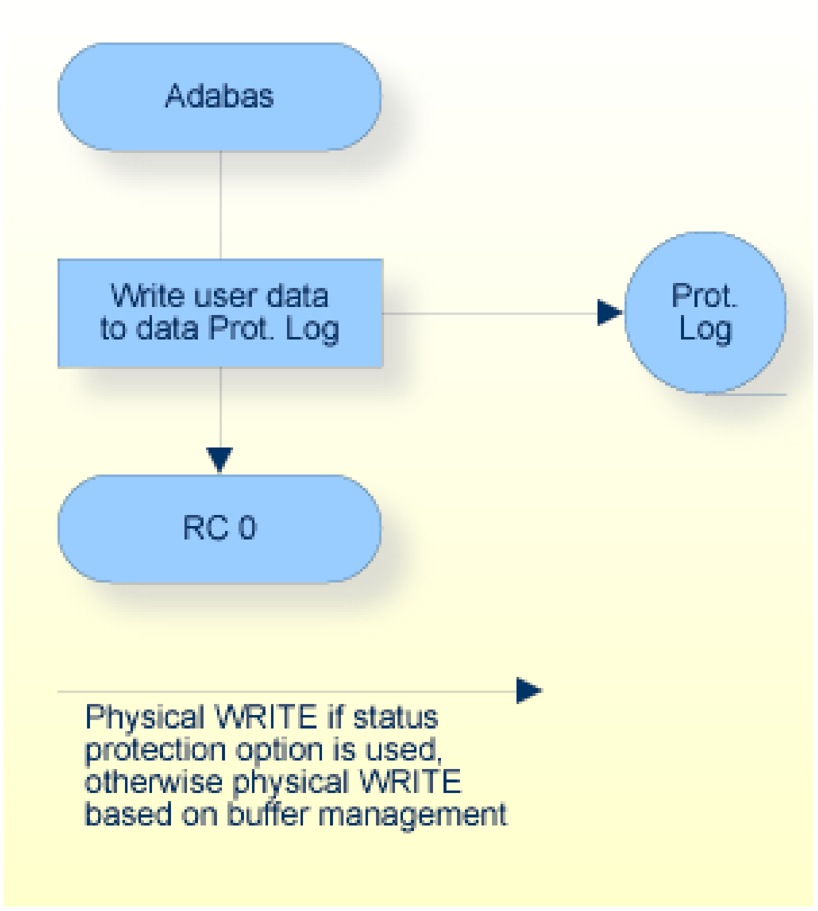
```
Command ID            USR4 (checkpoint identifier)
```

# 10 C5 Command (Write User Data to Protection Log)

## Function and Use

The C5 command is used to write user data to the Adabas protection log.

The data written have no effect on Adabas recovery processing. The recovery utility ADAREC ignores all data written to the data protection log as a result of a C5 command.


Adabas
Write user data to data Prot. Log
Prot. Log
RC 0
Physical WRITE if status protection option is used, otherwise physical WRITE based on buffer management

**C5 Command, Procedure Flow**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | -/- |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| Record Buffer Length (ACB only) | B | F/U |
| Additions 2 | A,B | -/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | */– |
| Record Buffer | F/U |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | –/– |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Ababas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |
| * | Not used but must be included in parameter list of CALL statement |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**

C5

**Response Code**

Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**Record Buffer Length (ACB only)**

The number of bytes specified in this field will be written to the Adabas data protection log.

The maximum length which may be specified is 2000 bytes.

**Additions 2**

For some response codes, Adabas returns detailed information in this field. See *Adabas Messages and Codes* for further information.

## Record Buffer

The information to be written to the data protection log is provided in this buffer.

The format of the data protection log is alphanumeric.

## Example

The information `ULRR0422 UPDATES FOR JANUARY' is to be written to the Adabas data protection log.

**Control Block**

```
Command Code         C5
```

```
Record Buffer Length  28
```

## Buffer Areas

```
Record Buffer        ULRR0422 UPDATES FOR JANUARY
```

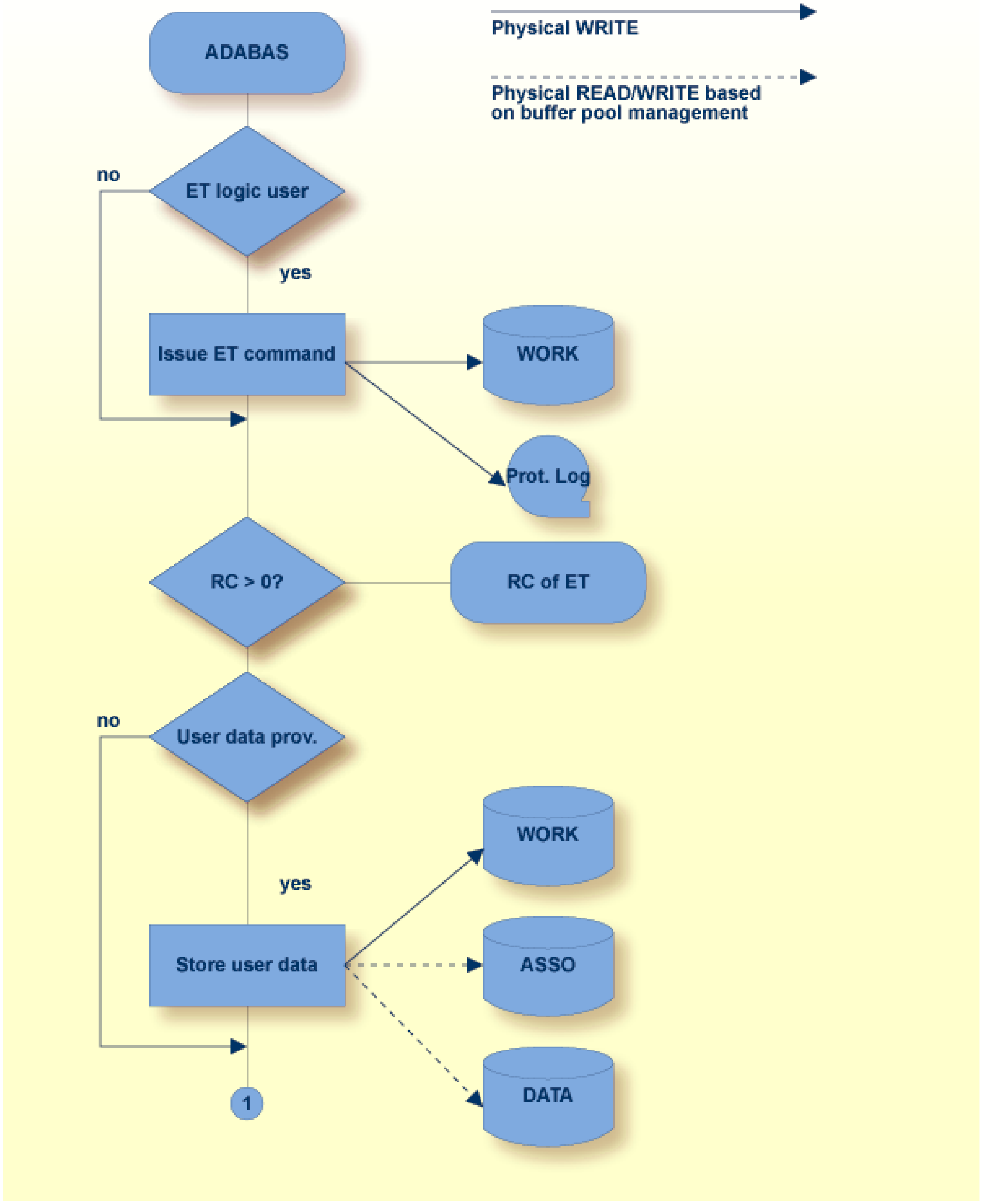
# 11 CL Command (Close User Session)

## Function and Use

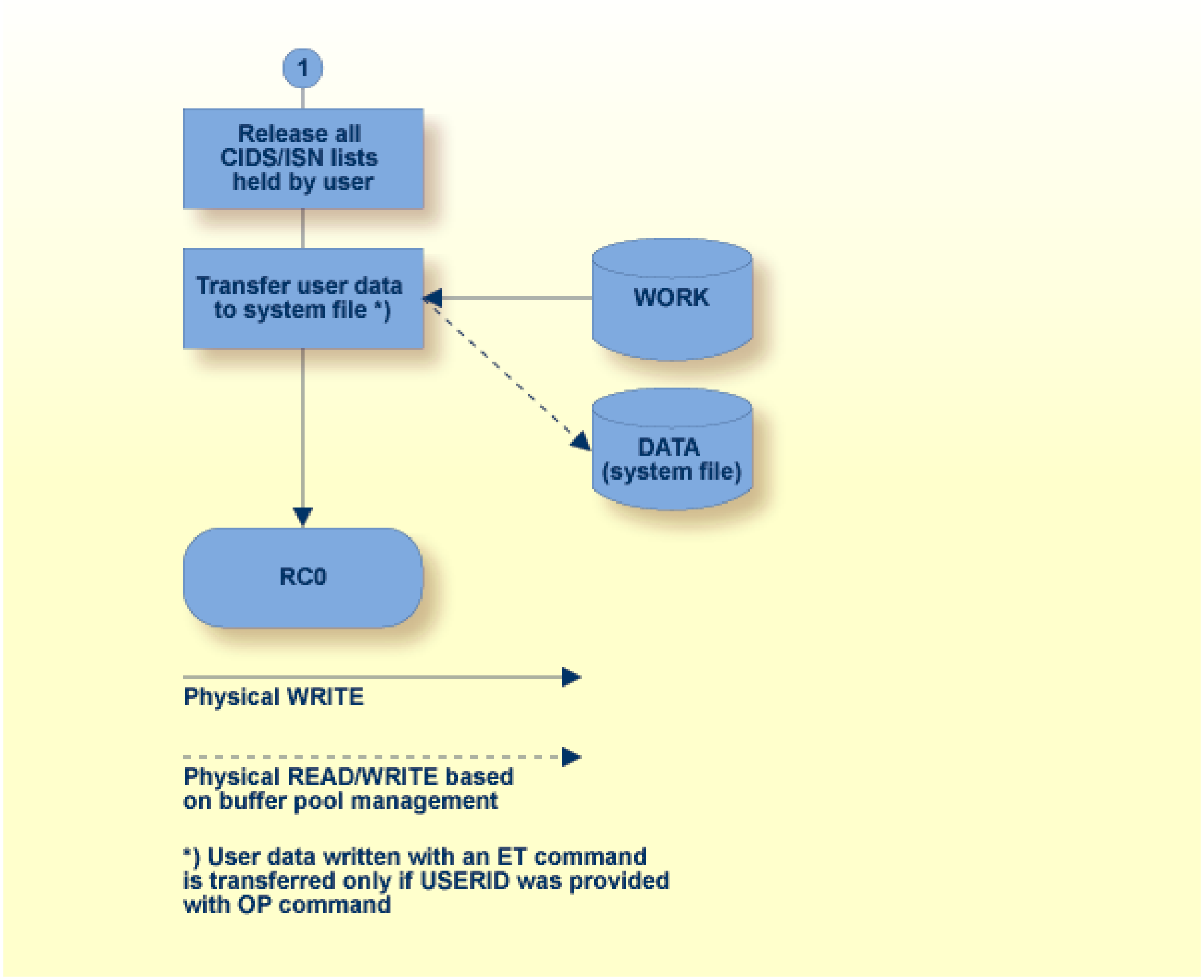
The CL command is used to terminate a user session. It is recommended that all user programs issue a CL command when database processing is complete.

A CL command results in:

- An implicit ET command (ET logic users only);
- The storing of user data in an Adabas system file (optional);
- The physical writing of the current data protection block to the Adabas data protection log and WORK container;
- The release of all records currently in hold status for the user, and the release of all Command ID entries (and corresponding ISN lists) assigned to the user;
- The transfer of the user's data from the Adabas WORK container to an Adabas system file. This is done only if a USERID was provided with the OP command; otherwise, any user data stored in the user–data system file (written by an ET command) during the session is not retained.



**CL Command, Procedure Flow**


1
Release all CIDS/ISN lists held by user
Transfer user data to system file *)
WORK
DATA (system file)
RC0
Physical WRITE
Physical READ/WRITE based on buffer pool management
*) User data written with an ET command is transferred only if USERID was provided with OP command

**CL Command, Procedure Flow (continued)**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | -/A |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| Record Buffer Length (ACB only) | B | $ F/U |
| Command Option 2 | A | F/U |
| Additions 2 | A,B | -/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | $ */– |
| Record Buffer | $ F/U |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | –/– |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |
| * | Not used but must be included in parameter list of CALL statement |
| $ | Only used if user data to be stored |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
CL

**Command ID**
If ET commands have been issued during the user session, Adabas will return the transaction sequence number of the user's last successfully executed transaction in this field. The number is provided in binary format.

If no ET command has been successfully executed during this session, this field is set to 0.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**Record Buffer Length (ACB only)**
If user data is to be stored in an Adabas system file, the length of the record buffer must be specified in this field. The length specified determines the number of bytes of user data to be stored.

The maximum length which may be specified is 2000 bytes.

If no user data is to be stored, this field is not used.

**Command Option 2**
An `E' in this field indicates that user data provided in the record buffer is to be stored in an Adabas system file.

**Additions 2**
For some response codes, Adabas returns detailed information in this field. See *Adabas Messages and Codes* for further information.

## Record Buffer

The user data which is to be stored is provided in this buffer. The number of bytes actually stored is determined by the value specified in the record buffer length field. The data will be retained only if the user has issued an OP command in which a non-blank USERID was provided. If so, the data will be retained until the user issues the next ET or CL command in which user data is provided. If a non-blank USERID was not provided, the data cannot be retrieved in a subsequent session.

The user data can be in any format and Adabas performs no conversion on it.

## Examples

**Example 1:**

The user program has completed all database activity and issues the CL command. No user data is to be stored.

**Control Block:**

```
Command Code          CL
```

```
Command Option 2      b (no user data is to be stored)
```

**Example 2:**

The user program issues a CL command and provides user data to be stored in an Adabas system file.

**Control Block:**

```
Command Code          CL
```

```
Record Buffer Length  17 (17 bytes of user data to be stored)
```

```
Command Option 2      E (user data is to be stored)
```

**Buffer Areas:**

```
Record Buffer         USER 7 NORMAL END
```

# 12 E1 Command (Delete Record)

## Function and Use

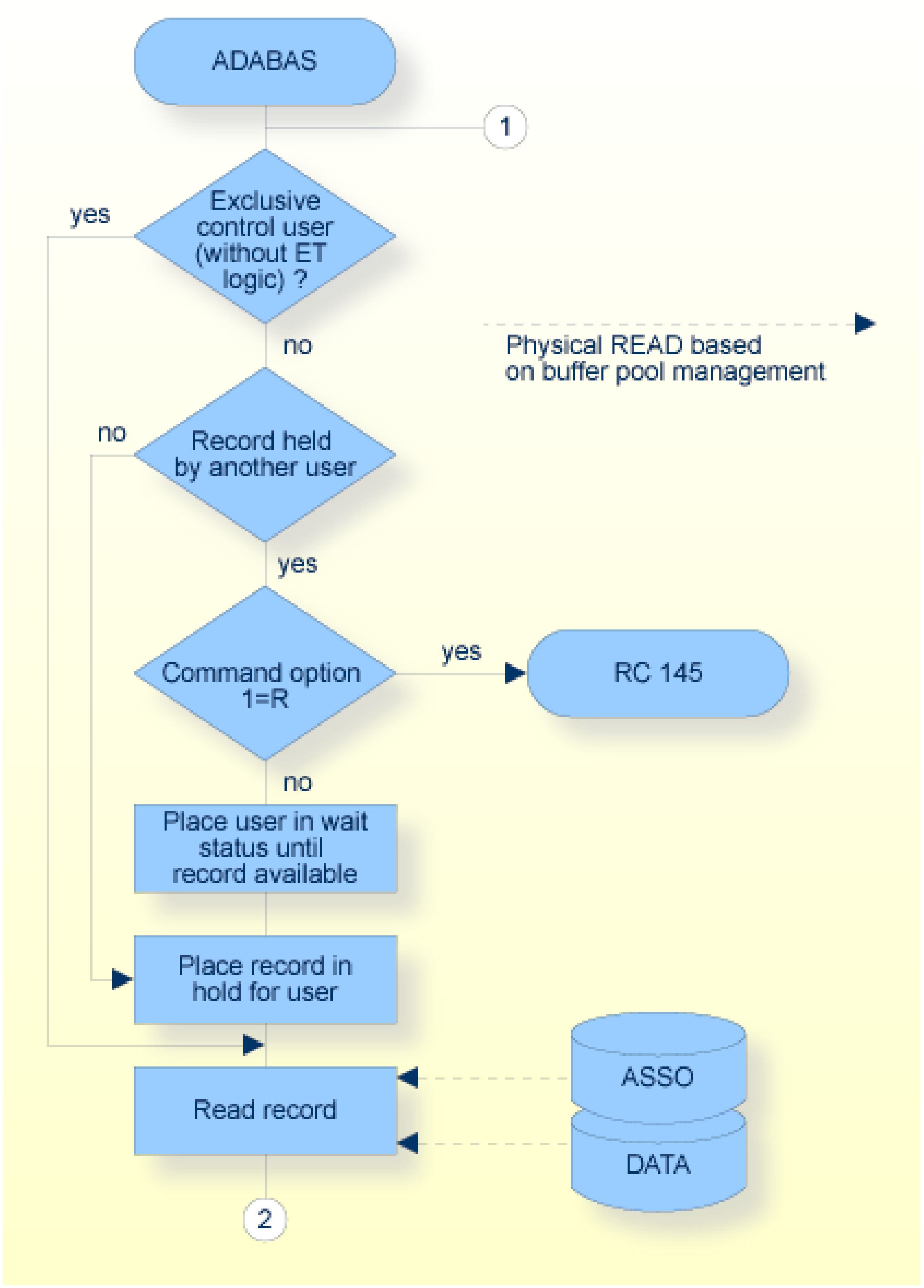
The E1 command is used to delete a record.

The user specifies the file number and ISN of the record to be deleted. Adabas deletes the record from Data Storage and makes any necessary updates to the Associator.

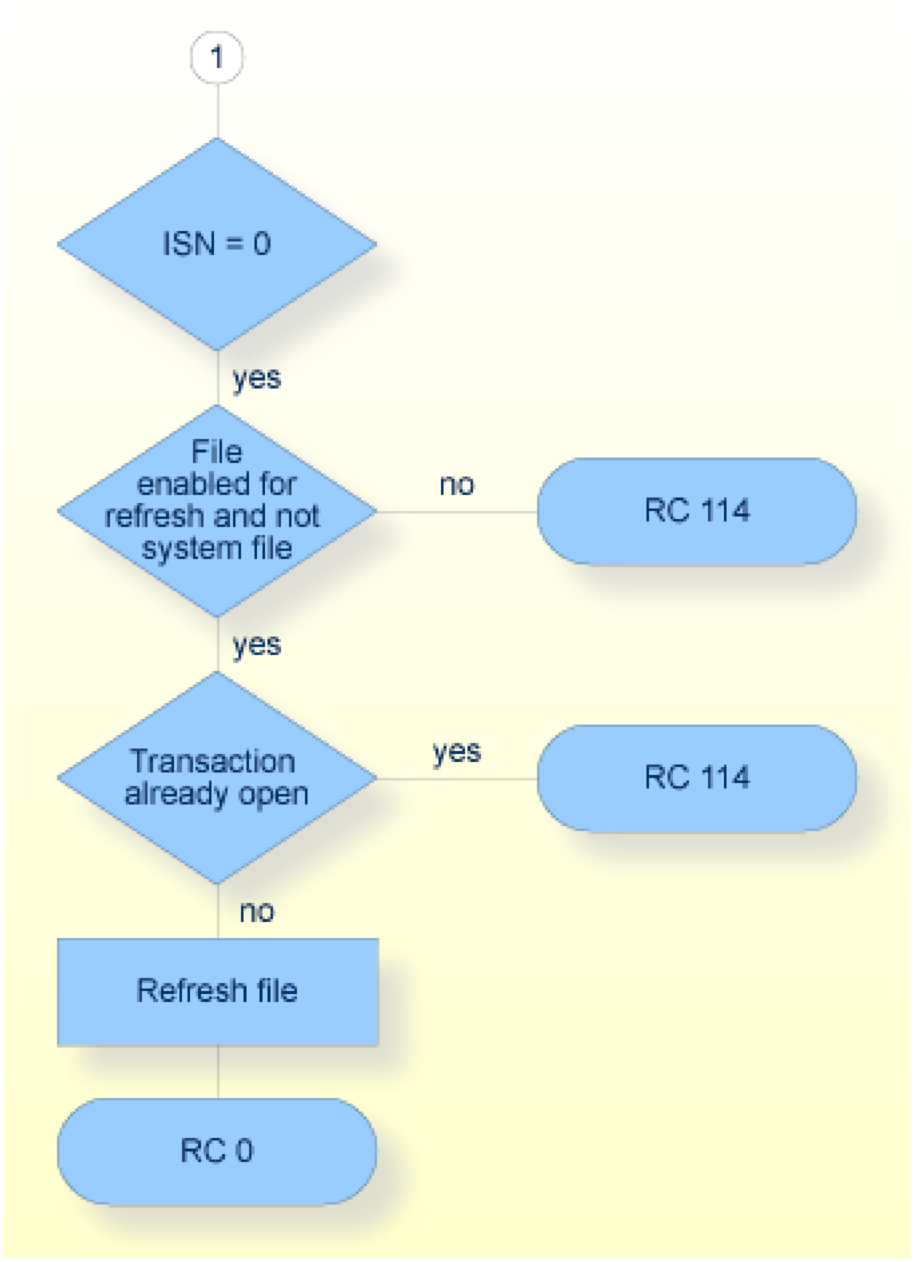
If the user is an ET user and the record to be deleted is not in hold status for the user, Adabas will place the record in hold status for the user. If the record is in hold status for another user, the E1 command will be suspended until the record becomes available, unless the return option is used, in which case response code 145 is returned.

The user can also position to a record via an ADAM (Adabas direct access method) key. In this case, the command option 2 must be set to V, and the search and value buffers must contain the ADAM descriptor name and the ADAM key respectively. If this method is used, the ISN field in the control block becomes an output field.

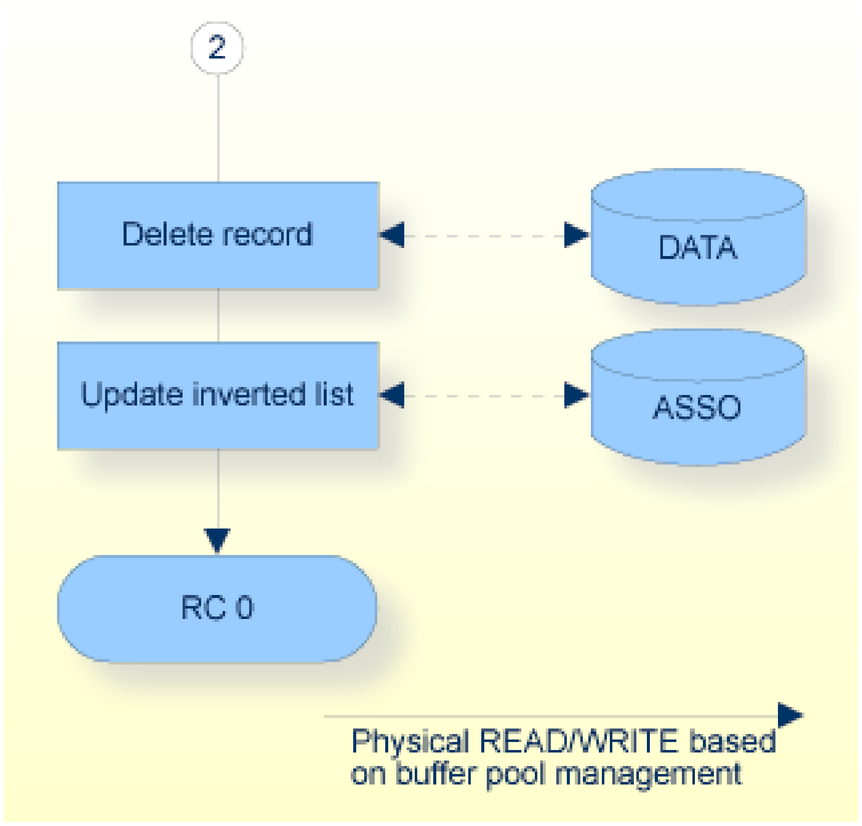
The E1 command can also be used to refresh a file - refreshing a file resets it to a state of zero records loaded. If a file is to be refreshed, the ISN field in the control block must be set to zero, and the command ID field must be blank.


ADABAS
1
yes
Exclusive control user (without ET logic) ?
no
Physical READ based on buffer pool management
no
Record held by another user
yes
Command option 1=R
yes
RC 145
no
Place user in wait status until record available
Place record in hold for user
Read record
ASSO
DATA
2

**E1 Command, Procedure Flow**



**E1 Command, Procedure Flow (continued)**



**E1 Command, Procedure Flow (continued)**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN | B | F/U |
| Search Buffer Length (ACB only) | B | F/U |
| Value Buffer Length (ACB only) | B | F/U |
| Command Option 1 | A | F/U |
| Command Option 2 | A | F/U |
| Additions 2 | A,B | -/A |
| Additions 3 | A | F/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | –/– |
| Record Buffer | –/– |
| Search Buffer | F/U (2) |
| Value Buffer | F/U (2) |
| ISN Buffer | –/– |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.
(2) Only required if the V option is used

## Control Block

**Command Code**
E1

**Command ID**
The command ID must be set to blanks if a file is to be refreshed.

**File Number**
The number of the file which contains the record to be deleted.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**ISN**
The ISN of the record to be deleted.

If the ADAM option is used, Adabas returns the ISN of the record deleted.

If the file is to be refreshed (reset to a state of zero records loaded), this field must be set to zero.

**Search Buffer Length (ACB only)**
The search buffer length (in bytes). The search buffer area defined in the user program must be as large as (or larger than) the length specified.

**Value Buffer Length (ACB only)**
The value buffer length (in bytes). The value buffer area defined in the user program must be as large as (or larger than) the length specified.

**Command Option 1**
An `R' in this field indicates that the return option is to be used. If an E1 command is issued, and the record to be deleted is currently held by another user, Adabas will return response code 145 rather than placing the user in wait status until the record becomes available.

**Command Option 2**
A `V' in this field indicates that the ADAM option is being used. A value (the ADAM key) can only be supplied for ADAM descriptor files. The value for the key is given in the value buffer, and the descriptor name is specified in the search buffer. If the ADAM option is used, the ISN field becomes an output field.

Response code 52 is returned if no record with the specified ADAM key is found.

**Additions 2**

For some response codes, Adabas returns detailed information in this field. to the *Adabas Messages and Codes* for further information.

**Additions 3**

This field is used to provide a security password.

If the file to be used is not security protected, this field should be set to blanks. If the file is security protected, the user must provide a valid password.

Adabas sets this field to blanks during command processing to protect the integrity of any password provided.

## Search Buffer

If the ADAM option (command option 2 = V) is used, the search buffer contains the field specification of the ADAM descriptor. A response 61 is returned if the field name is not an ADAM key.

## Value Buffer

If the ADAM option (command option 2 = V) is used, the value buffer contains the value of the ADAM key.

## Examples

**Example 1:**

ISN 4 in file 2 is to be deleted.

**Control Block:**

```
Command Code          E1
```

```
File Number          2 (record to be deleted is in file 2)
```

```
ISN                  4 (record with ISN 4 to be deleted)
```

```
Command Option 1     b (Return Option not to be used)
```

```
Additions 3          Password (file 2 is security protected)
```

**Example 2:**

A set of ISNs (which were previously selected with a FIND command) is to be deleted from file 1.

**Control Block:**

```
Command Code         E1
```

```
File Number          1 (record to be deleted is in file 1)
```

```
ISN                  n (ISN resulting from previous Find)
```

```
Command Option 1     b (Return Option not to be used)
```

```
Additions 3          bbbbbbbb (file is not security protected)
```

The E1 command is repeated for each ISN which resulted from the FIND command.

# 13 ET Command (End Transaction)

## Function and Use

The ET command is used for two purposes:

- End of a logical transaction (command option ‘S’ has not been specified).
- End of a subtransaction (command option ‘S’ has been specified).

### End of a Logical Transaction

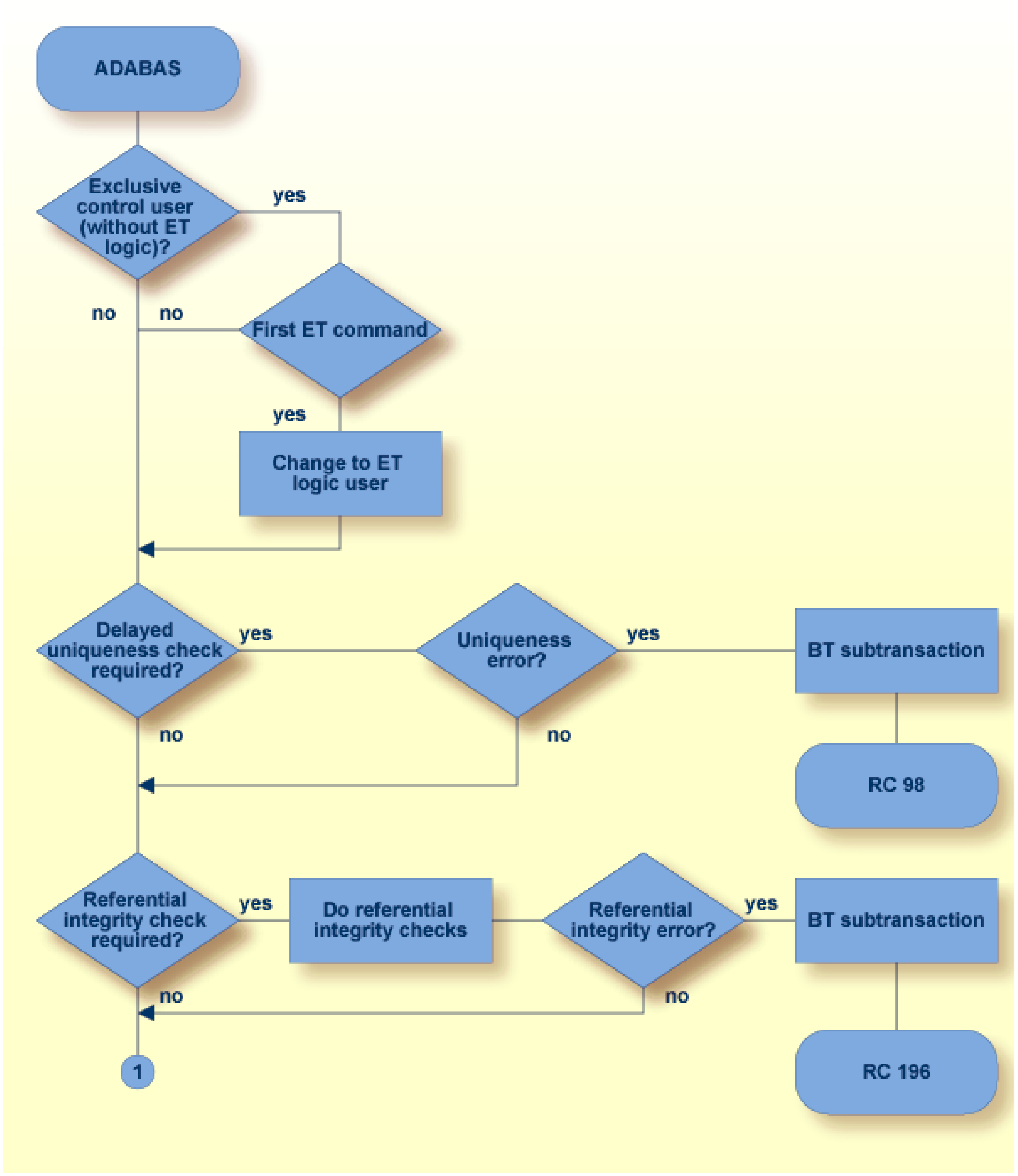
An ET command without the command option ‘S’results in:

- The writing of all current data protection information to the Adabas data protection log and Adabas work file for all update commands successfully executed during the transaction. This information may be needed by Adabas to apply all the updates which were performed during the transaction at the start of the next Adabas session. This will occur only if the current session is abnormally terminated (system failure) before these updates have been physically applied to the Associator and Data Storage;
- Downgrade of all exclusive locks of the user to shared locks, if command option ‘H’ is used. Otherwise the release of all records held by the user (without multifetch option), or the release of all records specified in the ISN buffer (with the multifetch option);
- Optionally, the storing of user data in an Adabas system file. This user data may be read subsequently with an OP or RE command and may be used for program restart;
- The returning of a unique sequence number for the transaction by Adabas. This sequence number may be used to identify the last successfully processed transaction if a restart is necessary;
- If subtransactions are activated, the end of the last subtransaction for the current transaction; this means if uniqueness and referential integrity checks have been delayed, they are performed now;
- In the case of a uniqueness error, the current subtransaction is backed out, and a response code 98 is returned. In the case of a referential integrity error, the current subtransaction is backed out, and a response code 196 is returned. If you activated subtransactions but didn’t use subtransactions, i.e. you did not use an ET command with command option ‘S’, this means that the complete transaction is rolled back;

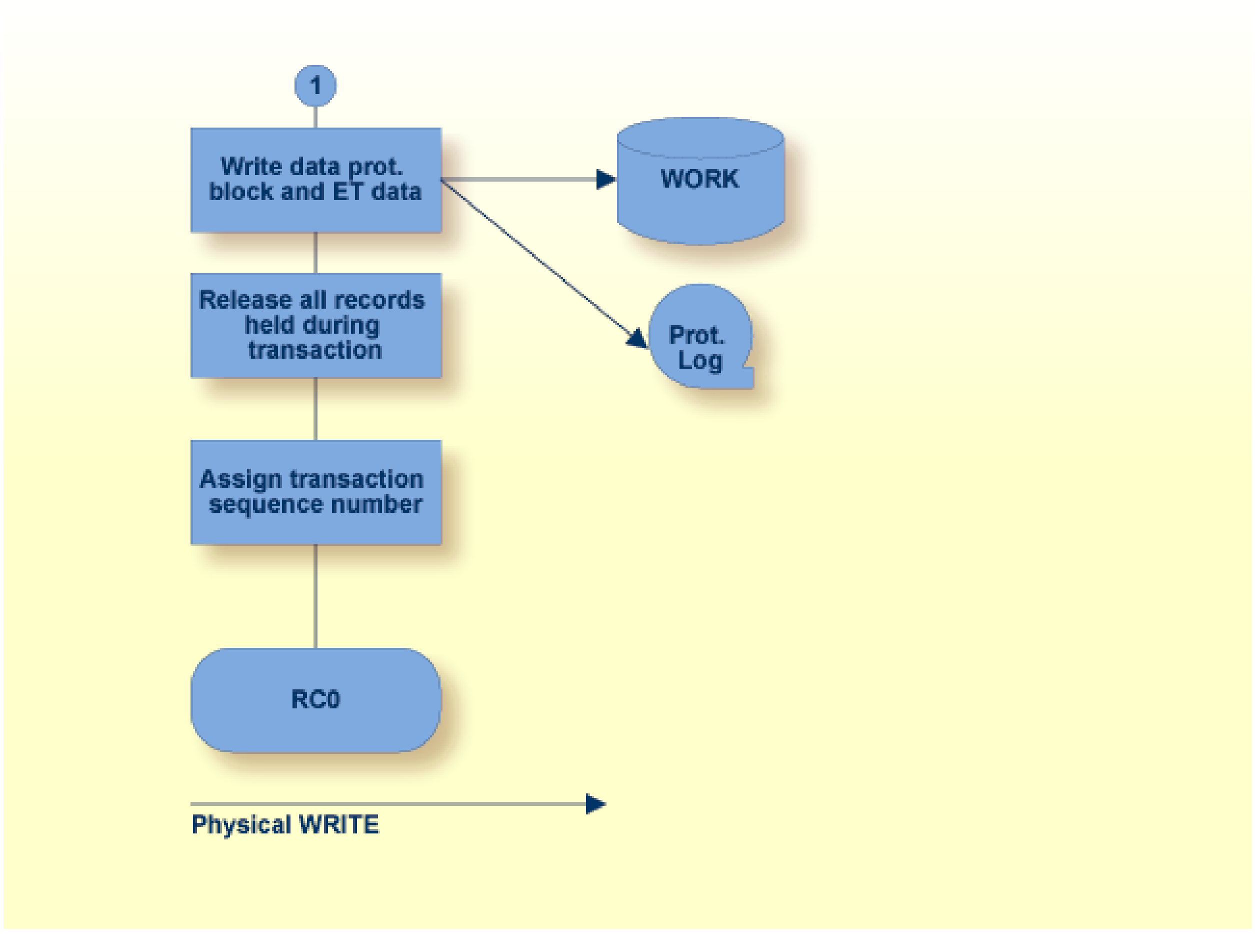
  **Note:** A referential integrity check can imply a large number of database operations, e.g. if you specified cascaded delete in the referential integrity constraint.

- If subtransactions are activated, the start of the first subtransaction for the next transaction. A new savepoint is defined with savepoint ID 0.

The successful execution of an end of a logical transaction guarantees that all of the updates performed during the transaction will be applied to the database, regardless of any subsequent user or Adabas session interruption.


ADABAS
Exclusive control user (without ET logic)?
yes
no
no
First ET command
yes
Change to ET logic user
Delayed uniqueness check required?
yes
Uniqueness error?
yes
BT subtransaction
no
no
RC 98
Referential integrity check required?
yes
Do referential integrity checks
Referential integrity error?
yes
BT subtransaction
no
no
1
RC 196

**ET Command - End of Logical Transaction, Procedure Flow**


1
Write data prot. block and ET data
WORK
Release all records held during transaction
Prot. Log
Assign transaction sequence number
RC0
Physical WRITE

**ET Command - End of Logical Transaction, Procedure Flow (continued)**

## End of a Subtransaction

An ET command with the command option 'S'results in:

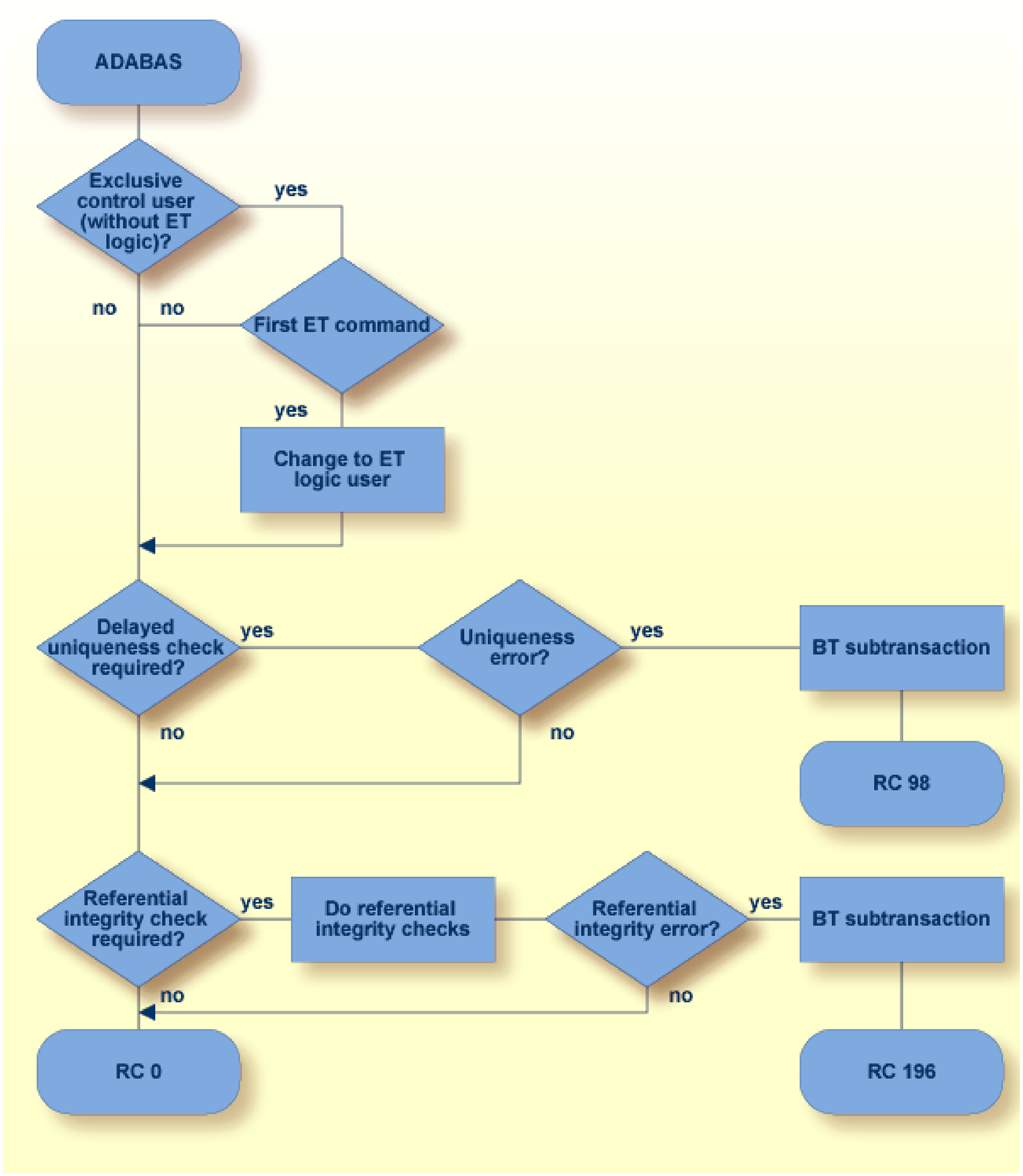
- If subtransactions are not enabled for the current Adabas user session, response code 22 is returned; the first 2 bytes of the Additions 2 field are set to 19;
- If there are uniqueness checks which have been delayed, they are performed now. In the case of a uniqueness error, the current subtransaction is backed out, and a response code 98 is returned. In the case of a referential integrity error, the current subtransaction is backed out, and a response code 196 is returned;

  **Note:** A referential integrity check can imply a large number of database operations, e.g. if you specified cascaded delete in the referential integrity constraint.

- Start of the next subtransaction. A new savepoint ID is defined; the savepoint ID is returned in the command ID field of the control block. The new savepoint ID may be the same as the previous savepoint ID, if there were no lock or update activities after the previous savepoint.

> **Note:** Usually the savepoint ID is incremented by 1. For some undocumented internal commands internal subtransactions are created. Therefore, usage of these commands can result in larger savepoint IDs.

**Note:** While the end of a logical transaction implies a commit, i.e. it is guaranteed that all of the updates performed during the transaction will be applied to the database, regardless of any subsequent user or Adabas session interruption, there is no commit at the end of a subtransaction: With a backout subtransaction you can backout all subtransactions belonging to a logical transaction until the complete transaction is committed.


ADABAS
Exclusive control user (without ET logic)?
yes
no
no
First ET command
yes
Change to ET logic user
Delayed uniqueness check required?
yes
Uniqueness error?
yes
BT subtransaction
no
no
RC 98
Referential integrity check required?
yes
Do referential integrity checks
Referential integrity error?
yes
BT subtransaction
no
no
RC 0
RC 196

**ET Command - End of Subransaction, Procedure Flow**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | -/A |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| Record Buffer Length (ACB only) | B | $ F/U |
| ISN Buffer Length (ACB only) | B | F/U (1) |
| Command Option 1 | A | F/U |
| Command Option 2 | A | F/U |
| Command Option 3 (ACBX only) | A | F/U |
| Additions 2 | A,B | -/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | $ */- |
| Record Buffer | $ F/U |
| Search Buffer | */– (2) |
| Value Buffer | */– (2) |
| ISN Buffer | F/U (2) |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |
| * | Not used but must be included in parameter list of CALL statement |

$ Only if user data to be stored

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.
(2) only if the multifetch feature is used

## Control Block

**Command Code**
ET

**Command ID**
Adabas returns in this field the sequence number for the transaction. This number is provided in binary format.

Transaction sequence numbers are assigned in ascending sequence during a given user session, starting with 1. The value 0 will be returned if the transaction has performed no updates.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**Record Buffer Length (ACB only)**
If user data is to be stored in an Adabas system file, the number of bytes of user data to be stored must be specified in this field.

Adabas will store the number of bytes specified in this field. The maximum number of bytes which may be specified is 2000 bytes.

If no user data is to be stored, this field is not used.

**ISN Buffer Length (ACB only)**
The ISN buffer length (in bytes). This buffer is only used in conjunction with the Multifetch feature. The value specified may not be smaller than 4 + (number of ISNs * 8), otherwise the Multifetch feature will be ignored.

**Command Option 1**
If this field is set to ‘S’, an end of subtransaction is performed, otherwise an end of a logical transaction is performed. If this field is blank, all records in hold for the current transaction are released. If the field is set to `M', only the ISNs specified in the ISN Buffer are released from hold. to Programming Considerations, Multifetch Feature for more detailed information.

If this field is set to ‘T’, the ET command releases all resources that are in use for the current user session: it is equivalent to (but more efficient than) a CL command followed by an OP command with the Record Buffer set to ‘.’ and command option 1 set to the same value as in the previous OP command.

**Note:** The ‘T’ option has been introduced for use after an OP without the ‘R’ option. If you perform an ET command with the ‘T’ option after an OP command with the ‘R’ option, all subsequent commands that access any file will get a response code 17. This is because using the ‘T’ option for the ET command by mistake does not allow access to other files that are not already in the file list.

**Command Option 2**
An `E' in this field indicates that user data is to be stored in an Adabas system file.

**Command Option 3 (ACBX only)**
If this field is set to ‘H’, all locked records remain in hold status, but exclusive locks are downgraded to shared locks.

**Additions 2**
For some response codes, Adabas returns detailed information in this field. See Adabas Messages and Codes for further information.

## Record Buffer

The user data to be stored in an Adabas system file is provided in this buffer.

The data will be retained until the user issues the next ET or CL command in which ET data is provided. The user data will be retained when the user session terminates only if the user issued an OP command in which a non-blank USERID was provided.

The user data can be in any format and Adabas performs no conversion on it.

## ISN Buffer

This buffer contains the ISNs and file numbers that are to be unlocked. This buffer is only used in conjunction with the Multifetch feature. If no records are to be unlocked, the first 4 bytes of this buffer must be set to zeros.

For a detailed layout description of the ISN buffer, see *Programming Considerations, Using the Multifetch Feature (Transaction Control)*.

## Examples

**Example 1: ET without user data**

**Control Block:**

```
Command Code          ET
```

```
Command Option 2      b (blank; no user data is to be stored)
```

**Example 2: ET with user data**

**Control Block:**

```
Command Code          ET
```

```
Record Buffer Length  25 (25 bytes of user data to be stored)
```

```
Command Option 2      E (user data to be stored)
```

**Buffer Areas:**

```
Record Buffer         User Data For Transaction
```

# 14 HI Command (Hold Record)

## Function and Use

The HI option is used to place record in hold status, i.e. lock the records for other users. Please refer to *Concepts and Facilities, Competitive Updating, Shared Locks and Hierarchical Locking* for further information. The command can also be used to upgrade the lock mode of a record from shared to exclusive. If the user already holds an exclusive lock for the record, the lock mode remains unchanged.

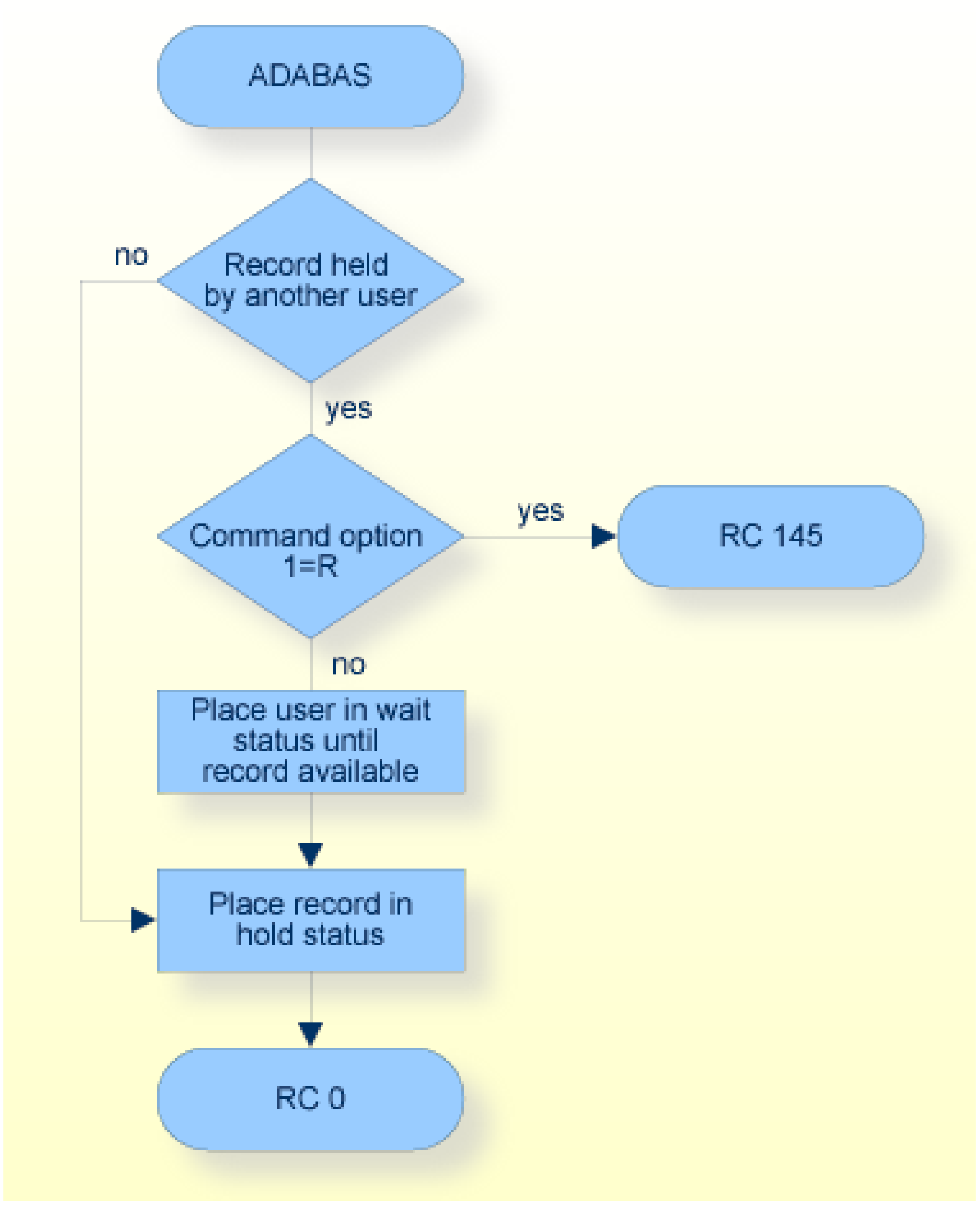
The record remains in the requested lock mode until the lock is upgraded by another command, or it is released with an ET or BT command, or it is released or downgraded with an RI command.

The user specifies the file number and ISN of the record to be held.

The HI command does not check whether the ISN to be placed in hold status exists.

Adabas will only execute this command if this is not prevented by a lock for another user. If the record is not available, the user will be placed in wait status and reactivated automatically by Adabas when the record becomes available again.

If the HI command is issued with the return option and the record to be held is currently not available, Adabas will return response code 145 instead of placing the user in wait status.


ADABAS
no
Record held by another user
yes
Command option 1=R
yes
RC 145
no
Place user in wait status until record available
Place record in hold status
RC 0

**HI Command, Procedure Flow**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN | B | F/U |
| Command Option 1 | A | F/U |
| Command Option 3 (ACBX only) | A | F/U |
| Additions 2 | A,B | -/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | –/– |
| Record Buffer | –/– |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | –/– |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
HI

**File Number**
The number of the file which contains the record to be held.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**ISN**
The ISN of the record to be placed in hold status.

**Command Option 1**
An `R' in this field indicates that the return option is to be used. If the record to be held is currently held by another user, Adabas will return response code 145 rather than placing the user in wait status until the record becomes available.

**Command Option 3 (ACBX only)**
An 'S' in this field places the record in shared hold status, if the record is not yet locked shared or exclusively. The record remains in shared hold status until the lock is upgraded to an exclusive lock, or until the lock is released again with an ET, BT or RI command.

A blank or binary zero in this field places the record in exclusive hold status.

**Additions 2**
For some response codes, Adabas returns detailed information in this field. See *Adabas Messages and Codes* for further information.

## Examples

The record identified by ISN 3 in file 2 is to be placed in exclusive hold status. Control is not to be returned until the record is available.

**Control Block:**

```
Command Code        hi
```

```
File Number           2 (record to be held is in file 2)
```

```
ISN                   3 (record with ISN 3 to be held)
```

```
Command Option 1      b (Return option not used)
```

```
Command Option 3      b (exclusive lock)
```

The record identified by ISN 4 in file 2 is to be placed in shared hold status. If the record is exclusively locked or it is waiting to be exclusively locked by another user, Adabas will return a response code 145.

**Control Block:**

```
Command Code         hi
```

```
File Number           2 (record to be held is in file 2)
```

```
ISN                   4 (record with ISN 4 to be held)
```

```
Command Option 1      R (Return option used)
```

```
Command Option 3      S
```

# 15 L1/L4 Command (Read Record)

## Function and Use

The L1/L4 command is used to read a single record from Data Storage. Using the Multifetch feature, it is also possible to read multiple records with a single L1/L4 command.

**Without the Multifetch Feature**

The user specifies the file number and ISN of the record to be read. The user indicates in the format buffer which fields are to be read. Adabas returns the requested field values in the record buffer.

The GET NEXT option provides for the reading of the records identified by the ISNs contained in an ISN list (which has been created previously by a Sx command) without the user having to provide the ISN of the record to be read with each L1/L4 call. Adabas automatically selects the ISN from the list and reads the record identified by that ISN. If the ISN no longer exists, the ISN is skipped, and the next ISN of the ISN list is read. Therefore, the number of ISNs read before you get a response code 3 (end of ISN list) may be smaller than the ISN quantity of the corresponding Sx command.

The READ ISN SEQUENCE option provides for the reading of a record identified by ISN, and if the ISN specified is not present in the file, the reading of the record with the next higher ISN which is present.

The L4 command performs the same function as the L1 command, and, in addition, places the record in shared or exclusive hold status, i.e. locks the record for other users. The L4 command should be used if the user needs to prevent other users from updating the record, e.g. because he wants to update the record. Please refer to *Concepts and Facilities, Competitive Updating, Shared Locks* for further information. If the user already holds a shared lock for the record, and requests an exclusive lock now, the lock is upgraded to the exclusive locking mode. If the user already holds an exclusive lock for the record, the locking mode remains unchanged.

If the GET NEXT option is used and the previous command for this command ID was issued with the command option ‘Q’, and the record read has not yet been released from shared hold status afterwards, this record read will be released from shared hold status again, with the following exceptions:

- If the same record has been read with the command option ‘Q’ in more than one command sequences, the record is only released if the next record has been read for all these command sequences, or an RC has been performed;
- It is not released if another command has locked the same record exclusively, or the record has been shared locked with command option ‘S’.

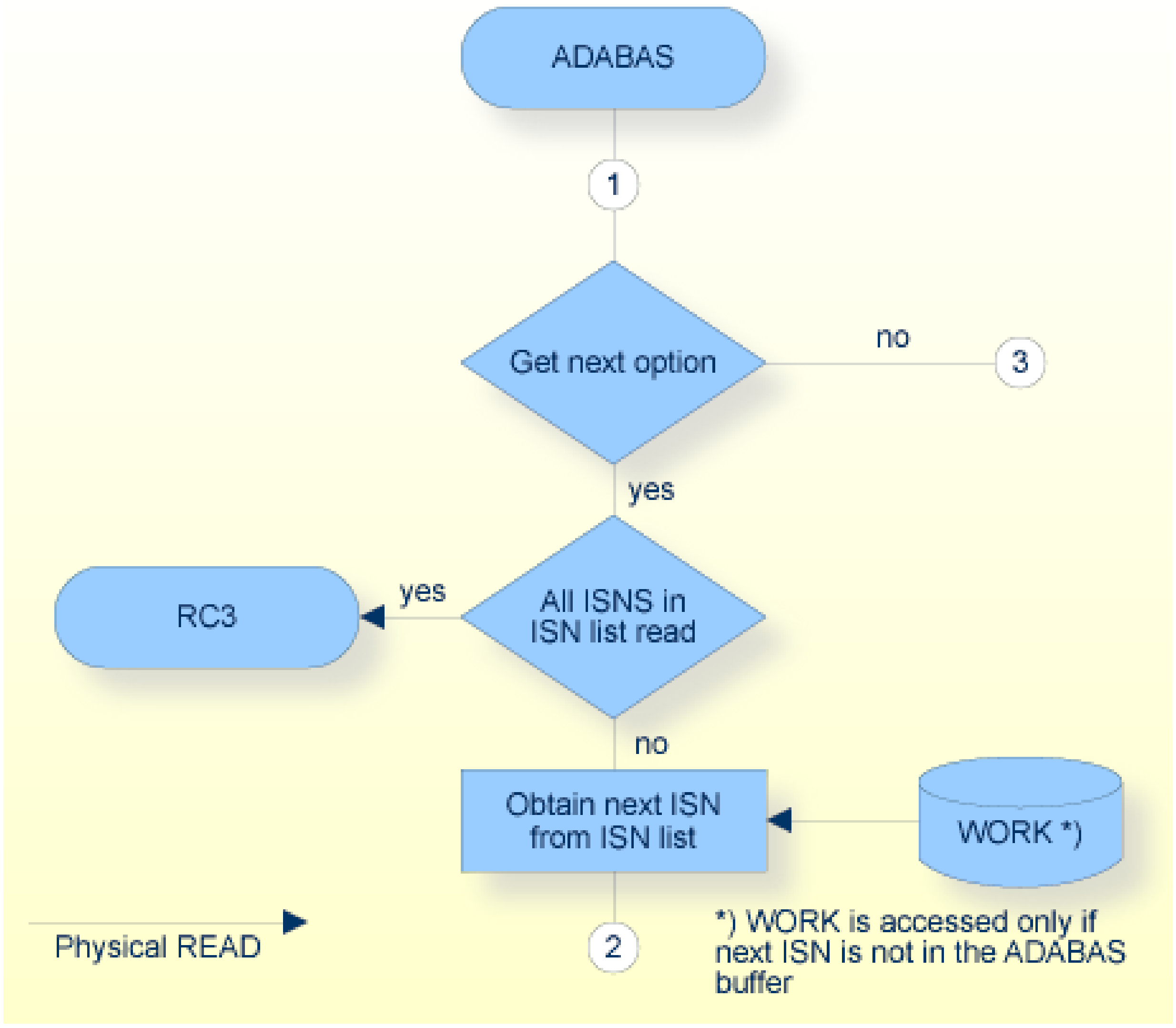
**With the Multifetch Feature**

The L1/L4 command supports the Multifetch feature. This is indicated by one of the values `M' or `O' in the Command Option 1 field (see the description of the Command option 1 field for more details). The Multifetch feature for the L1/L4 command is only available when used together with the following Command Option 2 values:

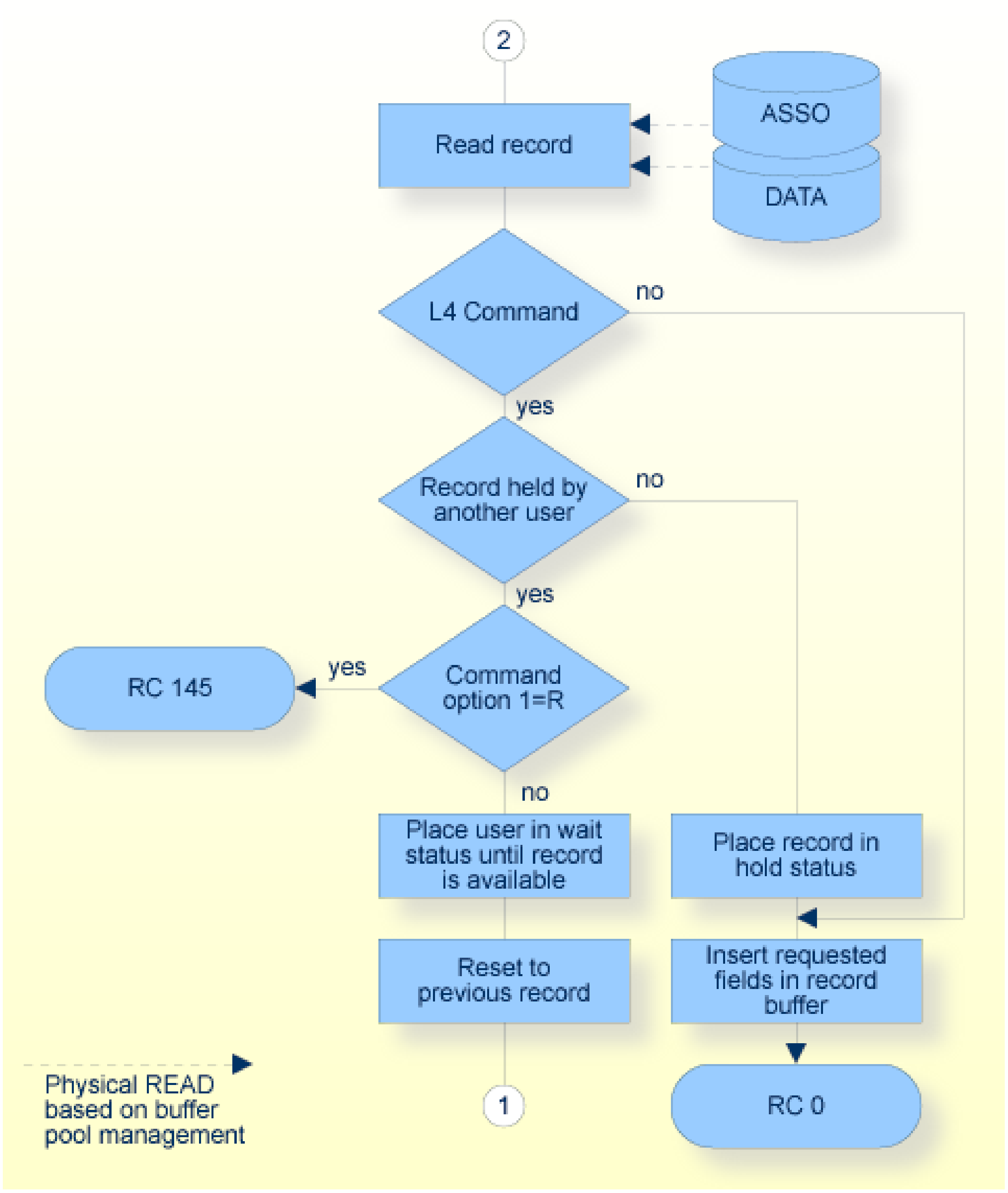
- `I' (read in ISN sequence)
- `N' (get next after FIND)

When the Multifetch feature is used, the maximum number of records which can be returned for a single L1/L4 call is limited by the following factors:

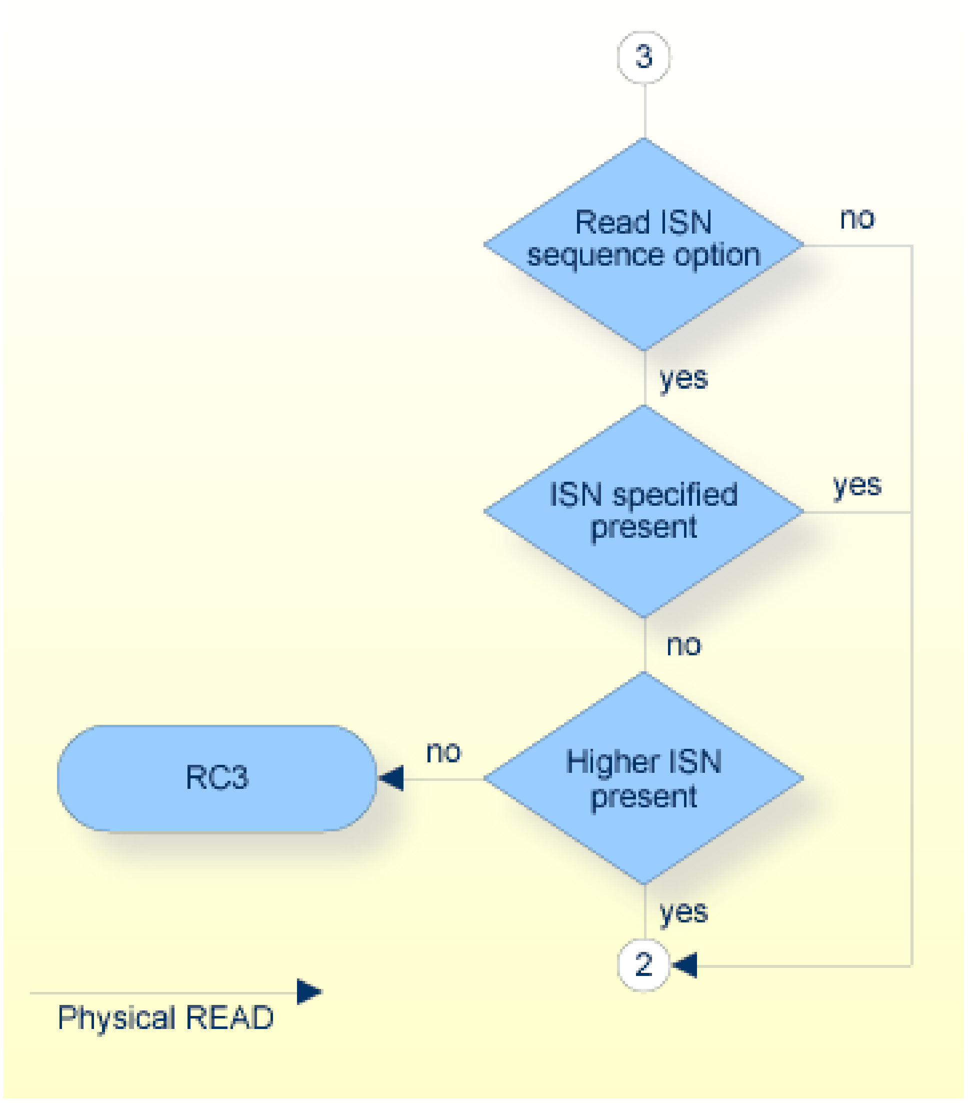
- The ISN Lower Limit field can be used to specify the maximum number of records to be returned, thus avoiding internal overheads when only a limited number of records are required.
- If the value in the ISN Lower Limit field is 0, the number of records returned is limited only by the size of the ISN/Multifetch Buffer and/or the Record Buffer.


ADABAS
1
Get next option
no
3
yes
RC3
yes
All ISNS in ISN list read
no
Obtain next ISN from ISN list
WORK *)
Physical READ
2
*) WORK is accessed only if next ISN is not in the ADABAS buffer

**L1/L4 Procedure Flow**


2
Read record
ASSO
DATA
L4 Command
no
yes
Record held by another user
no
yes
RC 145
yes
Command option 1=R
no
Place user in wait status until record is available
Place record in hold status
Reset to previous record
Insert requested fields in record buffer
Physical READ based on buffer pool management
1
RC 0

**L1/L4 Procedure Flow (continued)**



**L1/L4 Procedure Flow (continued)**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN | B | F/A |
| ISN Lower Limit | B | F/U |
| Format Buffer Length (ACB only) | B | F/U |
| Record Buffer Length (ACB only) | B | F/U |
| ISN Buffer Length (ACB only) | B | F/U |
| Command Option 1 | A | F/U |
| Command Option 2 | A | F/U |
| Command Option 3 (ACBX only) | A | F/U |
| Additions 2 | A,B | -/A |
| Additions 3 | A | F/A |
| Additions 5 | A | F/U |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | F/U |
| Record Buffer | –/A |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | –/A |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
L1/L4

**Command ID**
If a series of records is to be read with a series of L1/L4 calls and the same fields are to be specified in the format buffer for each call, this field should be set to a non-blank, non-zero value. This reduces the time required to process each L1/L4 call.

If the GET NEXT option is to be used, the Command ID of the ISN list to be used must be specified in this field. The format buffer may not be changed between successive L1/L4 calls when the GET NEXT option is used.

If only a single record is to be read, or if the format buffer is to be modified between L1/L4 calls, this field should be set to blanks or binary zeros.

The high-order byte of this field must not be set to hexadecimal `FF', except when automatic command ID generation is used (see Programming Considerations, Using Command IDs for additional information).

**File Number**
The number of the file which contains the record to be read.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

Response code 3 indicates an end-of-list condition (possible only if GET NEXT or ISN SEQUENCE option is used).

If the Multifetch feature is used and an error occurs while processing the first record, the response code is returned in this field. In this case, the contents of the ISN Buffer and Record Buffer are undefined.

If an error is detected for the second or any subsequent ISN during the processing loop of the Multifetch feature, the first non-zero response code will terminate the multifetch processing. In this case, the response code will be stored as an additional entry in the ISN Buffer itself, not in the Response Code field of the Control Block. Since there are two possible locations for a response code, an application program should check first the Response Code field in the Control Block for errors of a general nature, then in the response code field of each ISN Buffer entry individually.

**ISN**

The ISN of the record to be read. If the GET NEXT option is used, the following rules apply for this field:

- Adabas automatically selects ISNs from the ISN list in the order in which they are contained in the list, independent of the setting of this field;
- If the user wishes to position to a given ISN in the list, he has to use an S1/S4 command. See *Programming Considerations, Retrieving ISN Lists* for additional information.

If the READ ISN SEQUENCE option is used, the record identified by the ISN provided in this field will be read. If the ISN is not present in the file, the record with the next highest ISN will be read.

The GET NEXT and the READ ISN SEQUENCE options are mutually exclusive.

Adabas returns the ISN of the record which has been read in this field. This applies regardless of any options used.

If the Multifetch feature is used, the ISN returned in the control block is the ISN of the first record returned.

**ISN Lower Limit**

If the Multifetch feature is used, the maximum number of records read by a single L1/L4 command is limited to the value specified in this field. If this value is 0, the number of records is limited only by the size of the ISN Buffer and the Record Buffer.

If command option ‘L’ is used, and the Format Buffer contains a segment with *-position, the input value contains the current position of the segment. The value is increased by the length specified for the segment.

**Format Buffer Length (ACB only)**

The format buffer length (in bytes). The format buffer area defined in the user program must be as large as (or larger than) the length specified.

**Record Buffer Length (ACB only)**

The record buffer length (in bytes). The record buffer area defined in the user program must be as large as (or larger than) the length specified.

**ISN Buffer Length (ACB only)**

The ISN buffer length (in bytes). If the length of the ISN buffer is less than 20 bytes, the Multifetch feature will be ignored, and an Adabas response 53 will be returned.

This field is only used in conjunction with the Multifetch feature.

**Command Option 1**

An `R' in this field indicates that the return option is to be used. If an L4 command is issued and the record to be read and held is currently held by another user, Adabas will return response code 145 rather than placing the user in wait status until the record becomes available.

An `M' or an `O' in this field invokes the Multifetch feature. See *Programming Considerations, Using The Multifetch Feature* for additional information. The Multifetch feature can only be used if the Command Option 2 field is set to `I' or `N'.

**Command Option 2**

An `N' in this field indicates that the GET NEXT option is to be used. This option is used to read the records identified by the ISNs which are contained in an ISN list, without having to provide the ISN of the record to be read with each L1/L4 call. The ISN list to be used must be specified in the Command ID field. Response code 3 will be returned when all the ISNs in the list have been selected.

An `I' in this field indicates that the READ ISN SEQUENCE option is to be used. Adabas will read the record identified by the ISN specified in the ISN field if the ISN is present in the file. If the ISN is not present, the record with the next highest ISN will be read. If the ISN is not present and no higher ISN is present in the file, no record will be read and response code 3 will be returned.

An 'L' in this field is only allowed if the format buffer contains a segment with *-position. Then the ISN Lower Limit field is used to determine the current position of the segment with *-position. Without the command option 'L', the current position of a segment with *-position is the first (leftmost) byte of the field value. The 'L' option is not allowed together with the Multifetch feature.

**Command Option 3 (ACBX only)**

The command option 3 field is only relevant for an L4 command.

A 'C' in this field indicates that a shared lock is to be acquired for this record for only as long as the command is active. If the record was already locked before, it remains locked. Using this option avoids dirty reads: you see only the committed states of the record.

An 'S' in this field indicates that the record is to be placed in shared hold status. The lock will be released again when the current transaction is committed or backed out. If the command belongs to a subtransaction, the lock is also released when the current subtransaction is backed out. You can also release the lock with an RI command.

A 'Q' in this field indicates that the record is to be placed in shared hold status. The lock is released again at start of the next sequential read command for this read sequence, or when one of the events occurs that releases a record read with 'S' option, whichever happens first. If the same record is read by more than one command with the 'Q' option, the record is released only when, for all these command sequences, either the next record has been read, or an RC command has been issued. If the same record has also been read by another command with the ‘S’ option, or the record has been locked exclusively, the record is not released by reading the next record of the command sequence. The ‘Q’ option is not allowed in combination with Command Option 1 = ‘M’ (multifetch feature).

A blank in this field indicates that the record is locked exclusively.

**Additions 2**

If the command completes successfully, and at least one Adabas field is requested in the format buffer, Adabas returns in the first two bytes of this field the compressed record length of the data storage record that was accessed, in binary format. The last two bytes contain the length of the decompressed fields selected by the Format Buffer in binary format. If the Multifetch feature is used, this information refers to the first record read.

For some response codes, Adabas returns detailed information in this field. See *Adabas Messages And Codes* for further information.

**Additions 3**

This field is used to provide a security password.

If the file to be used is not security protected, this field should be set to blanks. If the file is security protected, the user must provide a valid password.

Adabas sets this field to blanks during command processing to protect the integrity of any password provided.

**Additions 5**

This field may be used to provide a separate format buffer ID that is used to identify the internal format buffer used for this command, or to provide a global format buffer ID.

As long as the first byte of the Additions 5 field is not alphanumeric, the value provided in the command ID field will also be used as the format buffer ID.

If the first byte is a lower case character, the bytes 5 to 8 of the Additions 5 field will be used as the separate local format buffer ID.

If the first byte is a digit or an upper case character, the Additions 5 field (8 bytes) will be used as a separate global format buffer ID, which means that the format buffer ID can be used by several users in parallel.

See *Programming Considerations, Using Command IDs* for additional information and examples.

## Format Buffer

The user specifies the fields to be read in this buffer. The format buffer definition, syntax and examples are provided in *Calling Adabas, Format and Record Buffers*.

## Record Buffer

Adabas returns the requested field values in this buffer. All values are returned according to the standard format and length of the field, unless the user has explicitly requested a different length and/or format in the format buffer.

When the Multifetch feature is used, the Record Buffer can contain data returned from multiple records. The Record Buffer consists of several entries, whereby each entry contains the requested field values from a single record. The number of entries returned, and the length of each entry is stored in a corresponding entry in the ISN Buffer.

## ISN Buffer/Multifetch Buffer

When the Multifetch feature is used with the ACB interface, the ISN Buffer contains information which describes the entries returned in the Record Buffer. If you use the Multifetch feature with the ACBX interface, multifetch buffers are used instead.

The first 4 bytes of the respective buffer specify the number of 16–byte entries which follow in the that buffer. Each 16–byte entry corresponds to an entry returned in the Record Buffer, and contains the following unsigned integer values (each 4 bytes long):

- the length of the entry in the Record Buffer
- the response code for the entry in the Record Buffer (this can be different from the value in the Response Code field in the Control Block)
  If this value is non–zero, this means that an error occurred which caused the multifetch processing to be terminated. In this case, there is no corresponding entry in the Record Buffer.
- the ISN
- an unused field

## Additional Considerations

The Command ID used is internally saved and used by Adabas. It will be released by Adabas when an end–of–file condition is detected, when an RC or CL command is issued, or when the Adabas session is terminated. The same command ID must not be used by the user for another read sequential command until it has been released.

Command IDs are not released at the end of a global transaction.

## Examples

**Note:** The Adabas file definitions in *Appendix A* are used in all examples in this section.

### Example 1: Reading a single Record

ISN 4 in file 1 is to be read. The values for fields AA and AB are to be returned.

**Control Block:**

```
Command Code          L1

Command ID            bbbb (only 1 record is to be read)

File Number           1

ISN                   4

Format Buffer Length  6 (or larger)

Record Buffer Length  10 (or larger)
```

```
Command Option 2      b (Get Next or Read ISN Seq.  options  not
                         used)
```

```
Additions 3           bbbbbbbb (file is not security protected)
```

**Buffer Areas:**

```
Format Buffer         AA,AB.
```

### Example 2: Reading a Record with ISN from the ISN List returned by a Find Command in the ISN Buffer

A set of records for which the ISNs have been previously obtained by a FIND command are to be read from file 2. The values for fields RA and XB are to be returned with the value for field XB to be returned with length 3 and format U.

**Control Block:**

```
Command Code          L1
```

```
Command ID            ABCD (a non-blank CID  is recommended for
                            a series  of  reads  in  which  the  same
                            fields are being read  in  each  record).
```

```
File Number           2
```

```
ISN                   n (ISNs  are  taken from  the  ISN  list
                          created by the Find command).
```

```
Format Buffer Length  10 (or larger)
```

```
Record Buffer Length  11 (or larger)
```

```
Command Option 2      b  (Get Next or Read ISN Seq. options not
                          used)
```

```
Additions 3           Password (file is security protected)
```

**Buffer Areas:**

```
Format Buffer         RA,XB,3,U.
```

**Note:** n indicates an ISN from the ISN list which resulted from the FIND command. The L1 call is repeated for each ISN in the ISN list.

### Example 2a: Reading a Set of Records using the Get Next Option

The requirement as stated for example 2 may also be resolved by using the GET NEXT option.

**Control Block:**

```
Command Code          L1
```

```
Command ID            ABCD (CID of ISN list to be used)
```

```
File Number           2
```

```
ISN                   0  (the entire ISN list is to be selected
                          starting with the first ISN in the list)
```

```
Format Buffer Length  10 (or larger)
```

```
Record Buffer Length  11 (or larger)
```

```
Command Option 2      N (Get Next option to be used)
```

```
Additions 3           Password (file 2 is security protected)
```

```
Additions 5           APL3FB01 (a global format buffer ID is
                                recommended for series of reads in
                                which the same fields are read in each
                                record for several users using the same
                                application)
```

**Buffer Areas:**

```
Format Buffer         RA,XB,3,U.
```

The L1 call is repeated for each ISN in the ISN list. No changes to the control block are required between L1 calls. Response code 3 will be returned when all the ISNs in the list have been selected.

### Example 3: Read with Hold

ISN 5 in file 2 is to be read and held for updating. The values for fields XC and XD are to be returned.

**Control Block:**

```
Command Code          L4 (Read With Hold)
```

```
Command ID            bbbb (only 1 record to be read)
```

```
File Number           2
```

```
ISN                   5
```

```
Format Buffer Length  6 (or larger)
```

```
Record Buffer Length  14 (or larger)
```

```
Command Option 1      b (Return option not used)
```

```
Command Option 2      b (Get Next or Read ISN Seq. options not
                         used)
```

```
Additions 3           Password (file is security protected)
```

**Buffer Areas:**

```
Format Buffer         XC,XD.
```

### Example 4: Read using the Read ISN Sequence Option

File 1 is to be read using the READ ISN SEQUENCE option. The values for fields AA, AB and AC are to be returned.

**Control Block:**

```
Command Code          L1
```

```
Command ID            BCDE (non-blank CID is recommended when a
                            series  of  records  for  which  the same
                            fields are to be returned is to be read)
```

```
File Number           1
```

```
ISN                   1  (If  ISN 1  is not present, the record
                          with the next higher ISN will be read)
```

```
Format Buffer Length  6 (or larger)
```

```
Record Buffer Length  30 (or larger)
```

```
Command Option 2      I (Read ISN Sequence option is invoked)
```

```
Additions 3           bbbbbbbb (file is not security protected)
```

**Buffer Areas:**

```
Format Buffer         GA,AC.
```

Adabas returns the ISN of the record which has been read in the ISN field of the control block. The record with the next highest ISN may be read by adding one to the ISN field and repeating the L1 command.

### Example 5: Reading multiple-value Fields and periodic Groups

The record identified by ISN 2 in file 1 is to be read. The value for field AA, all values for the multiple value field MF and all occurrences of the periodic group GB are to be returned.

1. Issue an L1 call to obtain the count of the number of values which exist for MF and the highest occurrence count for GB.

   **Control Block:**

   ```
   Command Code          L1
   ```

   ```
   Command ID            bbbb (only 1 record is to be read)
   ```

   ```
   File Number           1
   ```

```
ISN                 2
```

```
Format Buffer Length  8 (or larger)
```

```
Record Buffer Length  2 (or larger)
```

```
Command Option 2      b (Get Next or Read ISN Seq. option not
                         used)
```

**Buffer Areas:**

```
Format Buffer         MFC,GBC.
```

2. Assuming that the result of the above L1 call was that the record contains 4 values for MF and 6 occurrences for GB, repeat the L1 call using the following format buffer:

   ```
   AA,MF1-4,GB1-6
   ```

   For each call, the length of the format and record buffers must be sufficiently large to accommodate all entries and values.

When using this procedure to read a series of records, a non-blank CID should be used in step 1 and a blank CID should be used for step 2, since the content of the format buffer may vary with each step 2 call.

An alternative solution for example 5 usually provides better performance if the number of values/occurrences is small (less than 6) in a large percentage of the records.

1. Issue an L1 call in which the counts for MF and GB are requested, plus the expected number of values and occurrences of MF and GB, plus field AA.

   Assuming that the expected number of values for MF is 2 and the expected number of occurrences of GB is 3, the format buffer for step 1 would be:

   ```
   AA,MFC,GBC,MF1-2,GB1-3
   ```

   Maximum performance is normally achieved if a number which will retrieve all of the values/occurrences in 90 per cent of the records is specified.

2. If the count received for MF exceeds 2 or the count received for GB exceeds 3, then a format buffer similar to that in step 2 above would be used to obtain the additional values and/or occurrences. Otherwise, no additional call is required.

### Example 6: Read a LOB Value

The LOB field L1 of the record with ISN 6 is to be read. Assume that you have defined an area of 1,000,000 bytes for the LOB value. The LOB value can then be read with the following command:

**Control Block (ACBX):**

```
Command Code          L1
```

```
Command ID            EFGH
```

```
File Number           2
```

```
ISN                   6
```

```
Command Option 1      b (Return option not used)
```

```
Command Option 2      b (Get Next or Read ISN Seq. options not used)
```

```
Additions 3           Password (file is security protected)
```

**Buffer Areas:**

```
Format Buffer 1         L1L.
```

```
Record Buffer 1 Length  4
```

```
Format Buffer 2         L1,*.
```

```
Record Buffer 2 Length  1000000
```

If the LOB value is larger than 1,000,000 bytes, you can allocate a new memory area that is large enough for the LOB value, set the record buffer 2 length to the new value, and repeat the call.

### Example 7: Read a LOB Value piecemeal

A set of records has previously been found by an S1 command with command ID “ABCD” without format and ISN buffer.

The LOB field L1 is read piecemeal in segments of 1,000,000 bytes. The first call for a record uses the Get Next option to fetch the next record of the ISN list, and reads the first segment of the LOB value:

**Control Block (ACBX):**

```
Command Code          L4 (read with hold)
```

```
Command ID            ABCD
```

```
File Number           2
```

```
ISN                   unchanged from S1 command
```

```
ISN Lower Limit       0
```

```
Command Option 1      b (Return option not used)
```

```
Command Option 2      N (Get Next option)
```

```
Command Option 3      Q (Shared hold until next record read)
```

```
Additions 3           Password (file is security protected)
```

**Buffer Areas:**

```
Format Buffer 1         L1L.
```

```
Record Buffer 1 Length  4
```

```
Format Buffer 2         L1(*,1000000).
```

```
Record Buffer 2 Length  1000000
```

If the length of the LOB field is larger than 1.000.000, you can read the next segment of the LOB value with the following command:

**Control Block (ACBX):**

```
Command Code          L1
```

```
Command ID            ABCD
```

```
File Number           2
```

```
ISN                   unchanged from L4 command
```

```
ISN Lower Limit       unchanged from previous call
```

```
Command Option 1      b (Return option not used)
```

```
Command Option 2      L (*-position determined by ISN Lower Limit)
```

```
Additions 3          Password (file is security protected)
```

**Buffer Areas:**

```
Format Buffer 1        L1L.
```

```
Record Buffer 1 Length  4
```

```
Format Buffer 2        L1(*,1000000).
```

```
Record Buffer 2 Length  1000000
```

In order to read the complete LOB value, you must repeat this L1 call without changing the ISN Lower Limit between the calls, either as often as indicated by the LOB length or until you get a response code 3.

### Example 7a: Read a LOB Value piecemeal plus other Fields

If additional fields are to be read in the previous example, you would read these other fields several times, which is superfluous. Therefore, you can use two different format buffers for the L4 command for the non-LOB fields and the L1 command for the LOB field segment:

**Control Block (ACBX):**

```
Command Code         L4 (Read with Hold)
```

```
Command ID           ABCD
```

```
File Number          2
```

```
ISN                  unchanged from S1 command
```

```
Command Option 1      b (Return option not used)
```

```
Command Option 2      N (Get Next option)
```

```
Command Option 3      Q (Shared hold until next record read)
```

```
Additions 3           Password (file is security protected)
```

**Buffer Areas:**

```
Format Buffer 1         L1L,RG.
```

```
Record Buffer 1 Length  53 (or larger)
```

The LOB field is read piecemeal in segments of 1.000.000 bytes with the following command:

**Control Block (ACBX):**

```
Command Code          L1
```

```
Command ID            EFGH
```

```
File Number           2
```

```
ISN                   unchanged from L4 command
```

```
ISN Lower Limit       0
```

```
Command Option 1      b (Return option not used)
Command Option 2      L (*-position determined by ISN Lower Limit)
```

```
Additions 3           Password (file is security protected)
```

**Buffer Areas:**

```
Format Buffer 1         L1L,(*,1000000).
```

```
Record Buffer 1 Length  1000000 (or larger)
```

In order to read the complete LOB value, you must repeat this L1 call without changing the ISN Lower Limit between the calls, either as often as indicated by the LOB length or until you get response code 3.

# 16 L2/L5 Command (Read Physical Sequence)

## Function and Use

The L2 command is used to read a record from a set of records which are stored in physical sequence in Data Storage. Using the Multifetch feature, it is also possible to read multiple records with a single L2/L5 command.

**Without the Multifetch Feature**

The L2 command does not result in the reading of records in any particular logical order (unless the records were initially loaded in a particular logical sequence and no updating of the file which resulted in a change to this order has been performed).

Using the L2 command repeatedly, an entire file can be read at optimum speed since no access is required to the Associator (as with the L3 command), and all physical blocks are read in consecutive sequence.

The user specifies the file to be read and the fields within each record for which values are to be returned. The fields are specified in the format buffer. Adabas returns the requested field values in the record buffer.

The L5 command performs the same function as the L2 command, and, in addition, places the record in shared or exclusive hold status, i.e. locks the record for other users. The L5 command should be used if the user needs to prevent other users from updating the record, e.g. because he wants to update the record. Please refer to *Concepts and Facilities, Competitive Updating, Shared Locks and Hierarchical Locking* for further information. If the user already holds a shared lock for the record, and requests an exclusive lock now, the lock is upgraded to the exclusive locking mode. If the user already holds an exclusive lock for the record, the locking mode remains unchanged.

If the previous command for this command ID was issued with the ‘Q’ option and the record read has not yet been released from shared hold status afterwards, this record read will be released from shared hold status again with the following exceptions:

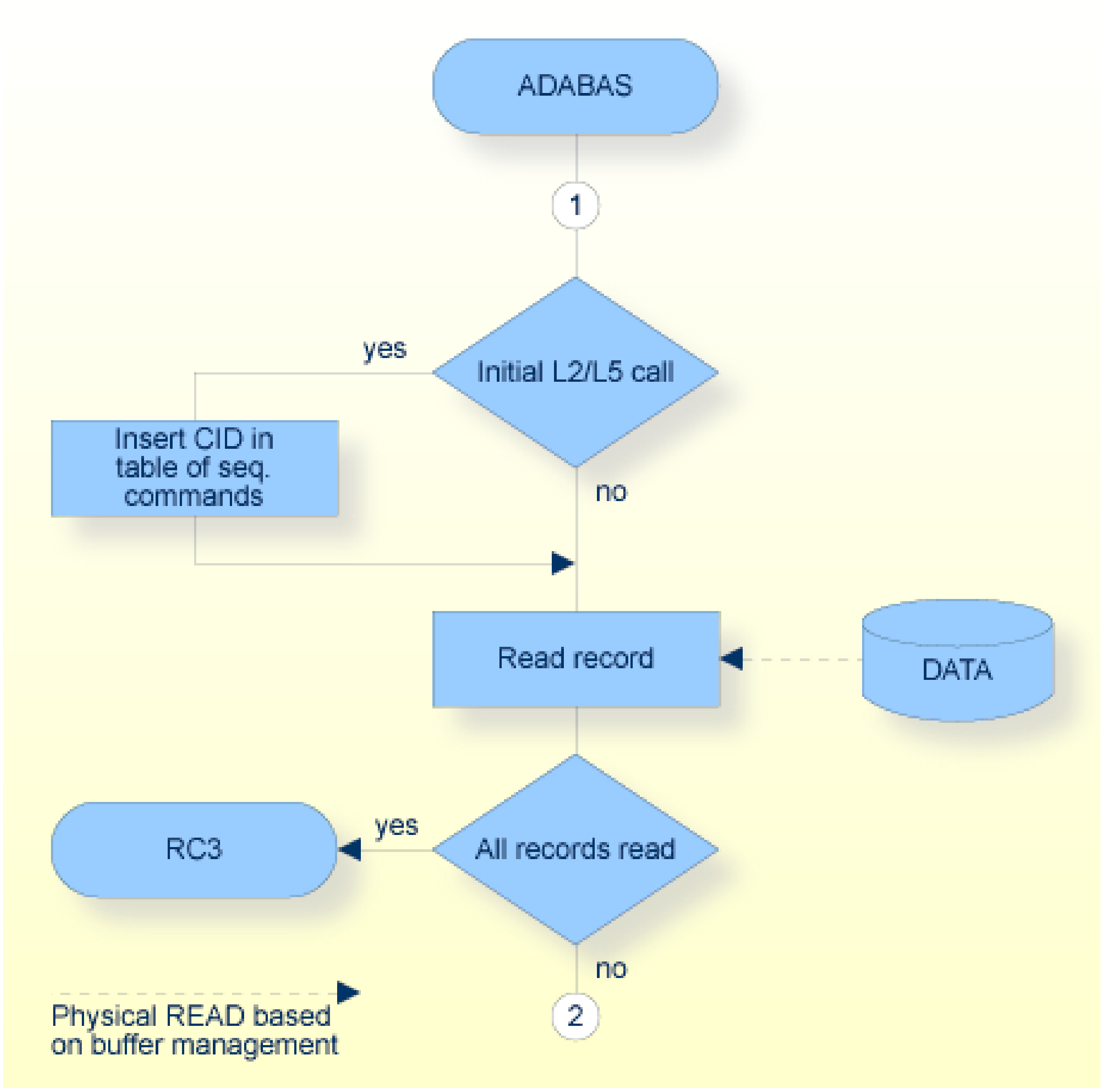
- If the same record has been read with the command option ‘Q’ in more than one command sequence, the record is only released if the next record has been read for all these command sequences, or an RC has been performed.
- It is not released if another command has locked the same record exclusively, or the record has been shared locked with the command option ‘S’.

**With the Multifetch Feature**

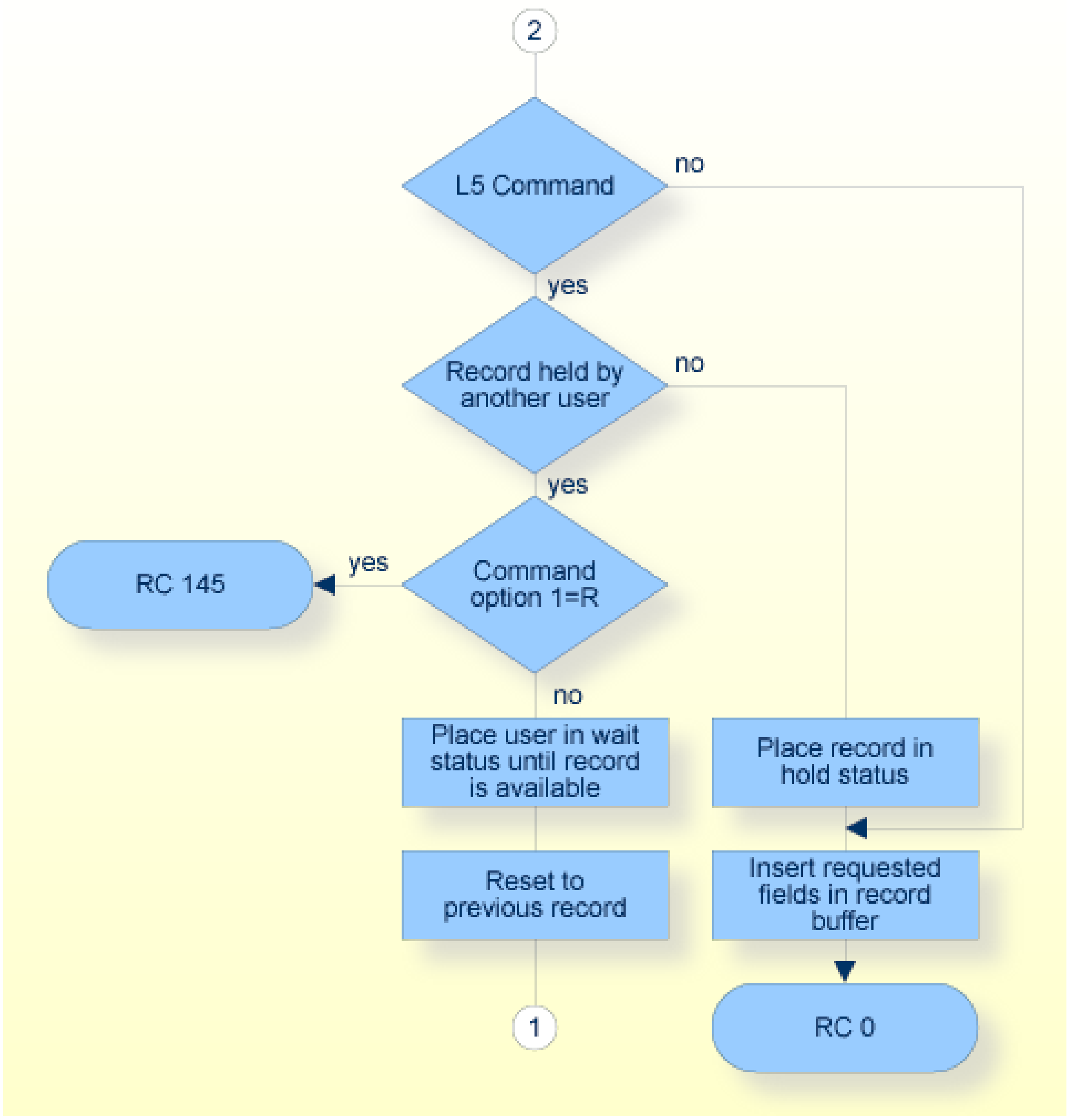
The L2/L5 command supports the Multifetch feature. This is indicated by one of the values `M' or `O' in the Command Option 1 field (see the description of the Command option 1 field for more details).

When the Multifetch feature is used, the maximum number of records which can be returned for a single L2/L5 call is limited by the following factors:

- The ISN Lower Limit field can be used to specify the maximum number of records to be returned, thus avoiding internal overheads when only a limited number of records are required.
- If the value in the ISN Lower Limit field is 0, the number of records returned is limited only by the size of the ISN/Multifetch Buffer and the Record Buffer.


ADABAS
1
yes
Initial L2/L5 call
Insert CID in table of seq. commands
no
Read record
DATA
yes
RC3
All records read
no
2
Physical READ based on buffer management

**L2/L5 Command, Procedure Flow**


2
L5 Command
no
yes
Record held by another user
no
yes
RC 145
yes
Command option 1=R
no
Place user in wait status until record is available
Place record in hold status
Reset to previous record
Insert requested fields in record buffer
1
RC 0

**L2/L5 Procedure Flow (continued)**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN | B | F/A |
| ISN Lower Limit | B | F/U |
| Format Buffer Length (ACB only) | B | F/U |
| Record Buffer Length (ACB only) | B | F/U |
| ISN Buffer Length (ACB only) | B | F/U |
| Command Option 1 | A | F/U |
| Command Option 3 (ACBX only) | A | F/U |
| Additions 2 | A,B | -/A |
| Additions 3 | A | F/A |
| Additions 5 | A | F/U |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | F/U |
| Record Buffer | –/A |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | –/A |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |

| | |
|---|---|
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**

L2/L5

**Command ID**

This field must be set to a non-blank, non-zero value. It is used by Adabas to provide the records in the correct physical order and to avoid the repetitive interpretation of the format buffer. The value provided must not be modified during any given sequential pass of a file.

The high-order byte of this field must not be set to hexadecimal `FF', except when automatic command ID generation is used (see *Programming Considerations, Using Command IDs* for additional information).

**File Number**

The number of the file to be read.

**Response Code**

Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

Response code 3 indicates an end-of-file condition has been detected. Response codes 17, 18, 21, 23, 145 are possible as well.

If the Multifetch feature is used and an error occurs before or while processing the first record, the response code is returned in this field. In this case, the contents of the ISN Buffer and Record Buffer are undefined.

If an error is detected for the second or any subsequent ISN during the processing loop of the Multifetch feature, the first non-zero response code will terminate the multifetch processing. In this case, the response code will be stored as an additional entry in the ISN Buffer itself, not in the Response Code field of the Control Block. Since there are two possible locations for a response code, an application program should check first the Response Code field in the Control Block for errors of a general nature, then in the response code field of each ISN Buffer entry individually.

**ISN**

If this field is set to zero before the initial L2/L5 call, the sequential pass will begin with the first record contained in the first physical block of the file.

If this field is set to an ISN value before the initial L2/L5 call, the sequential pass will begin at the first record physically located after the record identified by the ISN specified. The ISN specified must be present in the file, otherwise response code 23 will be returned.

This field need not be modified by the user after the initial L2/L5 call.

Adabas returns the ISN of the record which has been read in this field.

If the Multifetch feature is used, the ISN returned in the control block is the ISN of the first record read.

**ISN Lower Limit**

If the Multifetch feature is used, this field is used to limit the number of records to be returned. If this field is set to zero, the maximum number of records returned depends on the size of the record buffer and/or the size of the ISN buffer.

**Format Buffer Length (ACB only)**

The format buffer length (in bytes). The format buffer area defined in the user program must be as large as (or larger than) the length specified.

**Record Buffer Length (ACB only)**

The record buffer length (in bytes). The record buffer area defined in the user program must be as large as (or larger than) the length specified.

**ISN Buffer Length (ACB only)**

The ISN buffer length (in bytes). If the length of the ISN buffer is less than 20 bytes, the Multi-fetch option will be ignored.

This field is only used in conjunction with the Multifetch feature.

**Command Option 1**

An `R' in this field indicates that the return option is to be used. If an L5 command is issued, and the record to be read and held is currently held by another user, Adabas will return response code 145 rather than placing the user in wait status until the record becomes available.

An `M' in this field invokes the Multifetch feature. See *Programming Considerations, Using The Multifetch Feature* for additional information.

If the Multifetch feature is required with the Return option, the value `O' should be used instead of `M'.

**Command Option 3 (ACBX only)**

The command option 3 is relevant only for an L5 command.

A ‘C’ in this field indicates that a shared lock is to be acquired for this record only as long as the command is active. If the record was already locked before, the record remains locked. Using this option avoids dirty reads: you see only the committed states of the record.

An ‘S’ in this field indicates that the record is to be placed in shared hold status. The lock will be released again when the current transaction is committed or backed out. If the command belongs to a subtransaction, the lock is also released when the current subtransaction is backed out. You can also release the lock with an RI command.

A ‘Q’ in this field indicates that the record is to be placed in shared hold status. The lock is released again at start of next sequential read command for this read sequence, or when one of the events happens that releases a record read with ‘S’ option, whichever happens first. If the same record is read by more than one command with ‘Q’ option, the record is released only when, for all these command sequences, either the next record has been read, or an RC command has been issued. If the same record has also been read by another command with ‘S’ option, or the record has been locked exclusively, the record is not released by reading the next record of the command sequence. The ‘Q’ option is not allowed in combination with the command option 1 = ‘M’ (multifetch feature).

A blank in this field indicates that the record is locked exclusively.

**Additions 2**

If the command completes successfully,and at least one Adabas field is requested in the format buffer, Adabas returns in the first two bytes in this field the compressed record length of the data storage record that was accessed, in binary format. The last two bytes contain the length of the decompressed fields selected by the Format Buffer in binary format. If the Multifetch feature is used, this information refers to the first record read.

For some response codes, Adabas returns detailed information in this field. See *Adabas Messages And Codes* for further information.

**Additions 3**

This field is used to provide a security password.

If the file to be used is not security protected, this field should be set to blanks. If the file is security protected, the user must provide a valid password.

Adabas sets this field to blanks during command processing to protect the integrity of any password provided.

**Additions 5**

This field may be used to provide a separate format buffer ID that is used to identify the internal format buffer used for this command, or to provide a global format buffer ID.

As long as the first byte of the Additions 5 field is not alphanumeric, the value provided in the command ID field will also be used as the format buffer ID.

If the first byte is a lower case character, the bytes 5 to 8 of the Additions 5 field will be used as the separate local format buffer ID.

If the first byte is a digit or an upper case character, the Additions 5 field (8 bytes) will be used as a separate global format buffer ID, which means that the format buffer ID can be used by several users in parallel.

See *Programming Considerations, Using Command IDs* for additional information and examples.

## Format Buffer

The fields for which values are to be returned are specified in this buffer. Format buffer definition, syntax and examples are provided in *Calling Adabas, Format and Record Buffers*.

The format buffer may not be modified after the initial L2/L5 call has been issued.

## Record Buffer

Adabas returns the requested field values in this buffer. All field values are returned according to the standard length and format of each field, unless the user has explicitly requested a different length and/or format in the format buffer.

When the Multifetch feature is used, the Record Buffer can contain data returned from multiple records. The Record Buffer consists of several entries, whereby each entry contains the requested field values from a single record. The number of entries returned, and the length of each entry is stored in a corresponding entry in the ISN Buffer.

## ISN Buffer/Multifetch Buffer

When the Multifetch feature is used with the ACB interface, the ISN Buffer contains information which describes the entries returned in the Record Buffer. If you use the Multifetch feature with the ACBX interface, multifetch buffers are used instead.

The first 4 bytes of the respective buffer specify the number of 16-byte entries which follow in the that buffer. Each 16-byte entry corresponds to an entry returned in the Record Buffer, and contains the following unsigned integer values (each 4 bytes long):

- the length of the entry in the Record Buffer
- the response code for the entry in the Record Buffer (this can be different from the value in the Response Code field in the Control Block)
  If this value is non-zero, this means that an error occurred which caused the multifetch processing to be terminated. In this case, there is no corresponding entry in the Record Buffer.
- the ISN

- an unused field

## Additional Considerations

The following additional considerations are applicable for the L2/L5 command:

1. The Command ID used with the L2/L5 command is internally saved and used by Adabas. It will be released by Adabas when an end-of-file condition is detected, when an RC or CL command is issued, or when the Adabas session is terminated. The same command ID must not be used by the user for another read sequential command until it has been released.

   Command IDs are not released at the end of a global transaction (see the Administration Manual for additional information).
2. The user is permitted to update and/or delete records which he has read with an L2/L5 command. Adabas maintains information as to the next record to be provided to the user and is able to provide the correct next record despite any interim record update or or delete performed by this user. But it may be that he receives the same record more than once during a given sequential pass of the file.
3. If another user is updating the file being read with an L2/L5 command, or if the user performing the L2/L5 commands is updating or deleting a record in the file that is not the record currently read with the L2/L5 command, it is possible that the user reading with the L2/L5 command will not receive one or more records in the file, or that he receives one record more than once.

## Examples

**Example 1:**

File 2 is to be read in physical sequential order. All the values for all the fields within each record are to be returned.

**Control Block:**

```
Command Code          L2
```

| | |
|---|---|
| Command ID | EXL2 (non blank CID required) |
| File Number | 2 |
| ISN | 0 (all records are to be read) |
| Format Buffer Length | 3 (or larger) |
| Record Buffer Length | 49 (or larger) |
| Command Option 1 | b (Return option not used) |
| Additions 3 | Password (file 2 is security protected) |

**Buffer Areas:**

| | |
|---|---|
| Format Buffer | RG. |

The L2 call is repeated to obtain each successive record. The ISN field need not be modified between calls.

**Example 2:**

File 2 is to be read in physical sequential order. The values for fields RA, XA, and XB (3 bytes unpacked) are to be returned. Each record read is to be placed in hold status for updating purposes.

**Control Block:**

| | |
|---|---|
| Command Code | L5 |
| Command ID | EXL5 (non blank CID is required) |

```
File Number          2
```

```
ISN                  0 (all records are to be read)
```

```
Format Buffer Length  13 (or larger)
```

```
Record Buffer Length  21 (or larger)
```

```
Command Option 1     b (Return option not used)
```

```
Additions 3          Password (file 2 is security protected)
```

**Buffer Areas:**

```
Format Buffer        RA,XA,XB,3,U.
```

The L5 call is repeated to obtain each successive record. The ISN field need not be modified between L5 calls.

Each record held should be released by an ET command (ET logic users) or RI command (non-ET logic users).

# 17 L3/L6 Command (Read Logical Sequence)

## Function and Use

The L3/L6 command is used to read a record from a set of records which are stored in logical sequential order based on the ascending or descending sequence of the values for a given descriptor. Using the Multifetch feature, it is also possible to read multiple records with a single L3/L6 command.

**Without the Multifetch Feature**

The user specifies the file to be read, the descriptor to be used to control the read sequence (phonetic descriptors may not be used), and the fields within each record for which values are to be returned. Adabas returns the requested field values in the record buffer.

Optionally, a starting value, ending value or a range of values may be specified. In addition, positioning to a given value during the sequential read allows the user to `browse' through a file without having to read each record.

The L6 command performs the same function as the L3 command, and, in addition, places the record in shared or exclusive hold status, i.e. locks the record for other users. The L6 command should be used if the user needs to prevent other users from updating the record, e.g. because he wants to update the record. Please refer to *Concepts and Facilities, Competitive Updating, Shared Locks and Hierarchical Locking* for further information. If the user already holds a shared lock for the record and requests an exclusive lock now, the lock is upgraded to the exclusive locking mode. If the user already holds an exclusive lock for the record, the locking mode remains unchanged.

If the previous command for this command ID was issued with the ‘Q’ option, and the record read has not yet been released from shared hold status afterwards, this record read will be released from shared hold status again with the following exceptions:

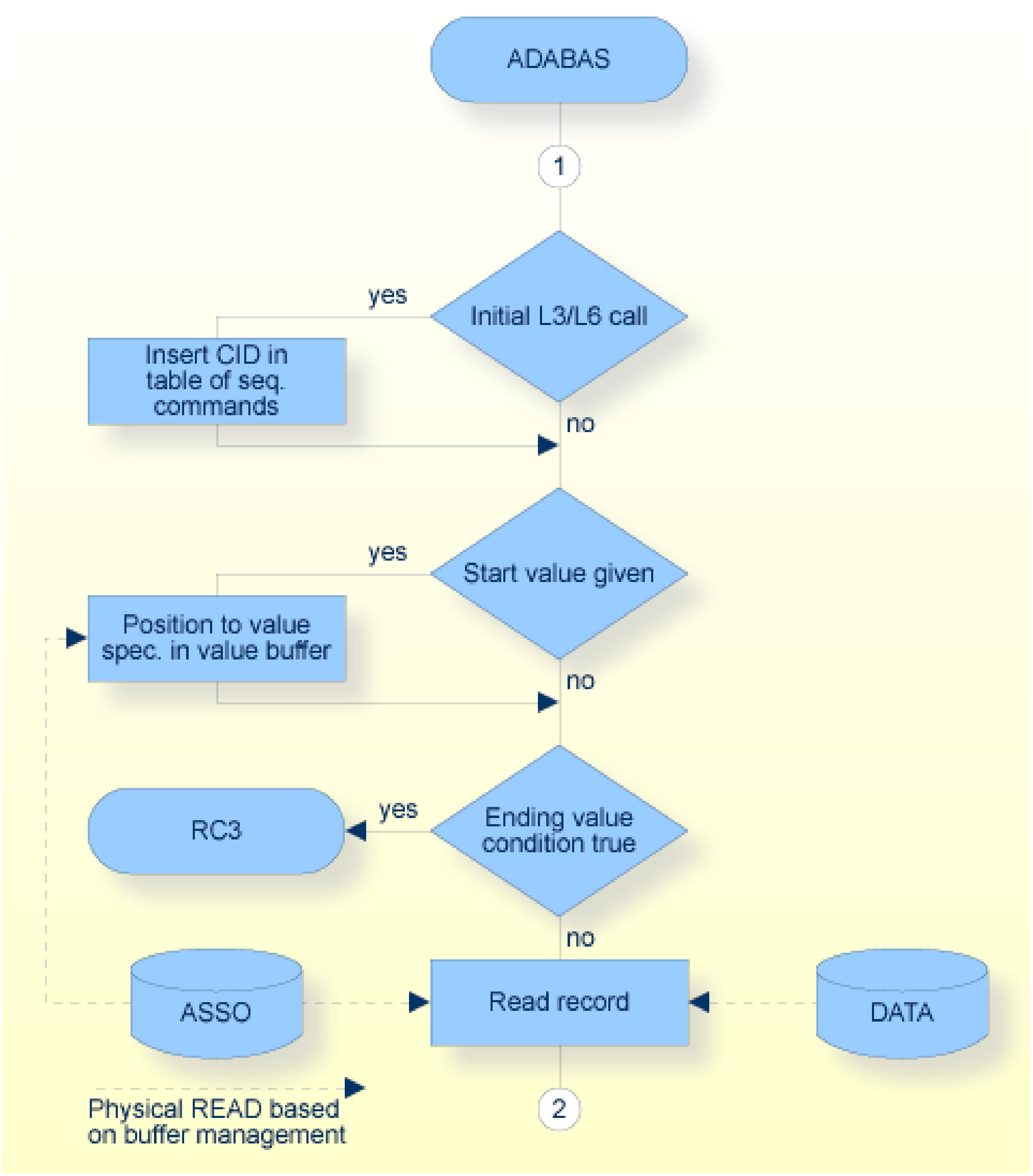
- If the same record has been read with the command option ‘Q’ in more than one command sequence, the record is only released if the next record has been read for all these command sequences, or an RC has been performed.
- It is not released if another command has locked the same record exclusively, or the record has been shared locked with the command option ‘S’.

**With the Multifetch Feature**

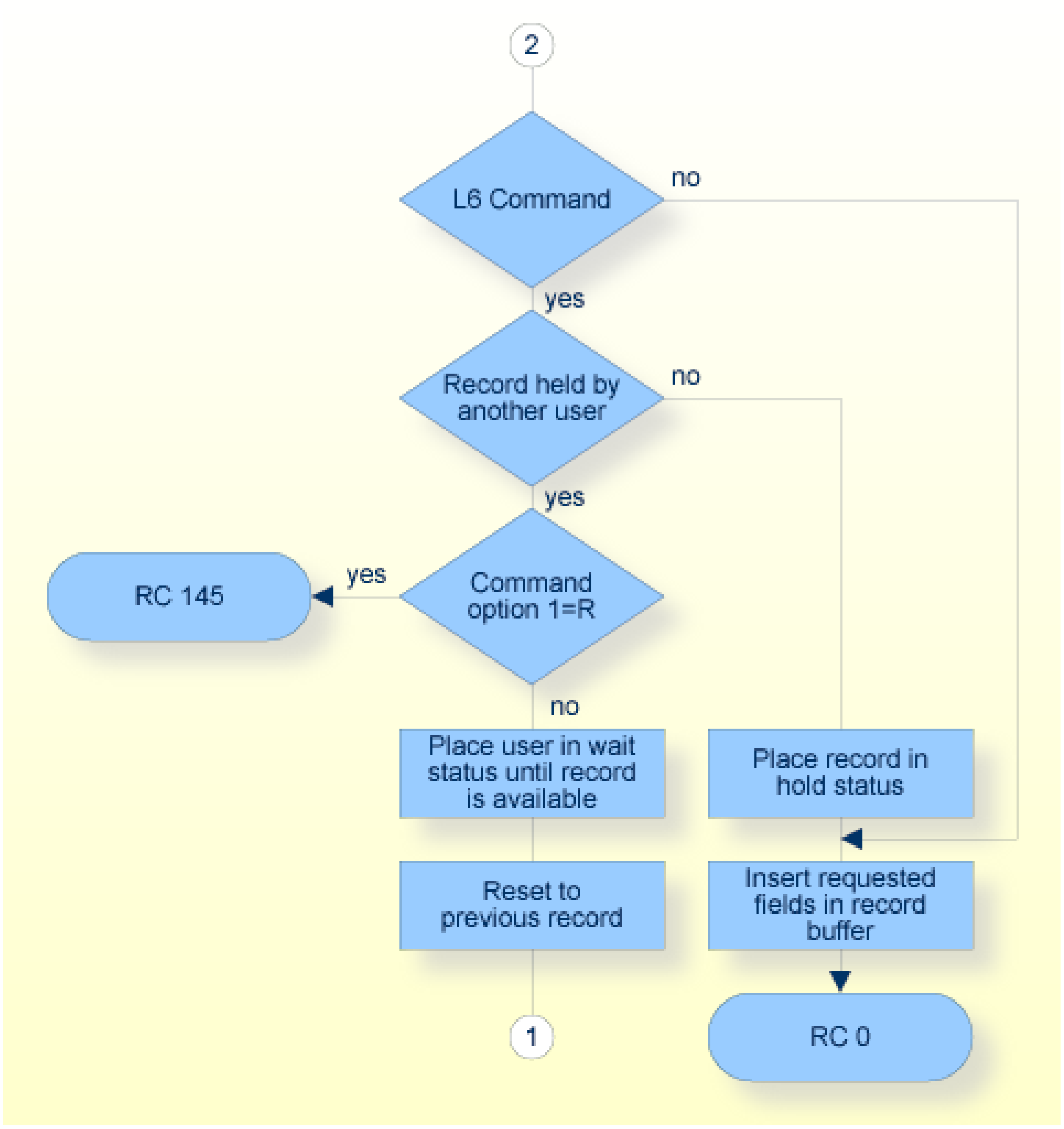
The L3/L6 command supports the Multifetch feature. This is indicated by one of the values `M' or `O' in the Command Option 1 field (see the description of the Command Option 1 field for more details).

When the Multifetch feature is used, the maximum number of records which can be returned for a single L3/L6 call is limited by the following factors:

- The ISN Lower Limit field can be used to specify the maximum number of records to be returned, thus avoiding internal overheads when only a limited number of records are required.
- If the value in the ISN Lower Limit field is 0, the number of records returned is limited only by the size of the ISN/Multifetch Buffer and the Record Buffer.


ADABAS
1
Initial L3/L6 call
yes
Insert CID in table of seq. commands
no
Start value given
yes
Position to value spec. in value buffer
no
Ending value condition true
yes
RC3
no
ASSO
Read record
DATA
2
Physical READ based on buffer management

**L3/L6 Command, Procedure Flow (without Multifetch Feature)**


2
L6 Command
no
yes
Record held by another user
no
yes
RC 145
yes
Command option 1=R
no
Place user in wait status until record is available
Place record in hold status
Reset to previous record
Insert requested fields in record buffer
1
RC 0

**L3/L6 Command, Procedure Flow (continued (without Multifetch Feature)**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN | B | F/A |
| ISN Lower Limit | B | F/U |
| ISN Quantity | B | -/A |
| Format Buffer Length (ACB only) | B | F/U |
| Record Buffer Length (ACB only) | B | F/U |
| Search Buffer Length (ACB only) | B | V F/U |
| Value Buffer Length (ACB only) | B | V F/U |
| ISN Buffer Length (ACB only) | B | F/U |
| Command Option 1 | A | F/U |
| Command Option 2 | A | F/U |
| Command Option 3 (ACBX only) | | |
| Additions 1 | A | F/A |
| Additions 2 | A,B | -/A |
| Additions 3 | A | F/A |
| Additions 5 | A | F/U |
| Command Time | B | -/A |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | F/U |
| Record Buffer | –/A |
| Search Buffer | V F/U |
| Value Buffer | V F/U |
| ISN Buffer | –/A |

**Formats:**

| | |
|---|---|
| A | alphanumeric |

| | |
|---|---|
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| V | Required if starting or ending values or range of values specified |
| – | Not used |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
L3/L6

**Command ID**
This field must be set to a non-blank, non-zero value if a sequential pass is required. The value provided is used by Adabas to provide the records in the correct sequence and to avoid repetitive interpretation of the format buffer.

The high-order byte of this field must not be set to hexadecimal `FF', except when automatic command ID generation is used (see *Programming Considerations, Using Command IDs* for additional information).

This field must not be modified during a given sequential pass of a file.

If only a single record is to be read, this field may be set to blanks or binary zeros.

**File Number**
The number of the file to be read.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

Response code 3 indicates that all entries of the specified descriptor have been processed or that the ending descriptor-value condition has become true.

If the Multifetch feature is used and an error occurs before or while processing the first record, the response code is returned in this field. In this case, the contents of the ISN Buffer and Record Buffer are undefined.

If an error is detected for the second or any subsequent ISN during the processing loop of the Multifetch feature, the first non-zero response code will terminate the multifetch processing. In this case, the response code will be stored as an additional entry in the ISN Buffer itself, not in the Response Code field of the Control Block. Since there are two possible locations for a response code, an application program should check first the Response Code field in the Control Block for errors of a general nature, then in the response code field of each ISN Buffer entry individually.

**ISN**

Together with the starting descriptor value specified in the value buffer, this field is used to determine the start position for the initial L3/L6 call, in cases where several records can have the same descriptor value (matching descriptor-value entries) and when value-repositioning is to be performed. Refer to Example 5, "Using Value Start or Repositioning Option", for more details.

If an ISN of zero is specified, reading starts at the lowest (ascending sequence) or highest (descending sequence) ISN within the matching starting descriptor-value entries.

If a non-zero ISN is specified, reading starts at the closest ISN within the matching starting descriptor-value entries that is greater than (ascending sequence) or less than (descending sequence) the ISN specified in the control block. If the resultant ISN is outside the range of ISNs that match the starting descriptor value, the current position is moved to the next descriptor value and its lowest ISN (ascending sequence) or to the previous descriptor value and its highest ISN (descending sequence).

This field is ignored if the starting descriptor value does not match, or if the GT (ascending sequence) or LT (descending sequence) comparators are specified in the search buffer.

Adabas returns the ISN of the record which has been read in this field.

When the Multifetch feature is used, the first record to be read is determined as described above. The ISNs of the other records read with this L3/L6 command follow according to the logical sequence being used.

**ISN Lower Limit**

If the Multifetch feature is used, the maximum number of records read by a single L3/L6 command is limited to the value specified in this field. If this value is 0, the number of records is limited only by the size of the ISN Buffer and the Record Buffer.

**ISN Quantity**

If the descriptor for which values are to be returned is in a periodic group, Adabas returns the occurrence number in which the returned value is located in this field. If the Multifetch feature is used, this field refers to the first record returned.

**Format Buffer Length (ACB only)**

The format buffer length (in bytes). The format buffer area defined in the user program must be as large as (or larger than) the length specified.

**Record Buffer Length (ACB only)**

The record buffer length (in bytes). The record buffer area defined in the user program must be as large as (or larger than) the length specified.

**Search Buffer Length (ACB only)**

The search buffer length (in bytes) if a starting value, ending value or a range of values has been specified.

The search buffer length is ignored if no value buffer is specified.

**Value Buffer Length (ACB only)**

The value buffer length (in bytes) if a starting value, ending value or a range of values has been specified.

The value buffer length is ignored if no search buffer is specified.

**ISN Buffer Length (ACB only)**

The length of the ISN Buffer. This value must be specified when the Multifetch feature is used. See the description of the ISN Buffer for information on how to calculate the length.

When the Multifetch feature is used, the Record Buffer can contain multiple records. The number of records returned, and the length of each record is stored in a corresponding entry in the ISN Buffer.

**Command Option 1**

An `R' in this field indicates that the Return option is to be used. If an L6 command is issued, and the record to be read and held is currently held by another user, Adabas will return response code 145 rather than placing the user in wait status until the record becomes available.

An `M' in this field indicates that the Multifetch feature is to be used.

If the Multifetch feature is required with the Return option, the value `O' should be used instead of `M'.

**Command Option 2**

The entry in this field specifies the sequence in which the entries for the given descriptor are processed.

An `A' in this field indicates that the descriptor values are to be read in ascending sequence. A starting value, ending value or range of values can be specified optionally. If the search buffer or value buffer is omitted, all of the entries for the given descriptor will be processed.

A `V' in this field indicates that the descriptor values are to be read in ascending sequence with a starting value. Both the search buffer and the value buffer must be present.

A `D' in this field indicates that the descriptor values are to be read in descending sequence. A starting value, ending value or range of values can be specified optionally. If the search buffer or value buffer is omitted, all of the entries for the given descriptor will be processed.

A blank value or any value other than the three specified above indicates that all descriptor values are to be processed in ascending sequence. The search buffer and value buffer will be ignored if they are specified.

When writing new applications, Software AG strongly recommends the use of the value `A' for this field. The use of the value `V' is supported in order to ensure upward compatibility.

When using the values `A' and `D', the processing sequence can be reversed simply by changing the contents of the command option 2 field.

**Command Option 3 (ACBX only)**

The command option 3 is relevant only for an L6 command.

A 'C' in this field indicates that a shared lock is to be acquired for this record only as long as the command is active. If the record was already locked before, the record remains locked. Using this option avoids dirty reads: you see only the committed states of the record.

An 'S' in this field indicates that the record is to be placed in shared hold status. The lock will be released again when the current transaction is committed or backed out. If the command belongs to a subtransaction, the lock is also released when the current subtransaction is backed out. You can also release the lock with an RI command.

A 'Q' in this field indicates that the record is to be placed in shared hold status. The lock is released again at start of next sequential read command for this read sequence, or when one of the events happens that releases a record read with 'S' option, whichever happens first. If the same record is read by more than one command with 'Q' option, the record is released only when, for all these command sequences, either the next record has been read, or an RC command has been issued. If the same record has also been read by another command with 'S' option, or the record has been locked exclusively, the record is not released by reading the next record of the command sequence. The 'Q' option is not allowed in combination with the command option 1 = 'M' (multifetch feature).

A blank in this field indicates that the record is locked exclusively.

**Additions 1**

The descriptor to be used to control the read sequence must be specified in this field. The descriptor name is specified in the first two positions of this field. The remaining positions must be set to blanks on the initial L3 call.

Phonetic descriptors must not be specified.

A descriptor which is a multiple-value field or derived from a periodic group may be specified, in which case the possibility exists that the same record may be read several times (once for each different value within a given record) during the sequential pass of the file.

If the descriptor specified is defined with the null value suppression (NU or NC) option, any records containing a null value for the descriptor will not be read.

If a search buffer is specified, the descriptor(s) in the search buffer must be identical to the descriptor specified in the Additions 1 field.

This field should not be modified by the user between L3/L6 calls. The exception to this rule is when the user wishes to reposition to a particular value during a sequential pass. Setting the last six positions of this field to blanks will cause a new setting of the start and ending value according to the current search and value buffer.

If response 3 occurs, Adabas sets the last six positions of this field to blanks.

**Additions 2**

If the command completes successfully, and at least one Adabas field is requested in the format buffer, Adabas returns in the first two bytes of this field the compressed record length of the data storage record that was accessed, in binary format. The last two bytes contain the length of the decompressed fields selected by the Format Buffer in binary format. If the Multifetch feature is used, this information refers to the first record read.

For some response codes, Adabas returns detailed information in this field. See *Adabas Messages And Codes* for further information.

**Additions 3**

This field is used to provide a security password.

If the file to be used is not security protected, this field should be set to blanks. If the file is security protected, the user must provide a valid password.

Adabas sets this field to blanks during command processing to protect the integrity of any password provided.

**Additions 5**

This field may be used to provide a separate format buffer ID that is used to identify the internal format buffer used for this command, or to provide a global format buffer ID.

As long as the first byte of the Additions 5 field is not alphanumeric, the value provided in the command ID field will also be used as the format buffer ID.

If the first byte is a lower case character, the bytes 5 to 8 of the Additions 5 field will be used as a separate local format buffer ID.

If the first byte is a digit or an upper case character, the Additions 5 field (8 bytes) will be used as a separate global format buffer ID, which means that the format buffer ID can be used by several users in parallel.

See *Programming Considerations, Using Command IDs* for additional information and examples.

## Format Buffer

The fields for which values are to be returned must be specified in this buffer. The syntax and examples of format buffer construction are provided in *Calling Adabas, Format and Record Buffers*.

## Record Buffer

Adabas returns the requested field values in this buffer. The field values are returned according to the standard format and length of each field, unless the user has explicitly requested a different format and/or length in the format buffer.

When the Multifetch feature is used, the Record Buffer can contain data returned from multiple records. The Record Buffer consists of several entries, whereby each entry contains the requested field values from a single record. The number of entries returned, and the length of each entry is stored in a corresponding entry in the ISN Buffer.

## Search Buffer

The search buffer is only required if a starting value, ending value or both are specified in the value buffer in order to limit the number of descriptor entries to be retrieved.

If either the search buffer or value buffer are omitted, all of the values for the given descriptor will be processed.

In a sequence of search buffer elements, a comma is used as separating character and a period as terminating character.

The syntax of the search buffer is provided in *Calling Adabas, Search and Value Buffers*.

## Value Buffer

The value buffer contains the starting value or the ending value or the lower and upper limits of a range in accordance with the specifications contained in the search buffer.

## ISN Buffer/Multifetch Buffer

When the Multifetch feature is used with the ACB buffer, the ISN Buffer contains information which describes the entries returned in the Record Buffer. If you use the Multifetch feature with the ACBX interface, multifetch buffers are used instead.

The first 4 bytes of the respective buffer specify the number of 16–byte entries which follow in the that buffer. Each 16–byte entry corresponds to an entry returned in the Record Buffer, and contains the following unsigned integer values (each 4 bytes long):

- the length of the entry in the Record Buffer
- the response code for the entry in the Record Buffer (this can be different from the value in the Response Code field in the Control Block)
  If this value is non-zero, this means that an error occurred which caused the multifetch processing to be terminated. In this case, there is no corresponding entry in the Record Buffer.
- the ISN
- if the field for which values are to be returned is a descriptor in a periodic group, the occurrence number within the periodic group.

## Additional Considerations

The following additional considerations are applicable for the L3/L6 command:

1. The Command ID used with the L3/L6 command is saved internally and used by Adabas. It will be released by Adabas when an end-of-file condition is detected, an RC or CL command is issued, or the Adabas session is terminated. The same command ID may not be used by the user for another read sequential command until it has been released.

   Command IDs are not released at the end of a global transaction.
2. The user is permitted to update and/or delete records from a file which is being read by the user with an L3/L6 command. Adabas maintains information about the last and next record to be provided to the user and is able to provide the correct next record despite any interim record update or deletion performed by the user.
3. If another user is updating the file being read with an L3/L6 command, it is possible that the user reading with the L3/L6 command will not receive one or more records in the file, or may receive the same record more than once, during a given sequential pass of the file.

## Examples

**Example 1:**

File 2 is to be read in logical sequential order. The descriptor RB is to be used for sequence control. The values for the fields RA and RB are to be returned. The entire file is to be read.

**Control Block:**

```
Command Code          L3

Command ID            EX01 (a non-blank CID is required)

File Number           2

Format Buffer Length  6 (or larger)

Record Buffer Length  18 (or larger)

Command Option 1      b (Return option not used)

Command Option 2      A

Additions 1           RBbbbbbb  (descriptor  RB  to be used for
                                 sequence control)

Additions 3           Password (file 2 is security protected)
```

**Buffer Areas:**

```
Format Buffer         RA,RB.
```

The L3 call is repeated to obtain each successive record. The CID and ADDITIONS 1 field must not be modified between L3 calls. Response code 3 will be returned when all records have been read.

**Example 2: Read in logical sequence with hold**

File 1 is to be read in logical sequential order. The descriptor AA is to be used for sequence control. The values for fields AA and AB are to be returned. Each record read is to be placed in hold status.

**Control Block:**

```
Command Code          L6

Command ID            EX02 (non blank CID required)

File Number           1

Format Buffer Length  3 (or larger)

Record Buffer Length  10 (or larger)

Command Option 1      b (Return option not used)

Command Option 2      A

Additions 1           AAbbbbbb  (descriptor  AA  to be used for
                                 sequence control)
```

```
Additions 3           bbbbbbbb (file 1 is not security protected)
```

**Buffer Areas:**

```
Format Buffer         GA.
```

Each record read should be released by an ET command (ET logic users), or RI command (non–ET logic users).

**Example 3: Starting with a given value**

The same requirement as Example 1 except that reading is to begin with the value `N'.

**Control Block:**

```
Command Code          L3
```

```
Command ID            EX03 (non blank CID required)
```

```
File Number           2
```

```
ISN                   0 (reading is to start with the lowest ISN that
                         contains the value given in the value buffer)
```

```
Format Buffer Length  6 (or larger)
```

```
Record Buffer Length  18 (or larger)
```

```
Search Buffer Length  7 (or larger)
```

```
Value Buffer Length   1 (or larger)
```

```
Command Option 1      b (Return option not used)
```

```
Command Option 2      A
```

```
Additions 1           RBbbbbbb  (RB  is to be used for sequence
                                control)
```

```
Additions 3           Password (file 2 is security protected)
```

**Buffer Areas:**

```
Format Buffer         RA,RB.
```

```
Search Buffer         RB,1,A. (GE is implicit)
```

```
Value Buffer          N (reading to begin with value N)
```

The initial L3 call will result in the return of the first record which contains the value N for the descriptor RB. If no records exist with this value, the first record which contains a value greater than N (such as NA) will be returned.

**Example 4: Value repositioning**

The same requirement as example 3 except that a value repositioning is to be performed. The sequential read process is to continue with the value Q.

**Control Block:**

```
Command Code          L3
```

```
Command ID            EX03 (the CID field  may  not  be changed
                      when performing a value repositioning)
```

|  |  |
|---|---|
| File Number | 2 |
| ISN | 0 (position to the lowest ISN that contains the value specified in the value buffer) |
| Format Buffer Length | 6 (or larger) |
| Record Buffer Length | 18 (or larger) |
| Search Buffer Length | 7 (or larger) |
| Value Buffer Length | 1 (or larger) |
| Command Option 1 | b (Return option not used) |
| Command Option 2 | A |
| Additions 1 | RBbbbbbb (the last six positions of this field must be set to blank for value repositioning) |
| Additions 3 | Password (file 2 is security protected) |

**Buffer Areas:**

|  |  |
|---|---|
| Format Buffer | RA,RB. |
| Search Buffer | RB,1,A. |

```
Value Buffer          Q (repositioning is to be to value Q)
```

**Example 5: Using Value Start or Repositioning Option**

Inverted list for the descriptor used for sequence control:

```
Value               ISN List
```

```
  A                   1,4
  B                   2
  D                   3,5
```

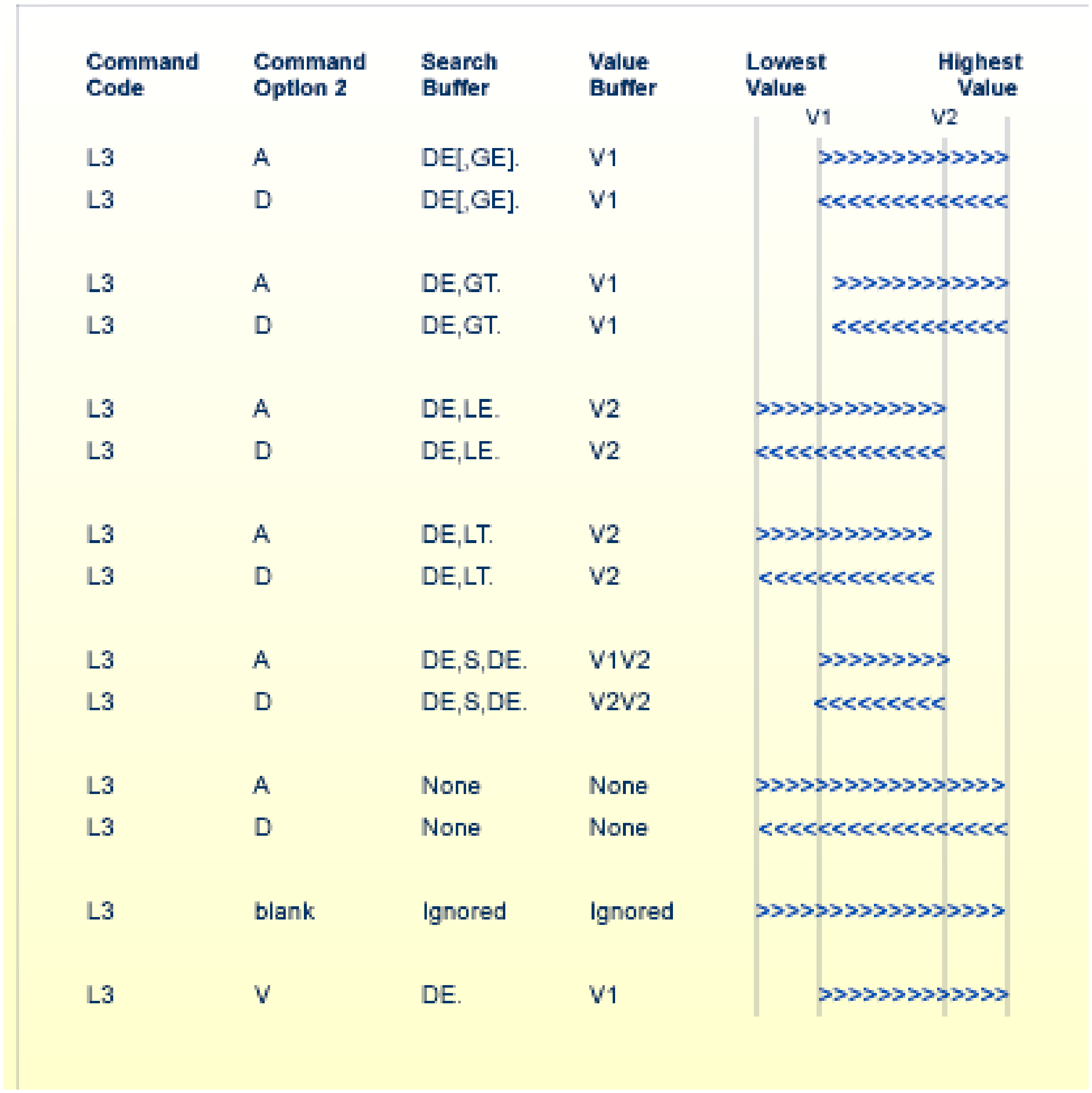
Initial L3/L6 command with Command Option 2 = A, or subsequent L3/L6 command with Command Option 2 = A and last six bytes of Additions 1 reset to blanks (value repositioning):

| User-supplied Value | User-supplied ISN | ISN with which reading will start (or continue) |
|---|---|---|
| A | 0 | 1 |
| A | 1 | 4 |
| A | 2 | 4 |
| A | 4 | 2 |
| A | 5 | 2 |
| B | 0 | 2 |
| B | 1 | 2 |
| B | 2 | 3 |
| B | 3 | 3 |
| BABC | 1 | 3 |
| C | 0 | 3 |
| D | 0 | 3 |
| D | 3 | 5 |
| D | 4 | 5 |
| D | 5 | Response Code 3 |
| E–Z | – | Response Code 3 |

**Example 6: Using The Ascending/Descending Option**

The following example illustrates the possibilities of using the ascending/descending option in conjunction with various search buffer and value buffer contents. The following applies to the file being read:

- the values V1 and V2 are both present, otherwise the "GE" and "LE" examples would be the same as the "GT" and "LT" examples;
- ISN=0 (no further positioning within matching start values).



| Command Code | Command Option 2 | Search Buffer | Value Buffer | Lowest Value ... Highest Value (V1, V2) |
|---|---|---|---|---|
| L3 | A | DE[,GE]. | V1 | >>>>>>>>>>>>> |
| L3 | D | DE[,GE]. | V1 | <<<<<<<<<<<<< |
| L3 | A | DE,GT. | V1 | >>>>>>>>>>>> |
| L3 | D | DE,GT. | V1 | <<<<<<<<<<<< |
| L3 | A | DE,LE. | V2 | >>>>>>>>>>>>> |
| L3 | D | DE,LE. | V2 | <<<<<<<<<<<<< |
| L3 | A | DE,LT. | V2 | >>>>>>>>>>>> |
| L3 | D | DE,LT. | V2 | <<<<<<<<<<<< |
| L3 | A | DE,S,DE. | V1V2 | >>>>>>>>> |
| L3 | D | DE,S,DE. | V2V2 | <<<<<<<<< |
| L3 | A | None | None | >>>>>>>>>>>>>>>>> |
| L3 | D | None | None | <<<<<<<<<<<<<<<<< |
| L3 | blank | Ignored | Ignored | >>>>>>>>>>>>>>>>> |
| L3 | V | DE. | V1 | >>>>>>>>>>>>> |

# 18 L9 Command (Read Descriptor Values)

## Function and Use

The L9 command is used to determine the range of values present for a descriptor and the number of records which contain each value. With the Multifetch feature, it is also possible to return information for more than one descriptor value with a single call of the L9 command.

Adabas determines this information by reading the Associator inverted lists. No access to Data Storage is required.

**Without the Multifetch Feature**

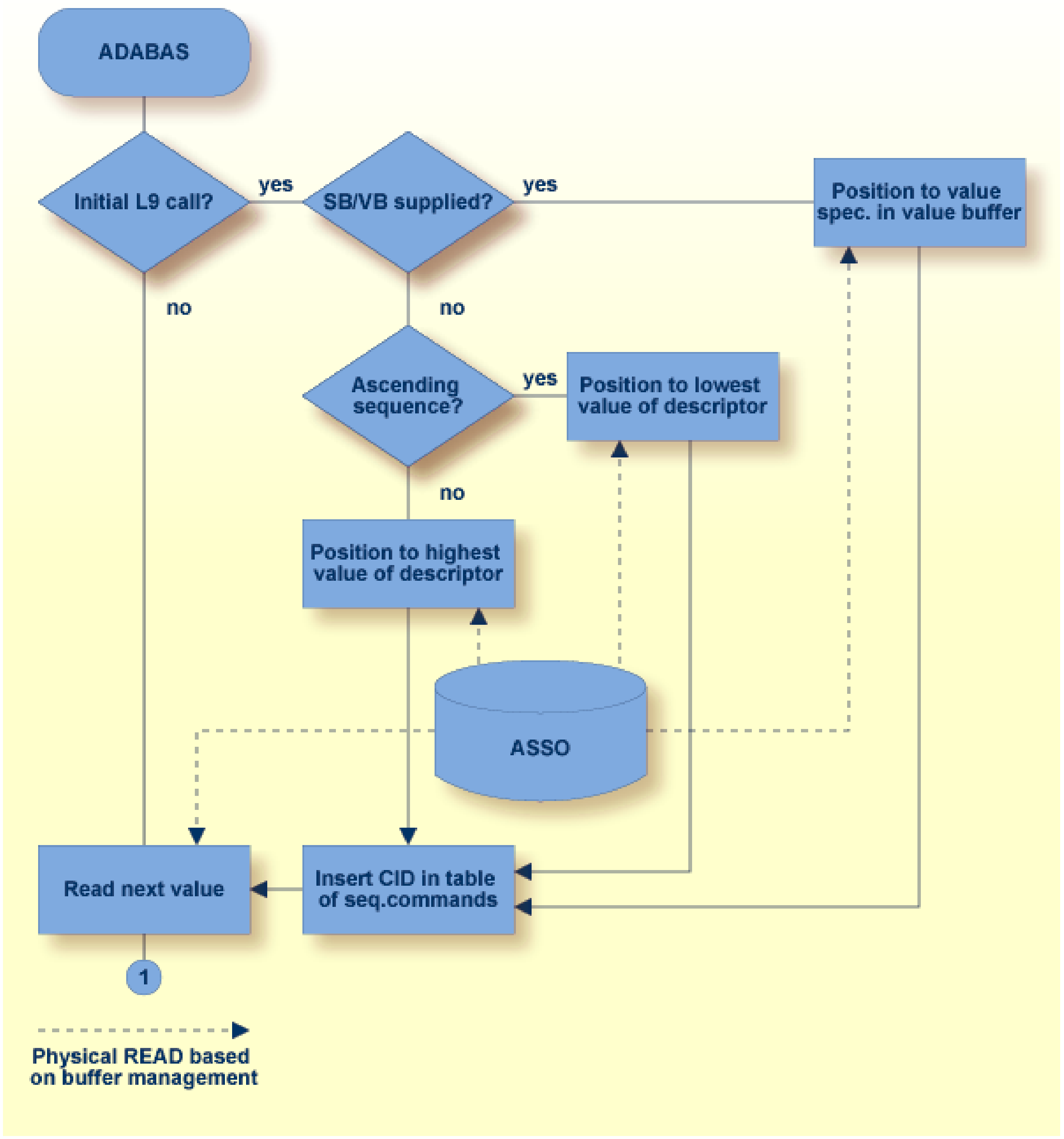
The user specifies the file in which the descriptor is contained, the descriptor for which the values are to be returned and the value at which processing is to begin. Adabas returns (with each L9 call) the next value for the descriptor in the record buffer, along with the number of records containing that value in the ISN quantity field of the control block. The values are provided in ascending or descending order. Null values for descriptors defined with the null value suppression option are not returned.

**With the Multifetch Feature**

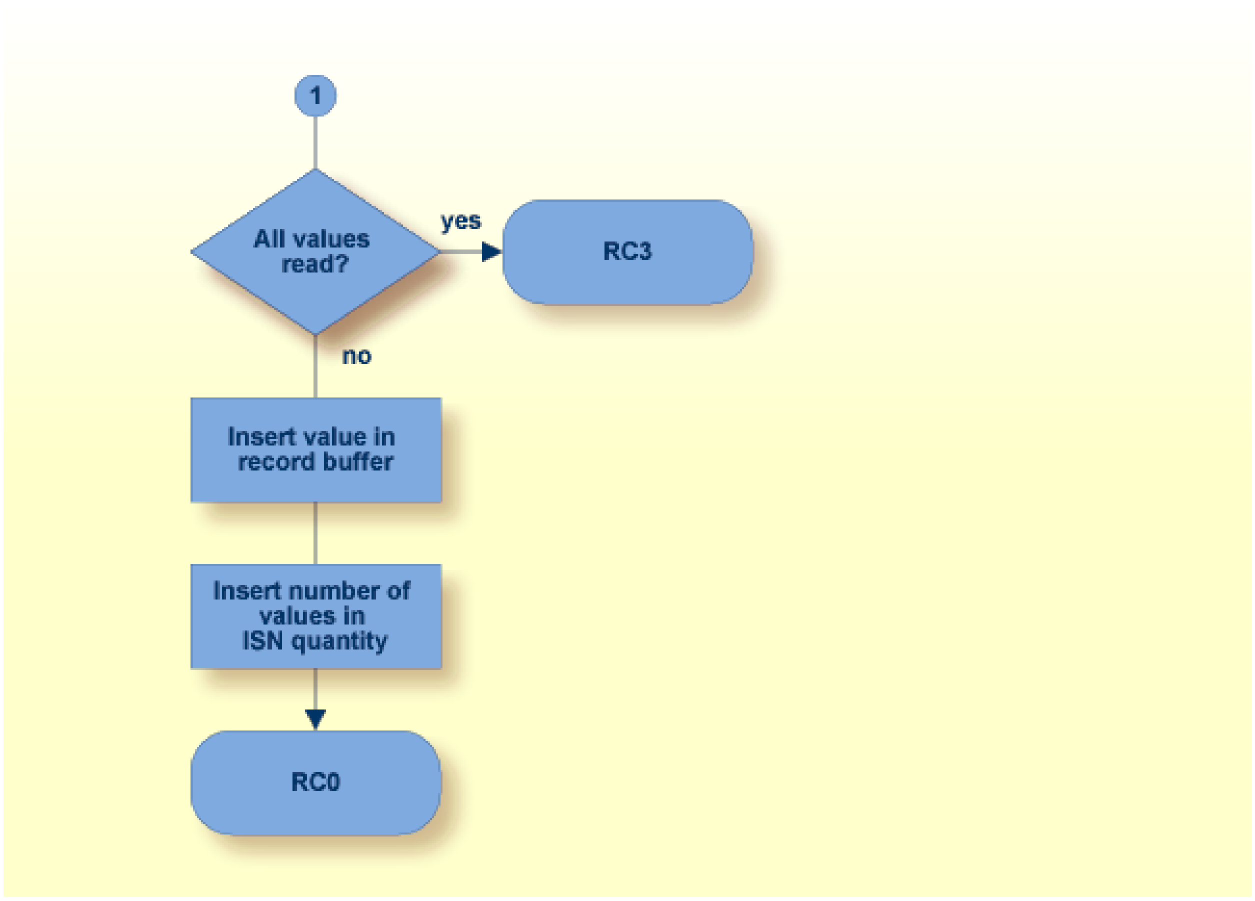
The L9 command supports the Multifetch feature. This is indicated by the value `M' in the Command Option 1 field (see the description of the Command option 1 field for more details).

When the Multifetch feature is used, the maximum number of descriptor values which can be processed by a single L9 call is limited by the following factors:

- The ISN Lower Limit field can be used to specify the maximum number of descriptor values to process, thus avoiding internal overheads when only a limited number of descriptors are required.
- If the value in the ISN Lower Limit field is 0, the number of descriptor values processed is limited only by the size of the ISN/Multifetch Buffer and the Record Buffer.


ADABAS
Initial L9 call?
yes
no
SB/VB supplied?
yes
no
Position to value spec. in value buffer
Ascending sequence?
yes
no
Position to lowest value of descriptor
Position to highest value of descriptor
ASSO
Read next value
Insert CID in table of seq.commands
1
Physical READ based on buffer management

**L9 Command, Procedure Flow**


1
All values read?
yes
RC3
no
Insert value in record buffer
Insert number of values in ISN quantity
RC0

**L9 Command, Procedure Flow (continued)**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN | B | -/A |
| ISN Lower Limit | B | F/U |
| ISN Quantity | B | -/A |
| Format Buffer Length (ACB only) | B | F/U |
| Record Buffer Length (ACB only) | B | F/U |
| Search Buffer Length (ACB only) | B | F/U |
| Value Buffer Length (ACB only) | B | F/U |
| ISN Buffer Length (ACB only) | B | F/U |
| Command Option 1 | A | F/U |
| Command Option 2 | A | F/U |
| Additions 1 | A | * F/U |
| Additions 2 | A,B | -/A |
| Additions 3 | A | F/A |
| Additions 5 | A | F/U |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | F/U |
| Record Buffer | –/A |
| Search Buffer | * F/U |
| Value Buffer | * F/U |
| ISN Buffer | –/A |

**Formats:**

A alphanumeric

| | |
|---|---|
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |
| * | optional |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
L9

**Command ID**
This field must be set to a non-blank, non-zero value if a sequential pass is required. This value is used by Adabas to provide the values in the correct sequence and to avoid the repetitive interpretation of the format buffer.

The high-order byte of this field must not be set to hexadecimal `FF', except when automatic command ID generation is used (see *Programming Considerations, Using Command IDs* for additional information).

This field must not be modified during a given sequential pass of the file.

If only a single index value is to be read, this field may be set to blanks or binary zeros.

**File Number**
The number of the file to be read.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

Response code 3 indicates that all entries of the specified descriptor have been processed or that the ending descriptor-value condition has become true.

If the Multifetch feature is used and an error occurs before or while processing the first record, the response code is returned in this field. In this case, the contents of the ISN/Multifetch Buffer and Record Buffer are undefined.

If an error is detected for the second or any subsequent ISN during the processing loop of the Multifetch feature, the first non-zero response code will terminate the multifetch processing. In this case, the response code will be stored as an additional entry in the ISN/Multifetch Buffer itself, not in the Response Code field of the Control Block. Since there are two possible locations for a response code, an application program should check first the Response Code field in the Control Block for errors of a general nature, then in the response code field of each ISN/Multifetch Buffer entry individually.

**ISN**

If the descriptor for which values are to be returned is contained within a periodic group, this field is used

- on the initial call to specify the occurrence number for which values are to returned if the search/value buffer is not defined. Otherwise the search buffer is used to determine the occurrence number, and the content of the ISN field is ignored. A value of zero indicates that all occurrences are to be returned. A non–zero value indicates that only values of the specified occurrence are to be returned.
- by Adabas to return the occurrence number of the value being returned in the record buffer as a four–byte integer.

When the Multifetch feature is used, the ISN field refers to the first descriptor value which will be processed by the L9 command.

**ISN Lower Limit**

If the Multifetch option is used, this field is used to limit the number of records to be returned. If this field is set to zero, the maximum number of records returned depends on the size of the record buffer and/or the size of the ISN buffer.

**ISN Quantity**

Adabas returns in this field the number of records containing the value returned in the record buffer. If the Multifetch feature is used, this field refers to the first descriptor value returned.

**Format Buffer Length (ACB only)**

The format buffer length (in bytes). The format buffer area defined in the user program must be as large as (or larger than) the length specified.

**Record Buffer Length (ACB only)**

The record buffer length (in bytes). The record buffer area defined in the user program must be as large as (or larger than) the length specified.

**Search Buffer Length (ACB only)**

The search buffer length (in bytes) if a starting value, ending value or a range of values has been specified.

If no search buffer length is supplied, the command starts from the lowest value.

**Value Buffer Length (ACB only)**

The value buffer length (in bytes) if a starting value, ending value or a range of values has been specified.

The value buffer length is ignored if no search buffer is specified.

**ISN Buffer Length (ACB only)**

The length of the ISN Buffer. This value must be specified when the Multifetch feature is used. See the description of the ISN Buffer for information on how to calculate the length.

When the Multifetch feature is used, the Record Buffer can contain data for multiple descriptor values. The number of descriptor values returned, and the number of bytes of data returned for each descriptor is stored in a corresponding entry in the ISN Buffer.

**Command Option 1**

An `M' in this field invokes the Multifetch feature. See *Programming Considerations, Using The Multifetch Feature* for more detailed information.

**Command Option 2**

An `A' in this field indicates that the descriptor values are to be read in ascending sequence. A starting value, ending value or range of values can be specified optionally. If the search buffer or value buffer is omitted, all of the entries for the given descriptor will be processed.

A `D' in this field indicates that the descriptor values are to be read in descending sequence. A starting value, ending value or range of values can be specified optionally. If the search buffer or value buffer is omitted, all of the entries for the given descriptor will be processed.

When using the values `A' and `D', the processing sequence can be reversed simply by changing the content of the command option 2 field.

When writing new applications, Software AG strongly recommends the use of the values `A' and `D' for this field. The use of other values is supported in order to ensure upward compatibility.

If the field contains a blank value or any value other than `A' or `D', the descriptor values are processed as if the value `A' had been specified.

**Additions 1**

If no search buffer or no value buffer has been supplied, Additions 1 must contain the name of the descriptor to be used to control the read sequence. The descriptor name is specified in the first two positions of this field. The remaining positions must be set to blanks. The descriptor name specified in Additions 1 must be the same as the descriptor name specified in the Format Buffer.

Phonetic descriptors must not be specified.

**Additions 2**

For some response codes, Adabas returns detailed information in this field. See *Adabas Messages And Codes* for further information.

**Additions 3**

This field is used to provide a security password.

If the file to be used is not security protected, this field should be set to blanks. If the file is security protected, the user must provide a valid password.

Adabas sets this field to blanks during command processing to protect the integrity of any password provided.

**Additions 5**

This field may be used to provide a separate format buffer ID that is used to identify the internal format buffer used for this command, or to provide a global format buffer ID.

As long as the first byte of the Additions 5 field is not alphanumeric, the value provided in the command ID field will also be used as the format buffer ID.

If the first byte is a lower case character, the bytes 5 to 8 of the Additions 5 field will be used as the separate local format buffer ID.

If the first byte is a digit or an upper case character, the Additions 5 field (8 bytes) will be used as a separate global format buffer ID, which means that the format buffer ID can be used by several users in parallel.

See *Programming Considerations, Using Command IDs* for additional information and examples.

## Format Buffer

The format in which the values are to be returned must be specified in this buffer.

The syntax of the format buffer is:

```
name[,length][,format].
```

**name**

The name of the descriptor for which values are to be returned. A phonetic descriptor may not be specified.

**length**

The length in which the value is to be returned. If length is not specified, the value will be returned using the default length of the descriptor as specified in the FDT.

**format**

The format in which the value is to be returned. The format specified must be compatible with the standard format of the descriptor. If the format is not specified, the value will be returned using the default format of the descriptor as specified in the FDT.

## Record Buffer

Adabas returns the descriptor values requested in this buffer. A different value is provided with each L9 call. The values are provided in ascending or descending sequence. If the descriptor is defined with the null value suppression option, the null value for the descriptor will not be returned.

When the Multifetch feature is used, the Record Buffer can contain data for multiple descriptor values. The number of descriptor values processed, and information concerning each descriptor value, is stored in the ISN/Multifetch Buffer.

## Search Buffer

The search buffer is only required if a starting value, ending value or both are specified in the value buffer in order to limit the number of descriptor entries to be retrieved.

If either the search buffer or the value buffer is omitted, all of the values for a given descriptor (or all of the values for a given occurrence) will be processed. In this case, the Additions 1 field must contain the descriptor name.

The syntax of the search buffer is provided in *Calling Adabas, Search and Value Buffers*.

## Value Buffer

The value buffer contains the starting value or the ending value or the lower and upper limits of a range in accordance with the specifications contained in the search buffer.

## ISN Buffer/Multifetch Buffers

When the Multifetch feature is used with the ACB interface, the ISN Buffer contains information which describes the descriptor entries returned in the Record Buffer. If you use the Multifetch feature with the ACBX interface, multifetch buffers are used instead.

The first 4 bytes of the respective buffer specify the number of 16-byte entries which follow in the that buffer. Each 16-byte entry corresponds to a descriptor entry returned in the Record Buffer, and contains the following unsigned integer values (each 4 bytes long):

- the number of bytes of information in the Record Buffer for the descriptor value

- the response code for the descriptor value (this can be different from the value in the Response Code field in the Control Block)
  If this value is non-zero, this means that an error occurred which caused the multifetch processing to be terminated. In this case, there is no corresponding entry for the descriptor value in the Record Buffer.
- if the descriptor is in a periodic group, the occurrence number within the periodic group.
- the number of ISNs found for the descriptor

## Additional Considerations

The following additional considerations are applicable for the L9 command:

1. The Command ID used with the L9 command is internally saved and used by Adabas. It will be released by Adabas when an end-of-file condition is detected, an RC or CL command is issued or the Adabas session is terminated. The same command ID may not be used by the user for another command until it has been released.
2. The user is permitted to update and/or delete records from a file which is being read by the user with an L9 command. Adabas maintains information about the last and next value to be provided to the user and is able to provide the correct next value despite any interim record update or deletion performed by the user.
3. If another user is updating the file being read with an L9 command, it is possible that the user reading with the L9 command will not receive one or more values in the file.

## Examples

**Example 1:**

The values for the descriptor RB in file 2 are to be returned. All values are to be returned.

**Control Block:**

```
Command Code          L9
Command ID            L901 (a non blank CID is required)
```

```
File Number           2
```

```
Format Buffer Length  3 (or larger)
```

```
Record Buffer Length  10 (or larger)
```

```
Search Buffer Length  5 (or larger)
```

```
Value Buffer Length   1 (or larger)
```

```
Command Option 2      A
```

```
Additions 3           Password (file 2 is security protected)
```

**Buffer Areas:**

```
Format Buffer         RB.  (the values are to be returned using
                            standard length and format)
```

```
Search Buffer         RB,1. (the values for descriptor  RB  are
                             to be returned, and the starting value is
                             being provided with standard  format  and
                             length = 1)
```

```
Value Buffer          b    (processing is to begin with the first
                             value for RB equal or greater than 'b'
```

Each successive L9 call will result in the return of the next value (values are provided in ascending order). The number of records containing the value is returned in the ISN Quantity field.

**Example 2:**

The values for descriptor AB in file 1 are to be returned. Only values which are equal to or greater than 20 are to be returned.

**Control Block:**

| | |
|---|---|
| Command Code | L9 |
| Command ID | L902 (a non blank CID is required) |
| File Number | 1 |
| Format Buffer Length | 7 (or larger) |
| Record Buffer Length | 3 (or larger) |
| Search Buffer Length | 7 (or larger) |
| Value Buffer Length | 2 (or larger) |
| Command Option 2 | A |
| Additions 3 | bbbbbbbb (file 1 is not security protected) |

**Buffer Areas:**

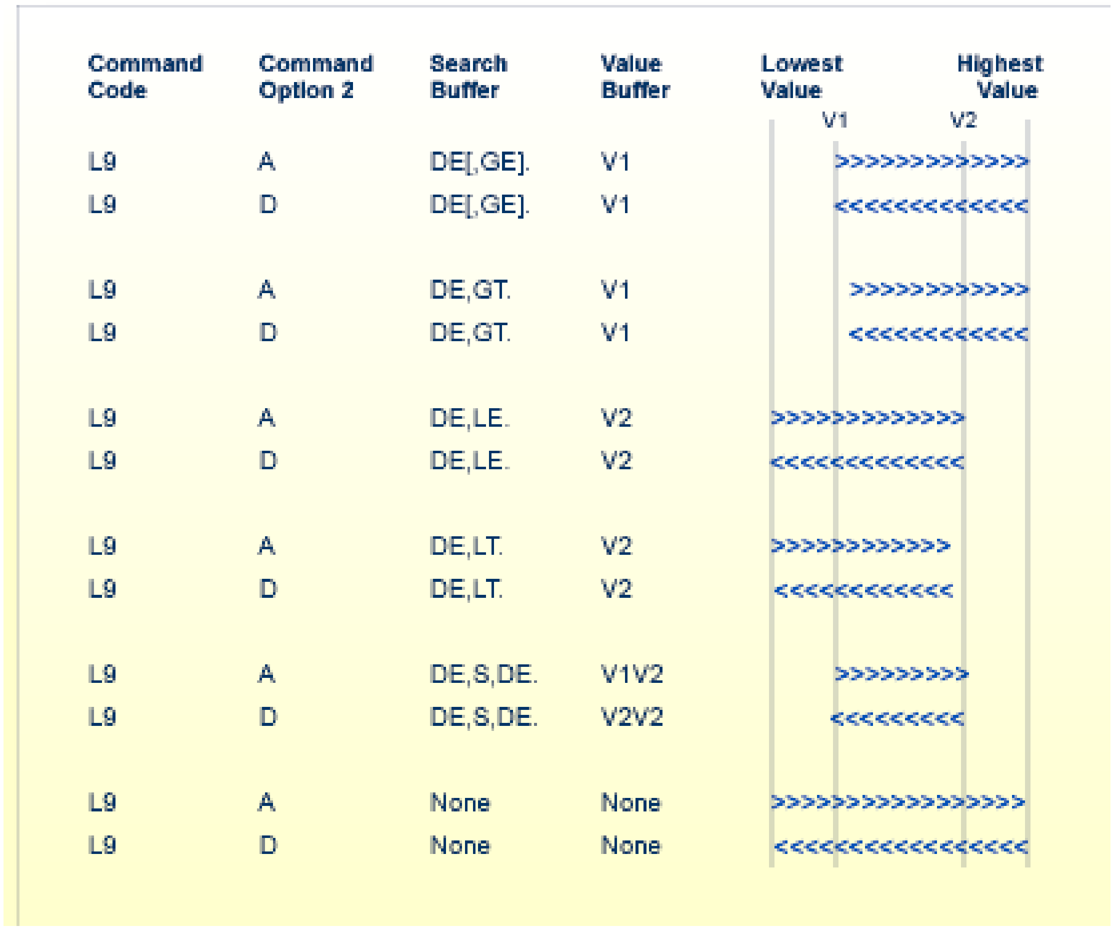
| | |
|---|---|
| Format Buffer | AB,3,U. (the values are to be returned with length = 3 and format = unpacked) |
| Search Buffer | AB,2,U. (the values for the descriptor AB are to be returned and the starting value is to be provided as a 2 byte unpacked number) |

```
Value Buffer          0x3230 (Processing  is to begin with the
                      first value for AB which is  equal to  or
                      greater than 20)
```

**Example 3: Using The Ascending/Descending Option**

The following example illustrates the possibilities of using the ascending/descending option in conjunction with various search buffer and value buffer contents. The following applies to the file being read:

- the values V1 and V2 are both present, otherwise the "GE" and "LE" examples would be the same as the `GT' and `LT' examples;



| Command Code | Command Option 2 | Search Buffer | Value Buffer | Lowest Value … Highest Value (V1, V2) |
|---|---|---|---|---|
| L9 | A | DE[,GE]. | V1 | >>>>>>>>>>>>> |
| L9 | D | DE[,GE]. | V1 | <<<<<<<<<<<<< |
| L9 | A | DE,GT. | V1 | >>>>>>>>>>>> |
| L9 | D | DE,GT. | V1 | <<<<<<<<<<<< |
| L9 | A | DE,LE. | V2 | >>>>>>>>>>>>> |
| L9 | D | DE,LE. | V2 | <<<<<<<<<<<<< |
| L9 | A | DE,LT. | V2 | >>>>>>>>>>>> |
| L9 | D | DE,LT. | V2 | <<<<<<<<<<<< |
| L9 | A | DE,S,DE. | V1V2 | >>>>>>>>> |
| L9 | D | DE,S,DE. | V2V2 | <<<<<<<<< |
| L9 | A | None | None | >>>>>>>>>>>>>>>>> |
| L9 | D | None | None | <<<<<<<<<<<<<<<<< |

# 19 LF Command (Read Field Definitions)

## Function and Use

The LF command is used to read the field definition information for a file.

The user specifies the file number for which the field definitions are to be returned.

Adabas returns the field definition information in the record buffer. The following information is returned for each field:

- Level number;
- Name;
- Standard length;
- Standard format;
- Definition options.

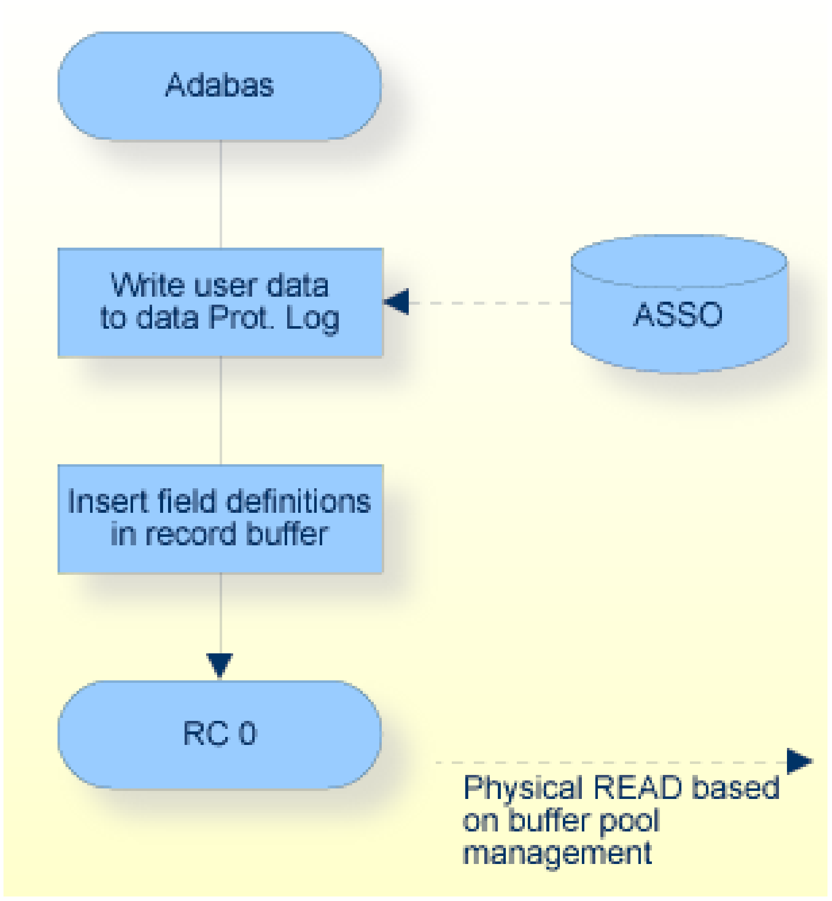
This command can be used to obtain the standard field length and format of a field during program execution in order to dynamically create a format or search buffer.

The timestamp when the field definitions were last updated is returned in the record buffer when the command option 2 = ‘X’. This may be one of the following:

- The time when the file was created;
- The time when a field was last added, changed or deleted;
- A descriptor was last inverted or released.

If a file is imported with ADAORD or restored with ADABCK, the timestamp of the import or restore is not stored; the original timestamp of the file when it was exported or dumped is kept.

**Note:** The LF command only reads meta information on the file, but not the file itself. For this reason, an LF command can also be performed on a file that is currently exclusively locked by an application program or which is locked by a utility.


Adabas
Write user data to data Prot. Log
ASSO
Insert field definitions in record buffer
RC 0
Physical READ based on buffer pool management

**LF Command, Procedure Flow**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | -/- |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| Record Buffer Length (ACB only) | B | F/U |
| Command Option 2 | A | F/U |
| Additions 2 | A,B | -/A |
| Additions 3 | A | F/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | */– |
| Record Buffer | –/A |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | –/– |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |
| * | Not used but must be included in parameter list of CALL statement |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
LF

**File Number**
The number of the file for which the field definition information is to be returned.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**Record Buffer Length (ACB only)**
The record buffer length (in bytes). The length specified must be large enough to contain all field definition information for the file and must not be greater than the size of the record buffer area defined in the user program.

**Command Option 2**
The setting of this field determines the format and type of information to be returned in the record buffer.

An ‘S’ in this field returns the standard information on the file that was available already with Adabas Version 6.1. However, this information is not complete; in particular, the new features introduced with Adabas Version 6.2 are not supported.

An ‘X’ in this field returns extended information on the file in the record buffer. This contains all information required to create a file with the same FDT.

An ‘F’ in this field returns full information on the FDT. It is similar to the ‘X’ option but with the difference that logically-deleted fields are also returned in the record buffer.

**Notes:**

1. You can only load data with the ADAMUP utility if the FDT of the decompressed data is the same as the FDT of the target file, including the logically-deleted fields.
2. Logically-deleted fields are returned in the record buffer without field names.

A blank in this field returns only the basic information on the fields of the file; only a subset of the options is returned. No information on special descriptors is returned.

**Additions 2**
Adabas returns the total length of the returned structure in this field.

For some response codes, Adabas returns detailed information in this field. See *Adabas Messages And Codes* for further information.

**Additions 3**

This field is used to provide a security password.

If the file to be used is not security protected, this field should be set to blanks. If the file is security protected, the user must provide a valid password.

Adabas sets this field to blanks during command processing to protect the integrity of any password provided.

## Record Buffer for Command Option 2 = 'S'

All field definition information is returned in the record buffer.

Setting the Command Option 2 field to `S' returns the information in the following format:

### Record Buffer General Layout

| Bytes | Usage |
|---|---|
| 1 - 2 | Total length of information |
| 3 - 4 | Number of fields in the FDT (including SDT) |
| 5 -N | Field definitions, each 8 bytes long |
| N - M | SDT (Special Descriptor Table)<br>Special descriptors are:<br>subdescriptors<br>superdescriptors<br>phonetic descriptors<br>hyperdescriptors<br>collation descriptors<br>For SDT definitions, an SDT element has an integral length and is a multiple of 8 bytes. |

## FDT Field Definitions

| Bytes | Usage |
|---|---|
| 1 | Indicator field name 'F' |
| 2 - 3 | Field name |
| 4 | Definition options<br>(00 = no definition options)<br>0x 01 = unique descriptor<br>0x 02 = parent of sub-/superdescriptor<br>0x 04 = parent of phonetic descriptor<br>0x 08 = PE group field/ PE inner group/ elementary field within PE group<br>0x 10 = null suppression<br>0x 20 = multiple field<br>0x 40 = fixed format<br>0x 80 = descriptor |
| 5 | Level number (binary) |
| 6 | Length (group = 0) |
| 7 | Format<br>(Space (no format) for group)<br>A = alpha<br>B = binary<br>F = fixed point<br>G = floating point<br>P = packed decimal<br>U = unpacked decimal<br>W = Unicode |
| 8 | 0x01 = NC option is active<br>0x02 = NN option is active<br>0x04 = LB option is active<br>0x08 = LA option is active<br>0x20 = HF option is active<br>0x40 = NV option is active<br>0x80 = NB option is active |

**Caution:** In order to be mainframe-compatible, the bit for the NV option has been changed: in Adabas Version 3.3 and earlier, NV option = 0x04; in Adabas Version 5.1 and higher and Adabas on mainframes, NV option = 0x40.

## SDT Field Definitions

| Byte 1 | Indicates |
|---|---|
| C | Collation descriptor, see collation descriptor definition |
| H | Hyperdescriptor, see hyperdescriptor definition |
| P | Phonetic descriptor, see phonetic descriptor definition |
| S | Subdescriptor, see subdescriptor definition |
| T | Superdescriptor, see superdescriptor definition |

## Collation Descriptor Definitions

| Bytes | Usage |
|---|---|
| 1 | C indicates collation descriptor |
| 2 - 3 | Collation descriptor name |
| 4 | Definition options<br>0x 01 = unique descriptor<br>0x 02 = unused<br>0x 04 = no HE option<br>0x 08 = periodic<br>0x 10 = NU<br>0x 20 = multiple field<br>0x 40 = unused<br>0x 80 = unused |
| 5 | Unused |
| 6 | Length (255 means length > 254) |
| 7 - 8 | Parent field name |

## Hyperdescriptor Definitions

| Bytes | Usage |
|---|---|
| 1 | H indicates hyperdescriptor |
| 2 - 3 | Hyperdescriptor name |
| 4 | Definition options<br>0x 01 = unique descriptor<br>0x 02 = unused<br>0x 04 = HE option used<br>0x 08 = periodic<br>0x 10 = null suppression<br>0x 20 = multiple field<br>0x 40 = unused<br>0x 80 = unused |
| 5 | Hyperexit number |
| 6 | Length |
| 7 | Format<br>A = alphanumeric<br>B = binary<br>F = fixed point<br>G = floating point<br>P = packed decimal<br>U = unpacked decimal<br>W = Unicode |
| 8 | Unused |
| 1 - 2 | 0x0000 indicates continuation |
| 3 - 8 | parent field name list (2 bytes each) |

## Phonetic Descriptor Definition

| Bytes | Usage |
|---|---|
| 1 | P indicates phonetic descriptor |
| 2 - 3 | Phonetic descriptor name |
| 4 | Unused |
| 5 - 6 | Parent field name |
| 7 -8 | Unused |

## Subdescriptor Definition

| Bytes | Usage |
|---|---|
| 1 | S indicates subdescriptor |
| 2 - 3 | Subdescriptor name |
| 4 | Definition options<br>0x 01 = unique descriptor<br>0x 02 = unused<br>0x 04 = unused<br>0x 08 = periodic<br>0x 10 = null suppression<br>0x 20 = multiple field<br>0x 40 = unused<br>0x 80 = descriptor |
| 5 - 6 | Parent field name |
| 7 | From byte |
| 8 | To byte |

## Superdescriptor Definition

| Bytes | Usage |
|---|---|
| 1 | T indicates superdescriptor |
| 2 - 3 | Superdescriptor name |
| 4 | Definition options<br>0x 01 = unique descriptor<br>0x 02 = unused<br>0x 04 = unused<br>0x 08 = periodic<br>0x 10 = null suppression<br>0x 20 = multiple field<br>0x 40 = unused<br>0x 80 = descriptor |
| 5 - 6 | Parent field 1 |
| 7 | From byte |
| 8 | To byte |

For the second and subsequent parent fields, the following table applies:

| Bytes | Usage |
|---|---|
| 1 | 0x00 indicates continuation |
| 2 - 3 | Unused |
| 4 | Format of superdescriptor |
| 5 - 6 | Parent field name |
| 7 | From byte |
| 8 | To byte |

## Record Buffer for Command Option 2 = Blank

Information for all fields of a file, but not for special descriptors (for example, superdescriptors), is returned in the record buffer, Setting the command option 2 to blank returns the information in the following format:

### Record Buffer General Layout

| Bytes | Usage |
|---|---|
| 1 - 4 | Number of fields in the FDT (excluding special descriptors) |
| 5 -N | Field definitions, each 6 bytes long |

### FDT Field Definitions

| Bytes | Usage |
|---|---|
| 1 | Level number (binary) |
| 2 - 3 | Field name |
| 4 | Length (0 for groups, 255 means length > 254) |
| 5 | Format |
| 6 | Definition options<br>(00 = no definition options)<br>0x 80 = unique descriptor<br>0x 40 = parent of sub-/superdescriptor<br>0x 20 = parent of phonetic descriptor<br>0x 10 = PE group field/ PE inner group/ elementary field within PE group<br>0x 08 = null suppression<br>0x 04 = multiple field<br>0x 02 = fixed format<br>0x 01 = descriptor |

**Note:** The definition options in the record buffer for command option 2 = blank are not mainframe-compatible. Mainframe-compatible LF commands are only possible with command option 2 = S.

## Record Buffer for Command Option 2 = ‘X’ and ‘F’

Setting command option 2 to ‘X’ or ‘F’returns the information in the following format:

### Record Buffer General Layout

| Bytes | Usage |
|---|---|
| 1 - 4 | Total length of information |
| 5 | Structure level (= 0) (with S option you get ‘F’ at this location) |
| 6 | Flag byte for future use |
| 7 - 8 | Number of entries in the FDT including Special Descriptor Table (SDT) and Referential Integrity Constraint Table (RIT)<br><br>The maximum number of entries in the FDT including SDT is 3214; additionally referential integri-ty constraints can be defined for a file. |
| 9 -16 | Unix timestamp (microseconds since 1970) |
| 17 - N | Field definitions, each 16 bytes long. Larger entries may be possible in future versions. |
| (N+1) - M | SDT (Special Descriptor Table). Special descriptors are:<br>■ Subdescriptors<br>■ Superdescriptors<br>■ Phonetic descriptors<br>■ Hyperdescriptors<br>■ Collation descriptors<br><br>The length of an SDT definition is a multiple of 4 bytes. |
| (M+1) - L | RIT (Referential Integrity Constraint Table) |

**Notes:**

- Each entry in the record buffer begins with the following fields:
  - Byte 1: Type of the entry
  - Byte 2: Length of the entry
  - Byte 3-4: Name of the entry
  - Byte 5: Format (Not for RI constraints)

  - Byte 6: Options (Not for RI constraints)
  - Descriptor length (Only for SDT entries)
- In order to be compatible with future Adabas versions, please consider the following:
  - Future Adabas versions may provide additional entry types for the ‘X’ option, therefore, you should not consider unknown entry types to be an error.
  - The record buffer entries may be larger than with the current Adabas version. Therefore, you should always use the entry length field to skip to the next entry. The information currently returned in the record buffer will also be returned at the same position in a record buffer entry with future Adabas versions.
  - The length of each entry is aligned to 4 bytes so that you can access 4-byte integer values without alignment problems.
- There may be more entries returned in the LF command than specified in the file definition, because the names of referential integrity constraints are defined only for the file containing the foreign key, but they are also re-turned by the LF command for the file containing the primary key.

## FDT Field Definitions

| Bytes | Usage |
|---|---|
| 1 | Indicator field name ‘F’ |
| 2 | Total length of ‘F’ entry |
| 3-4 | Field name |
| 5 | Format |
| 6 | Definition options<br>(00 = no definition options)<br>0x 01 = unique descriptor<br>0x 02 = parent of sub-/superdescriptor<br>0x 04 = parent of phonetic descriptor<br>0x 08 = PE group field/ PE inner group/<br>elementary field within PE group<br>0x 10 = null suppression<br>0x 20 = multiple field<br>0x 40 = fixed format<br>0x 80 = descriptor |
| 7 | 0x01 = NC option is active<br>0x02 = NN option is active<br>0x04 = LB option is active<br>0x08 = LA option is active<br>0x20 = HF option is active<br>0x40 = NV option is active<br>0x80 = NB option is active |
| 8 | Level number |

| Bytes | Usage |
|---|---|
| 9 | Date/time edit masks:<br>1 = E(DATE)<br>2 = E(TIME)<br>3 = E(DATETIME)<br>4 = E(TIMESTAMP)<br>5 = E(NATDATE)<br>6 = E(NATTIME)<br>7 = E(UNIXTIME)<br>8 = E(XTIMESTAMP) |
| 10 | Suboptions:<br><br>For DT option (indicated by byte 9 not equal to 0):<br>Bit 0x01 = TZ option is active<br><br>For desrciptor option (indicated by bit 0x80 in byte 6):<br>Bit 0x02 = TR option is active<br><br>For SY option (indicated by byte 11 not equal to 0):<br>Bit 0x40 = CR option is active |
| 11 | SY function:<br>1 = TIME<br>2 = SESSIONID<br>3 = OPUSER |
| 12 | Deactivation flag (only with command option 'F') :<br>Bit 0x01 = Field logically deleted |
| 13-16 | Field length |

## SDT Field Definitions

There are SDT field definitions for the same descriptor types and with the same first byte as for the 'S' option.

### Collation Descriptor Definitions

| Bytes | Usage |
|---|---|
| 1 | C indicates collation descriptor |
| 2 | Total length of C entry |
| 3-4 | Collation descriptor name |
| 5 | Format of parent field (On open systems only W, on mainframe also A is possible) |
| 6 | Definition options (same as byte 4 for 'S' option) |
| 7-8 | Standard length |
| 9-10 | Parent field name |
| 11-12 | Maximum internal length |

| Bytes | Usage |
|---|---|
| 13 | Additional options<br>0x04 = LA<br>0x08 = LB<br>0x80 = Collation defined via collation exit<br>Not set: collation defined via ICU<br>Other bits: unused |
| 14 | String length of collation attribute string (length byte and termination null character not included) |
| 15-14 + byte 14 | Collation attribute string as null terminated string, for example: "'de',PRIMARY"<br><br>If collation defined via collation exit: exit number as null terminated string, for example: "1" |

**Notes:**

1. The C entry length is aligned to 4 bytes.
2. The format of parent field is relevant if you specify a parent field value in the value buffer. If the collation descriptor is defined without the HE option, you can also specify the internal collation descriptor values. These values have the format A with the option NV.

### Hyperdescriptor Definitions

| Bytes | Usage |
|---|---|
| 1 | H indicates hyperdescriptor |
| 2 | Total length of H entry |
| 3-4 | Hyperdescriptor name |
| 5 | Format |
| 6 | Definition options (same as byte 4 for 'S' option) |
| 7-8 | Hyperdescriptor length |
| 9 | Exit number |
| 10 | Additional options (mainframe only, same as byte 8 for 'S' option) |
| 11 | unused |
| 12 | Number of parent fields |
| 13-12 + 2* byte 12 | Parent field names |

**Note:** The H entry length is aligned to 4 bytes.

## Phonetic Descriptor Definitions

| Bytes | Usage |
|---|---|
| 1 | P indicates phonetic descriptor |
| 2 | Total length of P entry |
| 3-4 | Phonetic descriptor name |
| 5 | Format (currently only A supported) |
| 6 | Options (unused) |
| 7-8 | Descriptor length |
| 9-10 | unused |
| 11-12 | Parent field name |

## Sub-/Superdescriptor Definitions

| Bytes | Usage |
|---|---|
| 1 | S indicates subdescriptor<br><br>T indicates superdescriptor |
| 2 | Total length of entry |
| 3-4 | Sub-/superdescriptor name |
| 5 | Format |
| 6 | Definition options (same as byte 4 for 'S' option) |
| 7-8 | Descriptor length |
| 9 | unused |
| 10 | Number of parent fields (1 for subdescriptor, > 1 for superdescriptor) |
| 11-10 + 6*byte 10 | Parent field entries – for each parent field:<br>■ Parent field name (2 bytes)<br>■ From byte (2 bytes)<br>■ To byte (2 bytes) |

**Note:** The sub-/superdescriptor entry length is aligned to 4 bytes.

### Referential Integrity Definitions

| Bytes | Usage |
|---|---|
| 1 | R indicates referential constraint |
| 2 | Total length of R entry |
| 3-4 | Constraint name |
| 5-8 | Reference file<br><br>In case of primary file entry, the file of the foreign key<br><br>In case of foreign file entry, the file of the primary key |
| 9-10 | Primary key name (00 means ISN) |
| 11-12 | Foreign key name |
| 13 | 1 primary file entry<br><br>2 foreign file entry |
| 14 | Update action:<br><br>0 No action<br><br>1 Cascade<br><br>2 Set NULL |
| 15 | Delete action:<br><br>0 No action<br><br>1 Cascade<br><br>2 Set NULL |
| 16 | Unused |

The FDTs of each of the two referenced files contain an entry for a referential constraint. If both the foreign and primary key are in the same file for a referential constraint, the FDT of this file contains two entries for this constraint.

**Note:** The name of a referential integrity constraint may also occur as the name of a field or special descriptor of the file.

## Example

The field definition information (including features introduced with Adabas version 6.2) for file 1 is to be read.

**Control Block:**

```
Command Code          LF
```

```
File Number           1 (field definitions for file 1 requested)
```

```
Record Buffer Length  100
```

```
Command Option 2      X
```

```
Additions 3           bbbbbbbb (file is not security protected)
```

# 20 MC Command (Multi-Call)

## Function and Use

The MC command is used to reduce the interprocess communications between an Adabas database and its application programs. It does not provide any additional functionality at the Adabas call interface. The MC command is a generalization of the multifetch behaviour for read commands: this generalization applies to any arbitrary sequence of Adabas commands with the exception of BT, CL, OP and MC.

**Note:** The MC command is only supported by the ACB interface, it is *NOT* supported by the ACBX interface.

Each Adabas command of such a sequence is represented by its control block and its command buffers, which are delivered in the corresponding buffers of the MC command. The control blocks are delivered at the beginning of the record buffer. The number of different Adabas commands must be specified in `cq_isn_quantity'.

An Adabas command sequence of an MC command can be terminated by an ET command. Only one ET command is allowed per MC command and this one must always be the last command of the sequence. If it is not the last command, the ET command will be rejected with response 22.

If a response code greater than 0 occurs during subcommand processing, the execution of further subcommands will be stopped immediately. However, an open transaction will not be backed out. The MC command returns response 16 and provides a pointer to the control block of the failed subcommand in `cb_addition_2'.

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN Quantity | B | F/U |
| Format Buffer Length (ACB only) | B | F/U |
| Record Buffer Length (ACB only) | B | F/U |
| Search Buffer Length (ACB only) | B | F/U |
| Value Buffer Length (ACB only) | B | F/U |
| ISN Buffer Length (ACB only) | B | F/U |
| Command Option 2 | A | F/U |
| Additions 2 | A,B | -/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | F/U |
| Record Buffer | F/A |
| Search Buffer | F/U |
| Value Buffer | F/U |
| ISN Buffer | F/A |

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
MC

**Response Code**
Adabas returns the response code of the MC command in this field. If a subcommand terminates with a non-zero response code, Adabas returns the response code 16 in this field and stops further processing of subcommands.

**ISN Quantity**
Number of subcommands delivered by the MC command. Value 0 means nothing to do.

**Format Buffer Length (ACB only)**
If there is no subcommand referring to a format buffer, this length can be set to 0.

**Record Buffer Length (ACB only)**
The length of the record buffer must be specified in this field. If ISN Quantity is not 0, this length must be set. The length specified must be large enough to accommodate all required record buffer entries and control blocks.

**Search Buffer Length (ACB only)**
If there is no subcommand referring to a search buffer, this length can be set to 0.

**Value Buffer Length (ACB only)**
If there is no subcommand referring to a value buffer, this length can be set to 0.

**ISN Buffer Length (ACB only)**
If there is no subcommand referring to an ISN buffer, this length can be set to 0.

**Command Option 2**
An `S' (SHORT) in this field indicates that only the control block of the MC command is returned when no error occurs during subcommand processing.

**Additions 2**
If response code 16 is returned, the following information will be returned in this field:

- The first two bytes will contain the sequence number of the subcall which has received a non-zero response code;
- The third and fourth bytes will contain the byte offset of the control block in the record buffer that caused the response code.

## Format Buffer

The format buffer contains a buffer header (see *Processing Considerations, Command Buffers*) and the format buffer(s) of the subcommands. It may be omitted if no subcommand references the format buffer.

## Record Buffer

The record buffer contains a buffer header followed immediately by the control blocks of all the subcommands followed by the record buffers of the subcommands.

## Search Buffer

The search buffer contains a buffer header (see *Processing Considerations, Command Buffers*) and the search buffer(s) of the subcommands. It may be omitted if no subcommand references the search buffer.

## Value Buffer

The value buffer contains a buffer header (see *Processing Considerations, Command Buffers*) and the value buffer(s) of the subcommands. It may be omitted if no subcommand references the value buffer.

## ISN Buffer

The ISN buffer contains a buffer header (see *Processing Considerations, Command Buffers*) and the ISN buffer(s) of the subcommands. It may be omitted if no subcommand references the ISN buffer.

## Processing Considerations

### Command Buffers

The command buffers of the several subcommands will be delivered within the corresponding command buffers of the MC command. Therefore, within a buffer, several entries can be present. Each buffer must be started (offset 0) by a two-element structure. This structure contains two two-byte offsets and is called the buffer header. The buffer header indicates what is within the buffer. The buffer can contain several command buffers or only one global buffer which is used by all the subcommands. All specified offsets are relative to the beginning of the corresponding command buffer.

The C definition for the buffer header is:

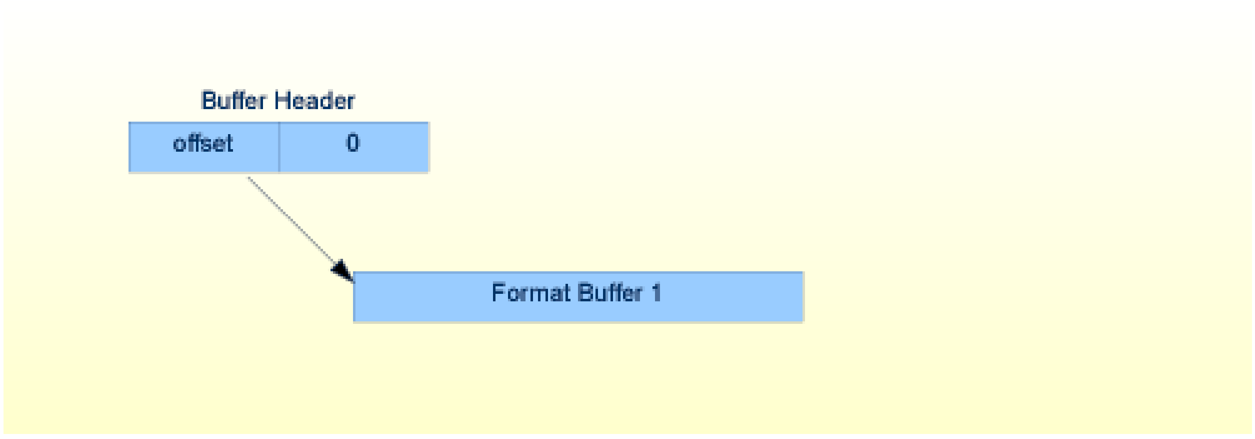
```
struct CmdBufHd
{
    unsigned short  GloBuf;     /* Offset to global buffer used by all subcommands */
     unsigned short  StartOff;   /* Offset to an array of buffer start
                                   offsets */

}
```

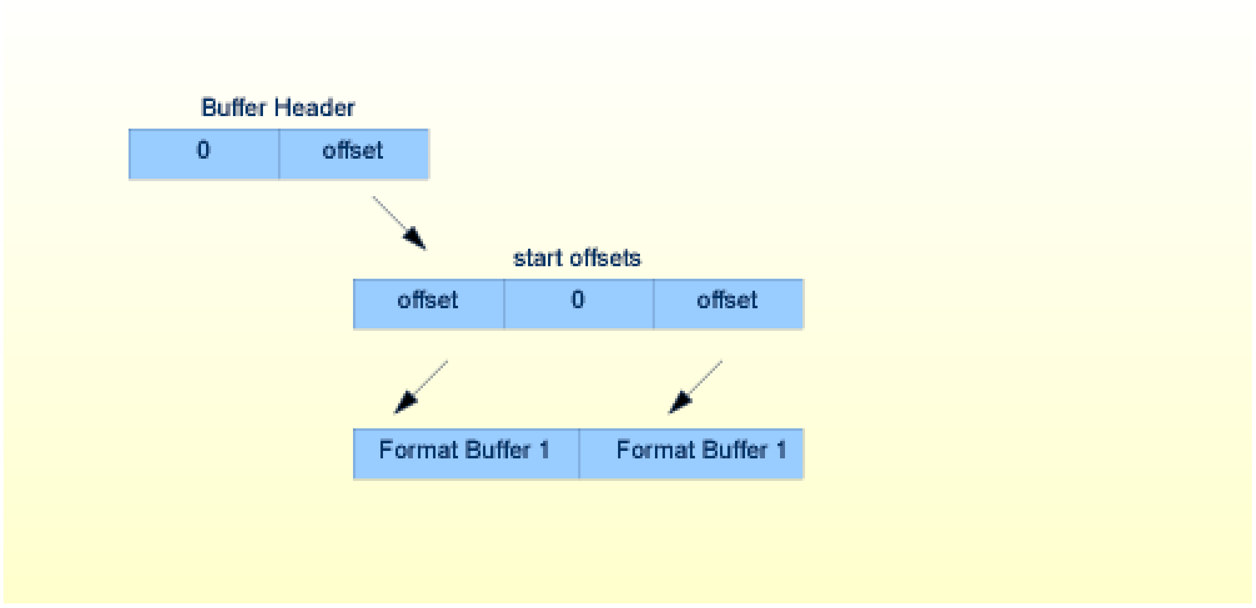
If both offsets of a buffer header are set, the corresponding MC command is rejected with response 146.

**Example:**

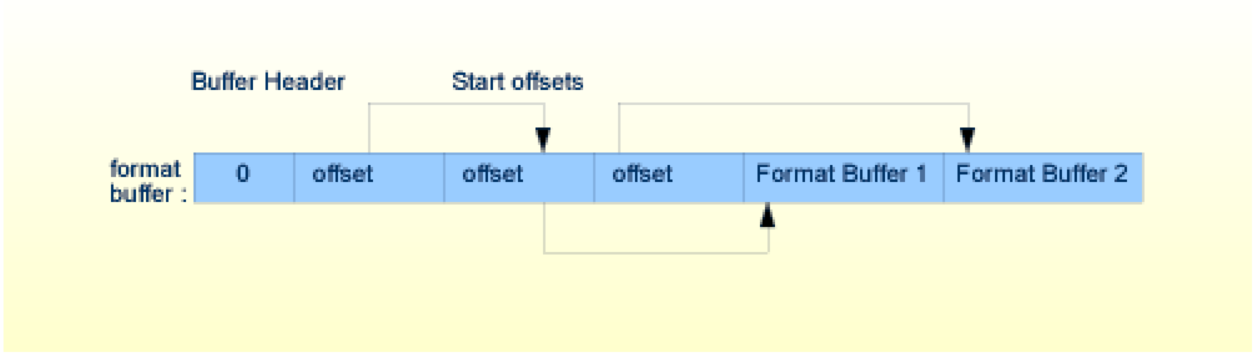
If the format buffer of an MC command contains a common format buffer for all subcommands, the `GloBuf' offset of the buffer header must point to the common format buffer and the `StartOff' offset must be 0.

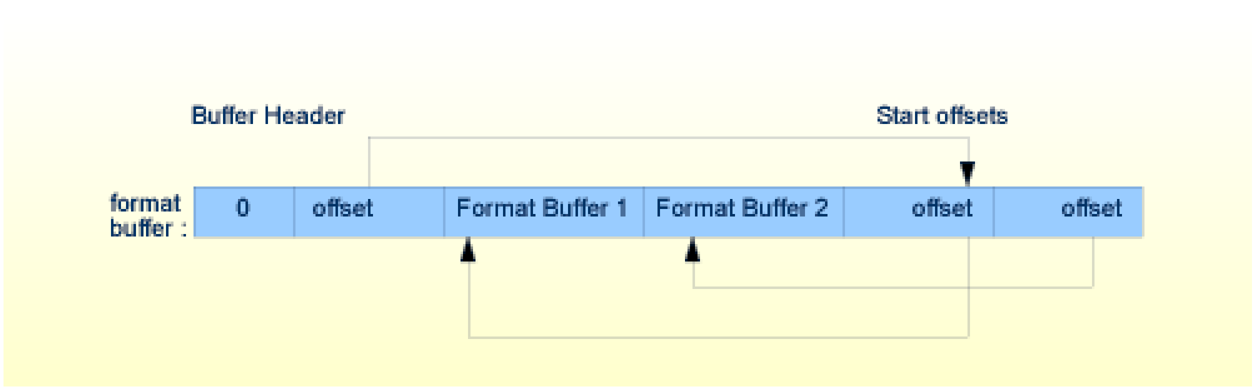


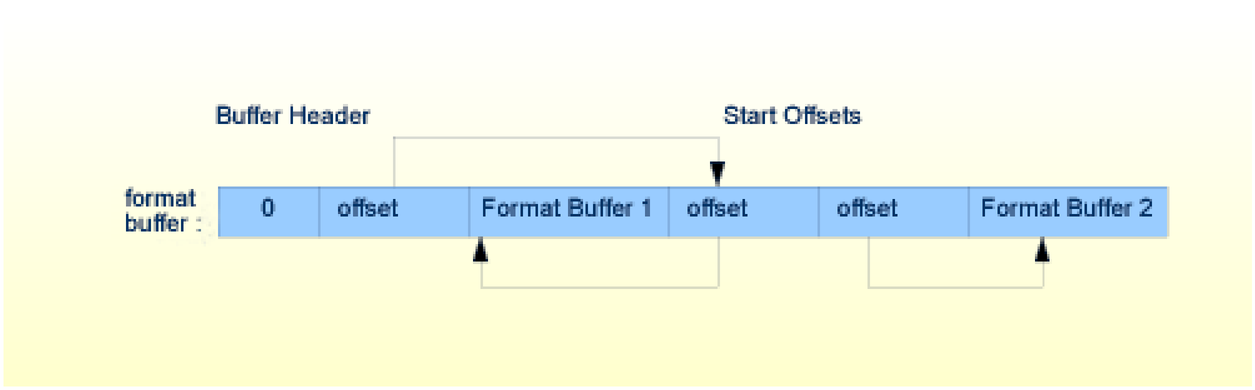
However, the buffer may contain several different format buffers. In this case, the buffer header has an offset to a second structure, which contains the start offsets to the different format buffers. The start offsets are 2–byte fields with the subcommand sequence number as index. A zero start offset indicates that the corresponding subcommand does not refer to that command buffer.

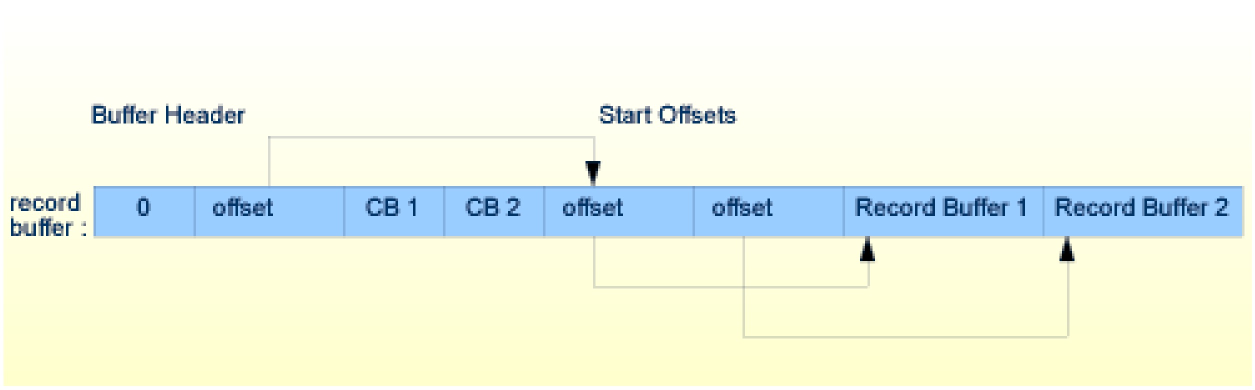


The array of the 2-byte start offsets is position-independent, but the whole array must be delivered contiguously. The only exception is the record buffer. The control blocks of the different subcommands must always be delivered directly behind the `Buffer Header' of the record buffer (offset=4).









If a subcommand does not use a command buffer, the corresponding offset will not be checked. A global input buffer will be ignored too if the buffer is not referred to by the command.

The control block and the ISN buffer of the subcommand must always start at an offset divisible by 4 within the corresponding MC command buffer. Because the subcommand control blocks must start directly behind the `buffer header', the first control block always starts at offset 4 and therefore all following control blocks start automatically with a correct offset, because the control–block length is always divisible by 4. Within the ISN buffer, the user must ensure that all offsets are divisible by 4. If not, the MC command is terminated by response 146 and no subcommands are processed.

If no subcommand of the sequences refers to a command buffer, the corresponding buffer of the MC command is not necessary and the buffer size can be set to 0.

It is also possible to specify the same start offset to a command buffer from different subcommands, but it will not be permitted to use this feature to transfer data from one subcommand to a subsequent one. There is no check for record–buffer or ISN–buffer consistency. The caller is responsible for the correct specification of the subcommand output buffers to avoid overwriting.

## Response Handling

If an error occurs, the current subcommand will be backed out, if necessary, and the execution of further subcommands will be stopped immediately. This will be done for all response codes greater than 0.

The errors will be signalled in the following way:

Errors which occur within the MC environment are returned in the control block of the MC command. `cb_add2' does not point to a control block of a subcommand. This kind of error occurs during start or end processing of the MC command (before the first subcommand or after the last subcommand is processed).

Errors which occur during subcommand processing are returned in their control blocks. The MC command itself will return response 16, to signal that a subcommand has failed. Additions 2 points to the control block of the failed subcommand.

Response table of an MC command:

| **RSP** | **ADD_2 (first two bytes)** | **ADD_2 (last two bytes)** | **Description** |
|---|---|---|---|
| 16 | Number of subcommand | Offset to subcommand's control block | Error during subcommand processing. Error code and additional information are delivered in the control block of the failed subcommand. |
| 146 | Number of command buffer | Command buffer name | Incorrect MC command buffer detected (illegal buffer header, start offsets out of range, ISN buffer offset not divisible by 4,...) |
| Response of MC command | additional information if present | | MC command was terminated during command start or command end processing. |

## Response 146

Explanation: This response is caused by one of the following:

- illegal buffer header detected (both offsets are set)
- an ISN-buffer offset is not divisible by 4
- `start offset array' out of MC buffer range
- record buffer of MC command not big enough to contain all subcommand control blocks
- invalid buffer length detected (subcommand)

The `cb_add2' field in the control block contains additional information for response 146:

| | first two bytes | last two bytes |
|---|---|---|
| Format Buffer | 1 | FB |
| Record Buffer | 2 | RB |
| Search Buffer | 3 | SB |
| Value Buffer | 4 | VB |
| ISN Buffer | 5 | IB |

**Note:** If an Adabas command with Prefetch/Multifetch option terminates with a response during a subsequent ISN processing, this response is not returned within the corresponding control block and therefore the MC command does not terminate with response 16. Such a subcommand is handled like a subcommand which is successfully terminated.

If an MC command is terminated with response 146, no subcommand has been processed.

Response table of a subcommand:

| RSP | ADD_2<br>(first two bytes) | ADD_2<br>(last two bytes) | Description |
|---|---|---|---|
| 146 | Number of command buffer | Command buffer name | Invalid buffer length detected, e.g. the end of subcommand buffer (offset + buffer length) is out of the corresponding MC command buffer range. |
| 22 | - | 22<br>23 | Invalid command or command which is not allowed within an MC command sequence.<br>ET command which is not the last command of an MC subcommand sequence. |
| Response of command | additional information if present | | Subcommand processing was terminated by response code `rsp'. |

# 21 N1/N2 Command (Add Record)

## Function and Use

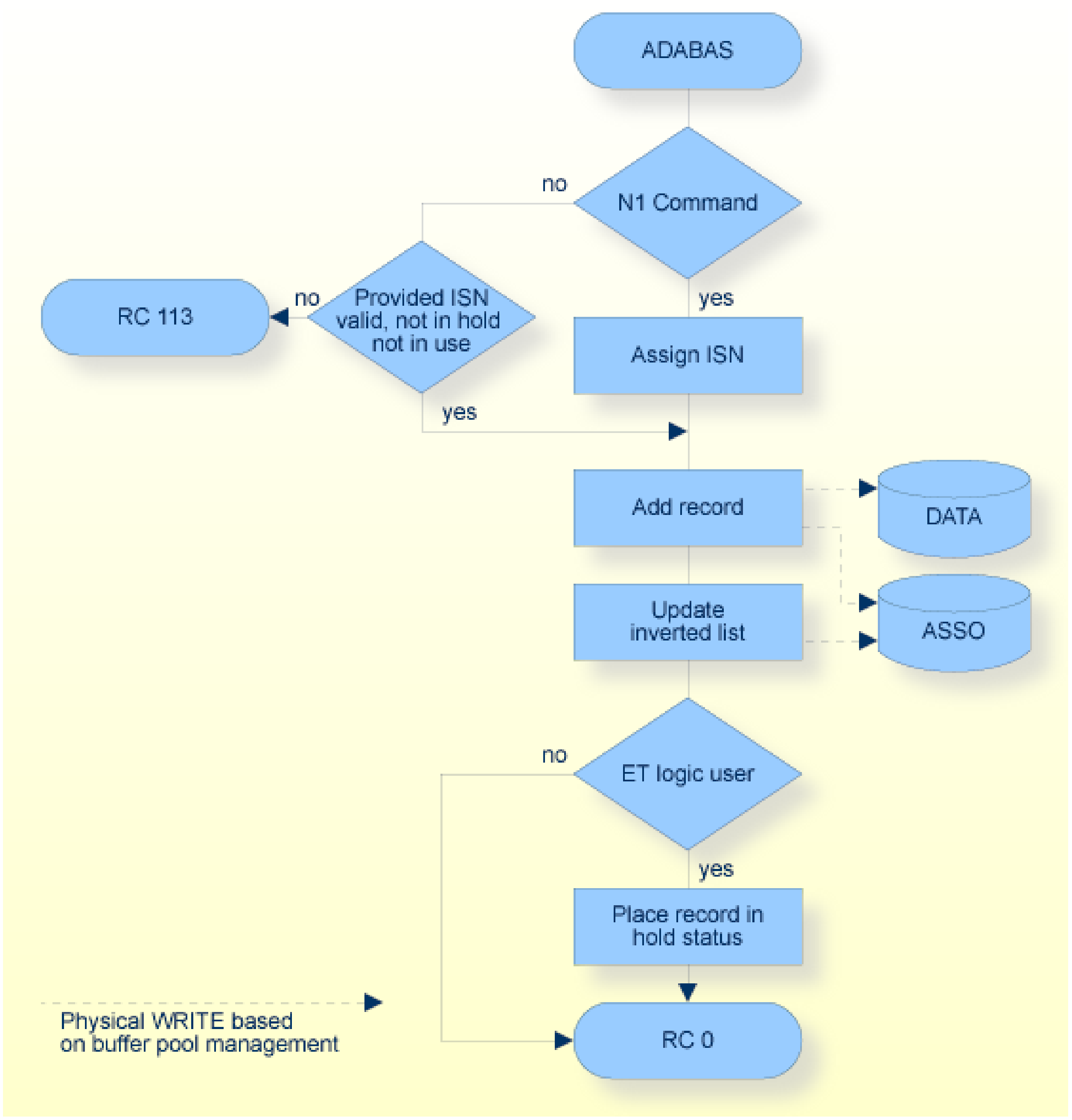
The N1 command is used to add a new record to a file.

The user specifies the file to which the record is to be added and the fields for which values are being provided. Any fields not specified will contain a null value in the record added.

Adabas assigns the record an ISN, adds the record to Data Storage and performs any Associator updating which may be required.

The N2 command is used if the ISN to be assigned to the record is provided by the user, regardless of whether REUSE=ISN is specified in ADAFDU.

If the user is an ET logic user, the record added is placed in exclusive hold status.


ADABAS
N1 Command
no
yes
Provided ISN valid, not in hold not in use
no
RC 113
yes
Assign ISN
Add record
DATA
Update inverted list
ASSO
ET logic user
no
yes
Place record in hold status
RC 0
Physical WRITE based on buffer pool management

**N1/N2 Command, Procedure Flow**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN | B | * |
| Format Buffer Length (ACB only) | B | F/U |
| Record Buffer Length (ACB only) | B | F/U |
| Additions 2 | A,B | -/A |
| Additions 3 | A | F/A |
| Additions 5 | A | F/U |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | F/U |
| Record Buffer | F/U |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | –/– |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |
| * | –/A for N1; F/U for N2 |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
N1/N2

**Command ID**
If a series of records is being added using a series of N1/N2 calls and the same fields are specified in the format buffer for each call, this field should be set to a non-blank, non-zero value. This results in a reduction in the time required to process each N1/N2 call.

If only a single record is being added, or if the format buffer is modified between N1/N2 calls, this field should be set to blanks or binary zeros.

The high-order byte of this field must not be set to hexadecimal `FF', except when automatic command ID generation is used (see Programming Considerations, Command IDs for additional information).

**File Number**
The number of the file to which the record is to be added.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**ISN**
If the N1 command is being used, Adabas returns the ISN assigned to the record in this field.

If the N2 command is being used, the ISN to be assigned to the record must be provided in this field. The ISN provided must not already be assigned to a record in the file and must be within the limit (MAXISN) in effect for the file. MAXISN is set by the DBA when the file is loaded.

**Format Buffer Length (ACB only)**
The format buffer length (in bytes). The format buffer area defined in the user program must be as large as (or larger than) the length specified.

**Record Buffer Length (ACB only)**
The record buffer length (in bytes). The record buffer area defined in the user program must be as large as (or larger than) the length specified.

**Additions 2**
If response code 0 is returned, Adabas returns the compressed record length of the record added in this field. The length is provided in the first two bytes in binary format.

For some response codes, Adabas returns detailed information in this field. See *Adabas Messages And Codes* for further information.

**Additions 3**

This field is used to provide a security password.

If the file to be used is not security protected, this field should be set to blanks. If the file is security protected, the user must provide a valid password.

Adabas sets this field to blanks during command processing to protect the integrity of any password provided.

**Additions 5**

This field may be used to provide a separate format buffer ID that is used to identify the internal format buffer used for this command, or to provide a global format buffer ID.

As long as the first byte of the Additions 5 field is not alphanumeric, the value provided in the command ID field will also be used as the format buffer ID.

If the first byte is a lower case character, the bytes 5 to 8 of the Additions 5 field will be used as the separate local format buffer ID.

If the first byte is a digit or an upper case character, the Additions 5 field (8 bytes) will be used as a separate global format buffer ID, which means that the format buffer ID can be used by several users in parallel.

See *Programming Considerations, Using Command IDs* for additional information and examples.

## Format Buffer

The fields for which values are being provided in the record buffer must be specified in this buffer.

The syntax and examples of format buffer construction are provided in *Calling Adabas, Format and Record Buffers*.

Any fields which are not specified will contain a null value in the record being added.

All non-NU descriptors which are not specified in the format buffer will have null values in the inverted list.

For non-NU descriptors which are contained in a periodic group, null values will be entered in the inverted list only for null occurrences which precede the highest occurrence number specified in the format buffer.

## Record Buffer

The value for each field specified in the format buffer must be provided in this buffer.

Each value must be provided according to the standard length and format of the field for which the value is being provided, unless a different length and/or format is specified in the format buffer.

If the field is defined as a variable length field (no standard length), a one byte binary field containing the length of the field (including the length byte) must be provided immediately before the value.

If the field for which the value is being provided is defined as a unique descriptor, the value provided must not already exist for the descriptor; otherwise, error response 98 will be returned.

## Additional Considerations

The following additional considerations are applicable for the N1/N2 command:

1. Subdescriptors, superdescriptors, and phonetic descriptors may not be specified in the format buffer. Adabas automatically creates the correct value for any of the above if a field, from which a subdescriptor, superdescriptor, or phonetic descriptor is derived, is specified in the format buffer.
2. The maximum record length after compression (including record ISN) is the maximum available DATA storage block size - 4.
3. A descriptor value may not exceed 1144 bytes, unless the descriptor is defined with the TR option.
4. If a field is specified using a length override which exceeds the standard length (not permitted if the field is defined with the fixed storage option), all subsequent references to this field should specify the length which was used. If a subsequent reference uses the standard length, value truncation for alphanumeric fields (if OPTIONS=TRUNCATION is specified) or a non–zero response code for numeric fields may occur.
5. Field names withput index may be specified more than once only if they are multiple-value fields. Multiple-value fields and fields in periodic groups with the same index may not be specified more than once. It is also forbidden to specify a group and a field in the group at the same time.
6. Numeric edit masks must not be specified in the format buffer.
7. A multiple-value count field or periodic-group count field specified in the format buffer will be ignored by Adabas. The corresponding value in the record buffer will also be ignored. A literal in the format buffer will be ignored by Adabas. The corresponding positions in the record buffer will also be ignored.

8. If a multiple-value field is specified in the format buffer, Adabas sets the multiple-value field count according to the following rules:

   - For a multiple-value field defined with the NU option, the count field is adjusted to reflect the number of existing non–blank values. Blank values are completely suppressed.

| | |
|---|---|
| Field Definition<br>Format Buffer<br>Record Buffer<br>Result after add | 01,MF,5,A,MU,NU<br>MF1-3<br>XXXXXYYYYYZZZZZ<br>XXXXX,YYYYY,ZZZZZ<br>MF count = 3 |
| Format Buffer<br>Record Buffer<br>Result after add | MF1-3<br>XXXXXbbbbbZZZZZ<br>XXXXX,ZZZZZ<br>MF count = 2 |
| Format Buffer<br>Record Buffer<br>Result after add | MF1-3<br>bbbbbbbbbbbbbbb (blanks)<br>Values suppressed<br>MF count = 0 |

   - For a multiple–value field defined without the NU option, the count is adjusted to reflect the number of existing values (including null values).

| | |
|---|---|
| Field Definition<br>Format Buffer<br>Record Buffer<br>Result after add | 01,MF,5,A,MU<br>MF1-3<br>XXXXXYYYYYbbbbb<br>XXXXX,YYYYY,b(blank)<br>MF count = 3 |
| Format Buffer<br>Record Buffer<br>Result after add | MF1<br>bbbbb (blanks)<br>b (blank)<br>MF count = 1 |

9. If a periodic group or a field within a periodic group is specified in the format buffer, Adabas sets the periodic group count equal to the highest occurrence number specified in the format buffer. If the highest occurrence is null–value suppressed, the count is adjusted accordingly.

| | |
|---|---|
| Field Definitions | 01,GB,PE<br>02,BA,1,B,DE,NU<br>02,BB,5,P,NU |
| Format Buffer<br>Record Buffer | GB1-2.<br>0x08000000500C09000000600C<br>(or ^X08000000500C09000000600C) |

| | |
|---|---|
| Result after add | GB (1st occurrence)<br>BA = 8 BB = 500<br>GB (2nd occurrence)<br>BA = 9 BB = 600<br>GB count = 2 |
| Format Buffer<br>Record Buffer | GB1-2.<br>0x00000000000C00000000000C<br>(or ^X00000000000C00000000000C) |
| Result after add | GB (1st occurrence)<br>Values Suppressed<br>GB (2nd occurrence)<br>Values Suppressed<br>GB count = 0 |

10. If a field defined with variable length (no standard length), as specified in the format buffer, the corresponding value in the record buffer must be preceded by a one byte binary number which represents the length of the value (including the length byte).

| | |
|---|---|
| Field Definitions | 01,AA,3,A<br>01,AB,0,A |
| Format Buffer<br>Record Buffer | AA,AB.<br>0x313233063132333435<br>(or ^X3132330631323334350) |

Fields AA and AB are to be added. The value for AA is 123. the value for AB (which is a variable length field) is 12345.

## Examples

### Example 1:

A record is to be added to file 1. The ISN of the record is to be assigned by Adabas. The field values which are to be provided are:

| FIELD | VALUE |
|---|---|
| AA | ABCD |
| MF (value 1) | AAA |
| MF (value 2) | BBB |
| BA (1st occurrence) | 5 |
| BA (2nd occurrence) | 6 |

**Control Block:**

| | |
|---|---|
| Command Code | N1 |
| Command ID | bbbb (only 1 record being added) |
| File Number | 1 |
| Format Buffer Length | 15 (or larger) |
| Record Buffer Length | 16 (or larger) |
| Additions 3 | bbbbbbbb (file 1 not security protected) |

**Buffer Areas:**

| | |
|---|---|
| Format Buffer | AA,MF1-2,BA1-2. |
| Record Buffer | 0x61626364202020206161616262620506<br>(or ^X61626364202020206161616262620506) |

**Example 2:**

A record is to be added to file 2. The ISN of the record is to be provided by the user. The field values to be provided are:

| FIELD | VALUE |
|---|---|
| RA | 12345678 |
| RB | ABCD |

**Control Block:**

| | |
|---|---|
| Command Code | N2 |
| Command ID | bbbb (only 1 record is to be added) |
| File Number | 2 |
| ISN | 20 (ISN 20 is to be assigned to the record) |
| Format Buffer Length | 6 (or larger) |
| Record Buffer Length | 18 (or larger) |
| Additions 3 | Password (file 2 is security protected) |

**Buffer Areas:**

| | |
|---|---|
| Format Buffer | RA,RB. |
| Record Buffer | 0x313233343536373861626364202020202020 (or ^X313233343536373861626364202020202020) |

# 22 OP Command (Open User Session)

## Function and Use

The OP command is used to indicate the beginning of a user session. If the user is not yet known to Adabas, a new user queue element will be allocated, which will exist until the user issues a CL command, or is stopped by Adabas or by an operator.

An OP command is mandatory if any of the following applies:

- The user is an exclusive control user (see the description in Concepts and Facilities);
- User data stored in an Adabas system file with an ET command is to be read;
- The user is to be an access-only user (no update commands permitted);

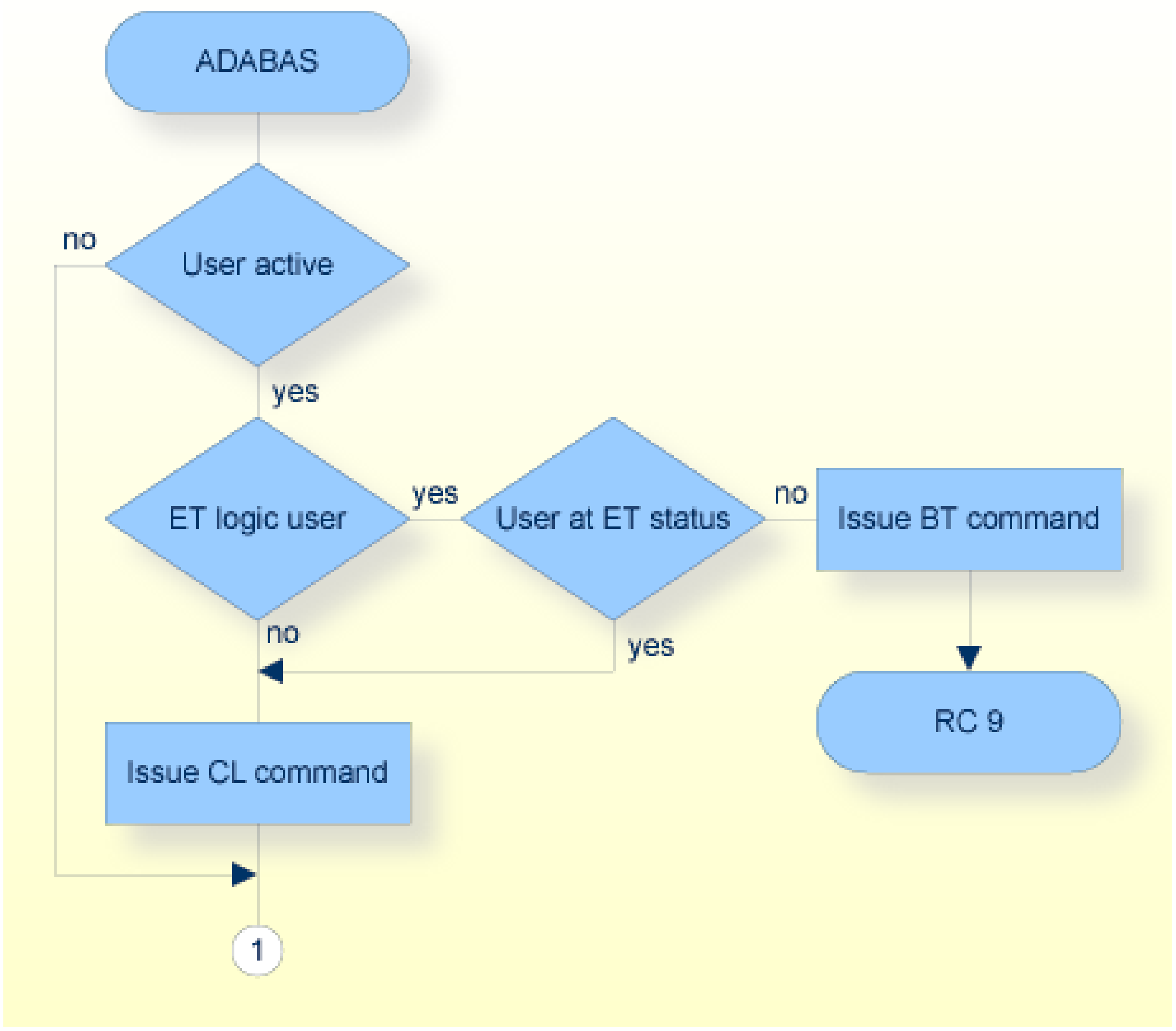
An OP command is optional for all other users if OPTIONS=OPEN_REQUIRED is not enabled for the current nucleus session. An implicit OP command will be issued by Adabas when the first Adabas command is issued by a user who is not currently identified to Adabas.

Users who use files which are security protected are not required to issue an OP command, but such users must provide a password with each command that involves a file which is security protected.

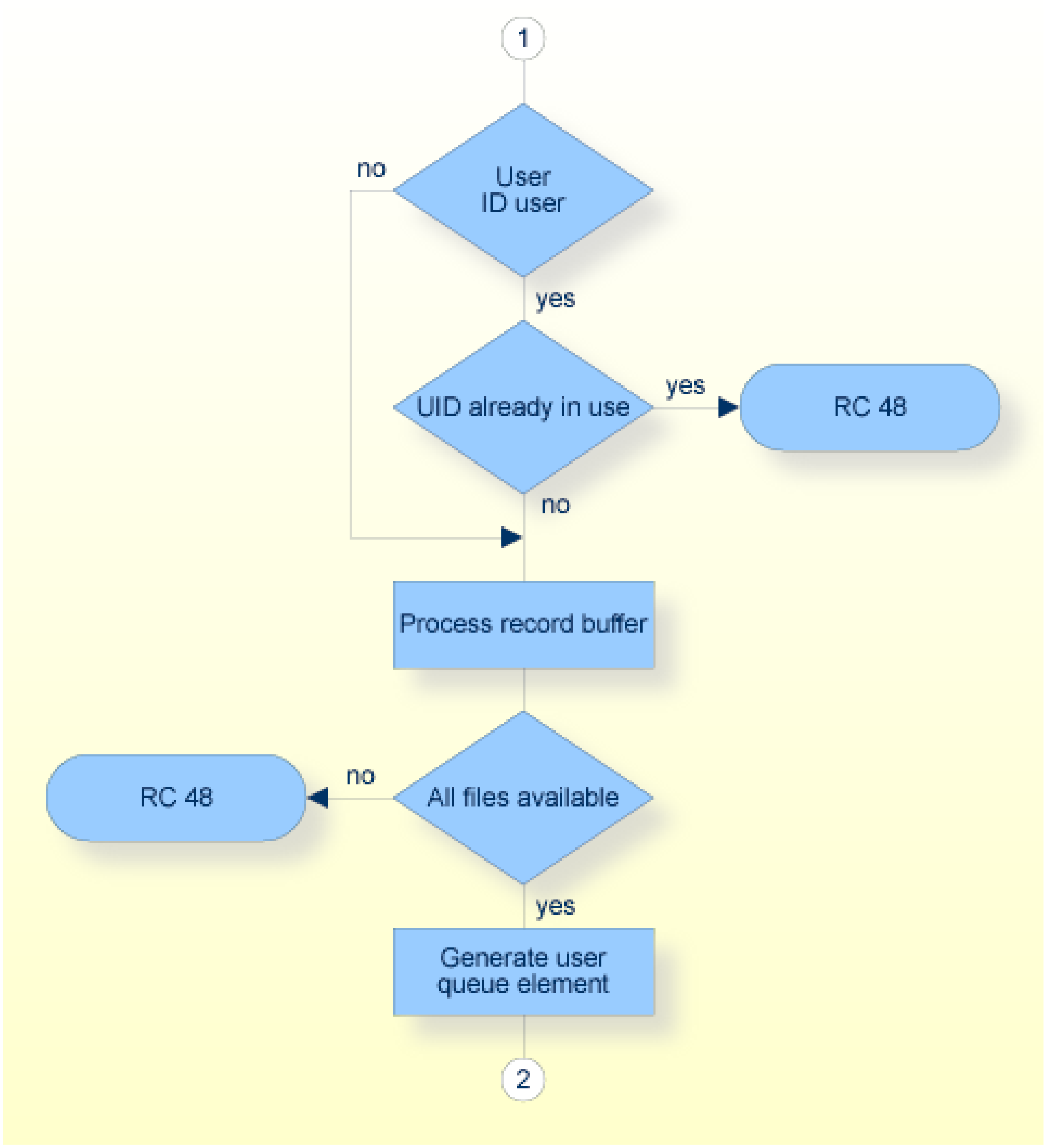
If an OP command is issued by an active ET logic user, and the user is not at ET status (currently has a record in hold), Adabas will issue a BT command for the user and will return response code 9 for the OP command. If an OP command is issued by any other type of active user, Adabas issues a CL command for the user prior to processing the OP command.

## User Types

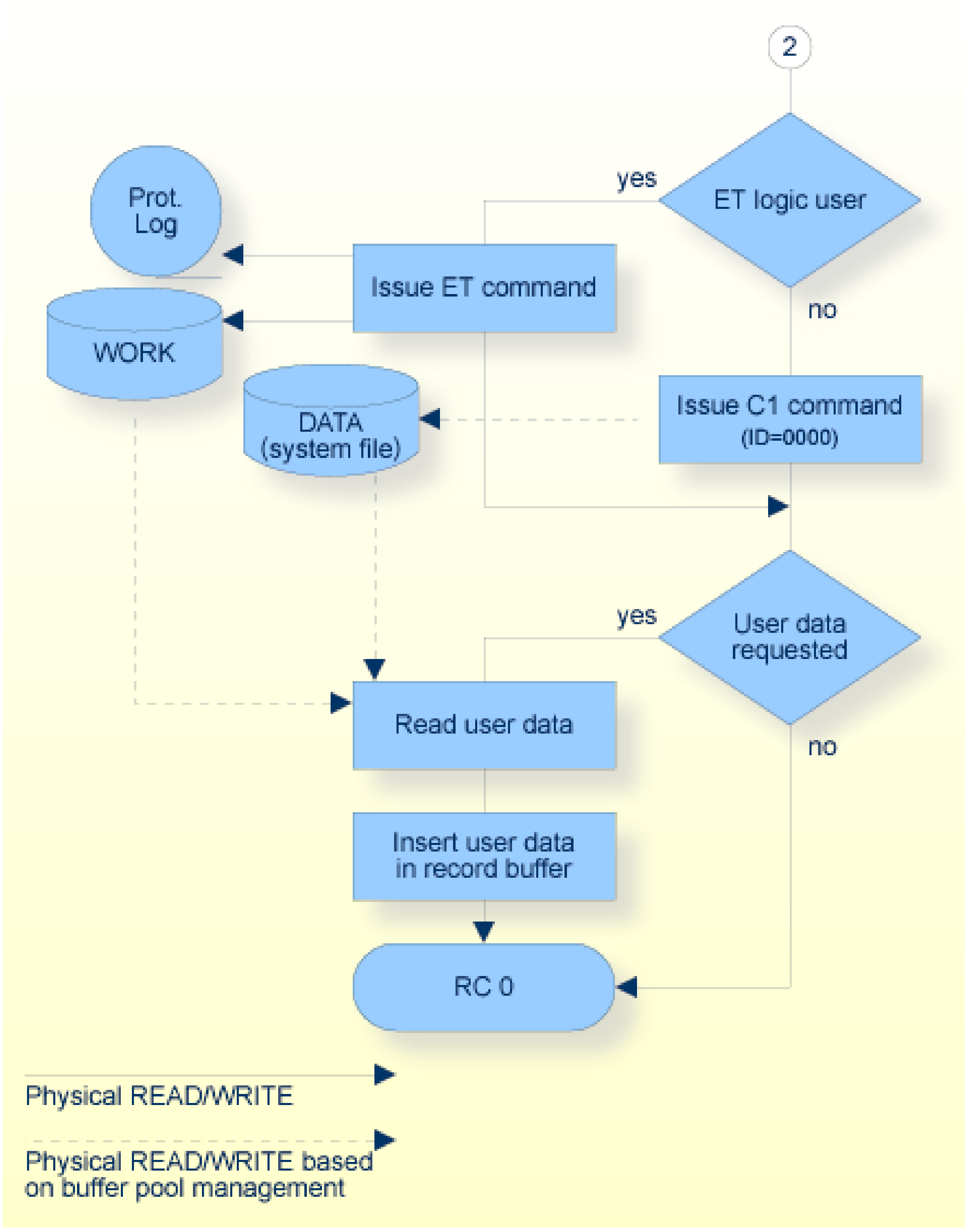
Adabas recognizes various user types, depending on the type of access/update performed by the user. See *Concepts and Facilities, User Types* for further details.



**OP Command, Procedure Flow**



**OP Command, Procedure Flow (continued)**



**OP Command, Procedure Flow (continued)**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | -/A |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN Lower Limit | B | F/A |
| ISN Quantity | B | F/A |
| Record Buffer Length (ACB only) | B | F/U |
| Command Option 1 | A | F/U |
| Command Option 2 | A | F/U |
| Additions 1 | A | F/U |
| Additions 2 | A,B | -/A |
| Additions 5 | A | -/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | */– |
| Record Buffer | F/U |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | –/– |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |

- - Not used
- * Not used but must be included in parameter list of CALL statement

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
OP

**Command ID**
For an access only user, Adabas always returns 0 in this field.

For an ET logic user, Adabas will return binary zeros in this field if the previous session for this user was successfully terminated with a CL command, or no previous session existed for this user. If this user is an ID user (Additions 1 is not blank) and the previous session of the current user ID was not successfully terminated with a CL command, Adabas will return the transaction sequence number of the last successfully-completed user transaction in this field. Additionally, response code 9 is returned.

**ISN Lower Limit**
This field may be used to provide a user-specific non-activity time limit.

If this field contains binary zeros, the non-activity time limit specified by the appropriate ADANUC parameter (TNAA, TNAE, or TNAX) for the Adabas session is in effect. Following successful OP completion, Adabas returns system release information in this field; the timeout information previously held here is instead returned in the Additions 5 field.

The following platform information is returned in the ISN Lower Limit field, considered as a 4-byte binary value:

| Byte | Contents |
|---|---|
| high order byte | Architecture byte describing the hardware architecture of the database:<br><br>Bit 0x01 set: low-order first<br>Bit 0x01 not set: high-oder first<br><br>Bit 0x04 set: EBCDIC<br>Bit 0x04 not set: ASCII<br><br>Bit 0x20 set: IEEE floating point<br>Bit 0x20 not set: Mainframe floating point<br><br>This implies: |

| Byte | Contents |
|---|---|
| | The architecture byte for Intel architectures (Windows, Linux) is 0x21.<br>The architecture byte for UNIX (AIX, HPUX, Solaris) zLINUX is 0x20.<br>The architecture byte for mainframe Adabas is 0x04. |
| | product line:<br>2 = Open Systems |
| | 0 |
| low order byte | local/remote indicator (local nucleus = 0, remote nucleus = 1) |

**ISN Quantity**

This field may be used to provide a user-specific transaction time limit.

If this field contains binary zeros, the non-activity time limit specified by the ADANUC parameter TT for the Adabas session is in effect. Following successful OP completion, Adabas returns system release information in this field; the timeout information previously held here is instead returned in the Additions 5 field.

The following version information is returned in the ISN Quantity field, considered as a 4-byte binary value:

| Byte | Contents |
|---|---|
| high order byte | major version |
| | minor version |
| | SM level |
| low order byte | patch level |

**Response Code**

Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**Record Buffer Length (ACB only)**

The length of the record buffer must be specified in this field. The length specified must be large enough to accommodate all required record buffer entries. The record buffer length should be set to zero if an empty record buffer is to be supplied.

If user data which is stored in an Adabas system file is to be returned, the length specified must be large enough to permit the user data to be inserted in the record buffer; otherwise, the user data will be truncated.

**Command Option 1**

An `R' in this field indicates that the user is to be restricted to only those files specified in the file list provided in the record buffer. Response code 17 will be returned if the user attempts to access/update a file that is not contained in the file list. If an R is not specified in this field, a request to access/update a file that is not in the file list is processed as described under User Queue Element in this section.

An `S' in this field enables the user section for subtransactions; this option implies the `R' option if a file list has been specified.

**Command Option 2**

An `E' in this field indicates that user data stored in an Adabas system file by a CL or ET command is to be returned in the record buffer.

The data stored with the last successful CL or ET command issued by the user in which user data was provided is returned.

This option may only be used if the user has provided the same USERID for both this user session and the session during which the user data was stored.

This field must be set to blank in all other cases.

**Additions 1**

This field may be used to provide a USERID for the user session.

A USERID must be provided if the user intends to store and/or read user data, and the user wants this data to be available during a subsequent user– or Adabas session.

- The user intends to store and/or read user data, and the user wants this data to be available during a subsequent user- or Adabas session;
- The user is to be assigned a special processing priority;

The value provided for the USERID must be unique for this user (not used by any other user), and must begin with a digit or an uppercase letter.

Users for whom none of the above conditions are true should set this field to blanks.

**Additions 2**

If the command is successful, the value returned in this field by Adabas depends on the user type:

| | |
|---|---|
| Access only user: | The input value is unchanged. |
| ET logic user: | The value returned is the ISN of the user's last successfully completed transaction for which user data was stored. If this is the first session or a non–ID user session, the value 0 is returned. |

For some response codes, Adabas returns detailed information in this field. See *Adabas Messages And Codes* for further information.

**Additions 5**

User-specific timeout values as specified in the fields ISN lower limit and ISN quantity are returned in the last 4 bytes of this field. The two high-order bytes contain the user-specific non-activity time limit in binary format, the two low-order bytes contain the user-specific transaction time limit in binary format.

## Record Buffer

The default encoding for W fields, the local time zone, and the files to be accessed and/or updated and the type of updating to be performed are specified in this buffer.

The syntax of this buffer is:

```
[OP_expression,...].
```

OP_expression may be one of the following:

| OP_expression | Explanation |
|---|---|
| TZ='timezone' | You must specify as *timezone* the name of a time zone that is contained in the tz database that is also known as the Olson database (see **https://www.iana.org/time-zones**). You must specify the full name of a location, for example:<br>*America/New_York* |
| WCHARSET='char_set' | You must specify as *char_set* an encoding name that is listed in **http://www.iana.org/assignments/character-sets** - most of the character sets listed there,are supported by ICU, which is used by Adabas for internationalization support. This character set is the default character set used for W fields in record and value buffers in the Adabas user session when no other character set has been specified in the format or search buffer.<br>Some character sets are platform-dependent, for example UTF-16. It is strongly recommended that you use the corresponding platform-independent character set, i.e. UTF-16BE or UTF-16LE instead of UTF-16. Otherwise Adabas cannot guarantee which variant is used. |
| usage=file_list | Here *usage* is one of the following:<br>■ ACC or ACCESS: files are to be accessed only.<br>■ UPD or UPDATE: files are to be updated (implies ET logic). Specifying 'UPD' or '.' makes the user an ET logic user.<br>■ EXF: files are to be opened under exclusive control of the user. No other user will be permitted to access the file while this user session is active. Exclusive control will be given only if no other active user has issued an OP command in which the ACC, EXF, EXU or UPD parameter was specified for the file.<br>■ EXU: file is to be updated under exclusive control of the user. No other user will be permitted to update the file while this user session is active. Exclusive control will be given only if no other active user has issued an OP command in which the EXF, EXU or UPD parameter was specified for the file.<br>The syntax of file_list is:<br>`filenumber [,filenumber] ...` |

| OP_expression | Explanation |
|---|---|
| | where *filenumber* is a number (leading zeros permitted) which indicates the Adabas file for which the preceding keyword is applicable. |

TZ and WCHARSET may be specified only once.

Duplicate file numbers for a given usage keyword are permitted. Duplicate file numbers across usage keywords are permitted. Each usage keyword may appear only once.

UPD, EXF and EXU also imply access to the file. If ACC or UPD are the only keywords specified, a file list is not required.

The following keyword combinations are permitted:

- ACC + EXF
- ACC + EXU
- ACC + UPD
- EXF + UPD
- EXU + UPD
- ACC + EXF + UPD
- ACC + EXU + UPD

A user may or may not be permitted to use a file currently in use by another user or by an Adabas utility (denoted by UTI) as shown below:

| USER 2 Requested Usage | USER 1 Current Usage | | | | |
|---|---|---|---|---|---|
| | ACC | EXF | EXU | UPD | UTI |
| ACC | yes | no | yes | yes | no |
| EXF | no | no | no | no | no |
| EXU | yes | no | no | no | no |
| UPD | yes | no | no | yes | no |
| UTI* | no | no | no | no | no |

*Some utilities permit access to a file by other users during utility execution. In this case, the utility issues an OP command with the EXU parameter specified.

If no keyword is supplied (in other words, if the record buffer is set to '.'), the user automatically becomes an ET logic user. In this case, the record buffer length can be set to zero in the Adabas control block and the record buffer need not be supplied.

## User Queue Element

During the time that a user is active, Adabas maintains a user queue element (UQE) for the user. The UQE contains a list of file numbers for the files the user is currently using. The file list is generated when the user issues an OP command and may be modified during the user session. If no OP command is issued, the file list will initially contain no files.

Each file in the file list is marked as one of the following:

- ACC (access only)
- EXF (read- and write-protected against all other users)
- EXU (access and update and under exclusive control)
- UPD (access and update)
- UTI (access and update and in use by an Adabas utility)

If a subsequent attempt is made to access a file that is not currently in the user's file list, a test will be made to determine whether the file is currently in use by an Adabas utility. If not, the file will be added to the user's file list and marked as ACC.

If a subsequent attempt is made to update a file that is not currently in the user's file list, the following tests are applied:

- Does the request conflict with the user type? For example, an access-only user may not issue update commands;
- Is the user's file list restricted because command option 1 in the OP command specifies the value `R'?;
- Is the file to be updated under exclusive control of another user or Adabas utility?

If the file is determined to be available for the user, the file is added to the user's file list and marked as UPD.

## Using OP to close previous session of same user

The following special feature is activated only if the nucleus has been started with OPTIONS= OPEN_REQUIRED.

If a user session is terminated without a CL (close) command, e.g. by switching off and re–booting a PC, the Adabas nucleus still considers the connection to be active. If the user wants to start a new session with the same user ID, the nucleus handles this OP command as if it was issued by a different user because of the new process ID, and so returns a response code to the user.

However, if this user keeps trying to start the new session and the first session remains inactive for a defined time frame (about 60 seconds) then the first session will be terminated. This termination may result in a backout transaction command (BT) for the first session. The second session for the user will then be able to start. If the first session in the meantime issues another command, the error response code 9 will be returned to the first session, and additional information indicating "open required" is returned in the field ADDITIONS_2 of the Control Block. See the Messages and Codes manual, response code 9, value "open required" in field ADDITIONS_2 for further details.

## Examples

**Example 1: Access-only user with character set UTF-16BE**

An access-only user session is to be opened with the character set UTF-16BE for W fields.

**Control Block**

```
Command Code          OP
```

```
Record Buffer Length  24 (or larger)
```

**Buffer Areas:**

```
Record Buffer         WCHARSET='UTF-16BE',ACC.
```

**Example 2: ET logic user**

A user session is to be opened, in which the user intends to access files 8 and 9, and update files 8 and 16. The user intends to store user data in an Adabas system file during the session. The user data which was stored during the previous session is to be read. The USERID for the user is USER0001.

**Control Block:**

```
Command Code          OP
```

```
Record Buffer Length  15 (or larger)
```

```
Command Option 2      E (user data is to be read)
```

```
Additions 1           USER0001 (USERID is required if user data
                                is to be stored and/or read)
```

**Buffer Areas:**

```
Record Buffer         ACC=9,UPD=8,16.
```

**Example 3: Exclusive control user without ET logic**

A user session is to be opened, in which the user wishes to have exclusive control of files 10, 11 and 12. The user does not intend to use ET commands and does not intend to store and/or read user data in/from an Adabas system file.

**Control Block:**

```
Command Code          OP
```

```
Record Buffer Length  13 (or larger)
```

```
Command Option 2      b (user data is not to be stored or read)
```

```
Additions 1           bbbbbbbb (USERID is not required)
```

**Buffer Areas:**

```
Record Buffer         EXU=10,11,12.
```

**Example 4: Exclusive control user with ET logic**

A user session is to be opened, in which the user wishes to have exclusive control of files 10,11 and 12. The user intends to use ET commands.

**Control Block:**

```
Command Code          OP
```

```
Record Buffer Length  26 (or larger)
```

```
Command Option 2      b (user data is not to be stored or read)
```

```
Additions 1           bbbbbbbb (USERID is not required)
```

**Buffer Areas:**

```
Record Buffer         EXU=10,11,12,UPD=10,11,12.
```

**Example 5: User-defined timeout value**

A user session is to be opened, in which the user intends to update files 5 and 7. The user ID of the user is USER0002. The user session will run with timeout values TNAE=30 minutes and TT=10 minutes.

**Control Block:**

```
Command Code          OP
```

```
Record Buffer Length  8
```

```
ISN Lower Limit       1800  (1800 seconds = 30 minutes)
```

```
ISN Quantity          600   (600 seconds = 10 minutes)
```

```
Additions 1           USER0002
```

**Buffer Areas:**

```
Record Buffer         UPD=5,7.
```

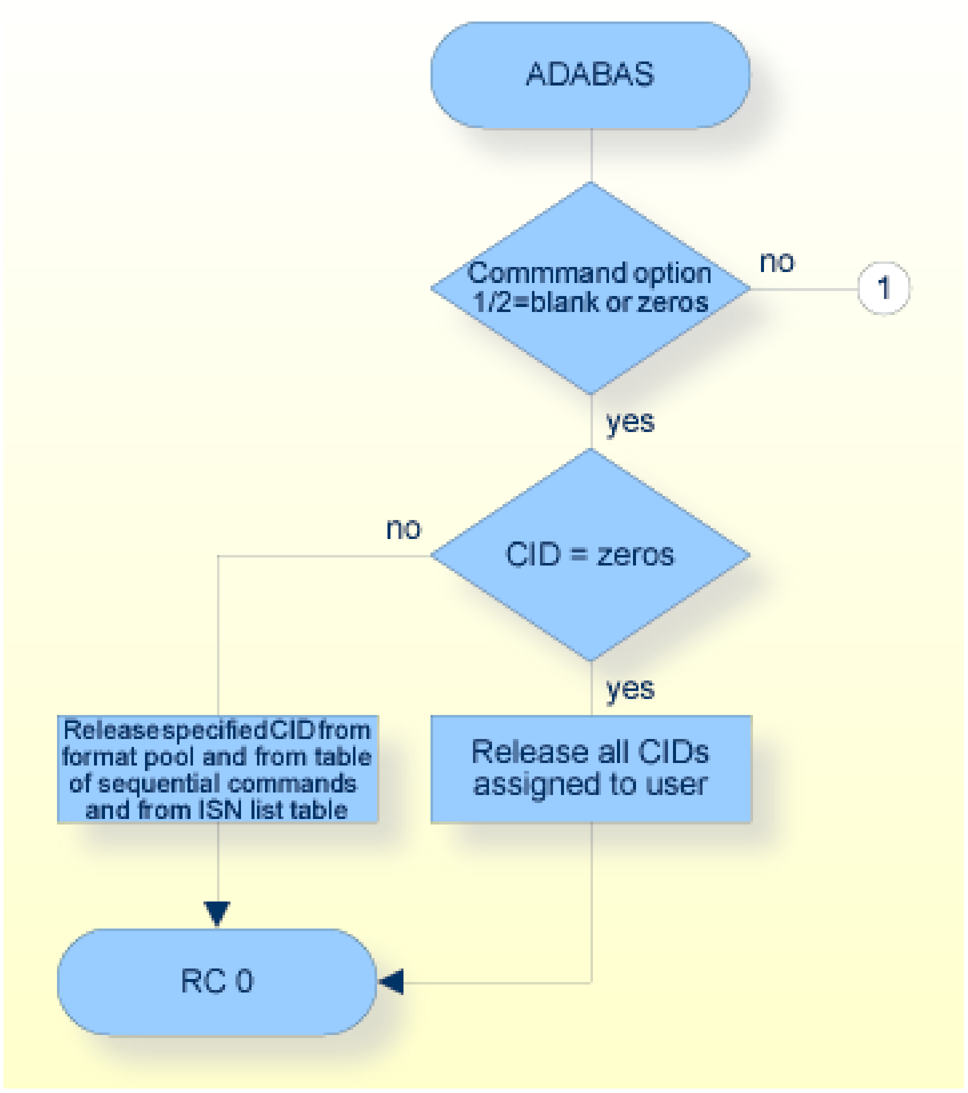
# 23 RC Command (Release Command ID)

## Function and Use

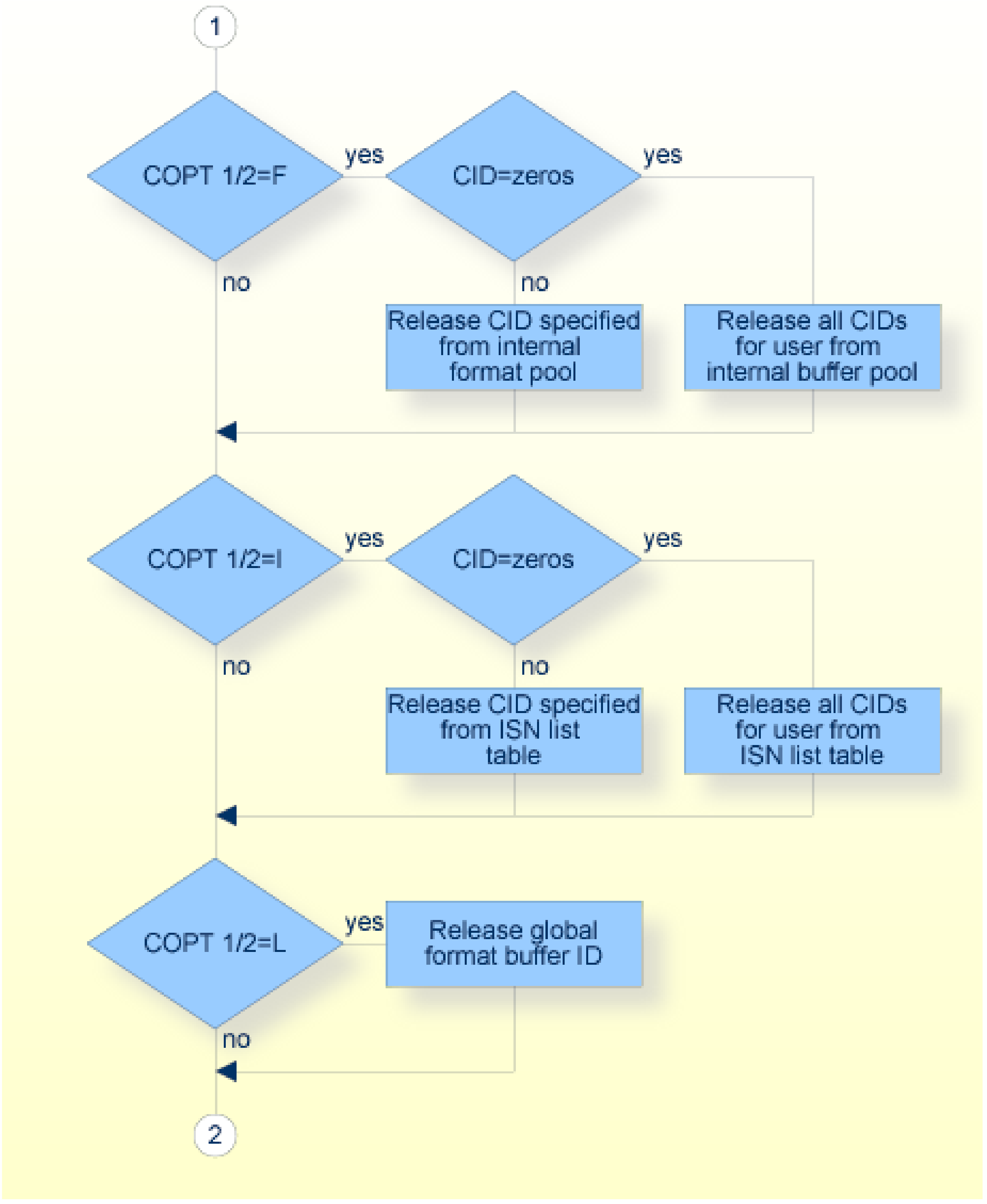
The RC command is used to release one or more command IDs that are currently assigned to a user.

A command ID should be released under any of the following circumstances:

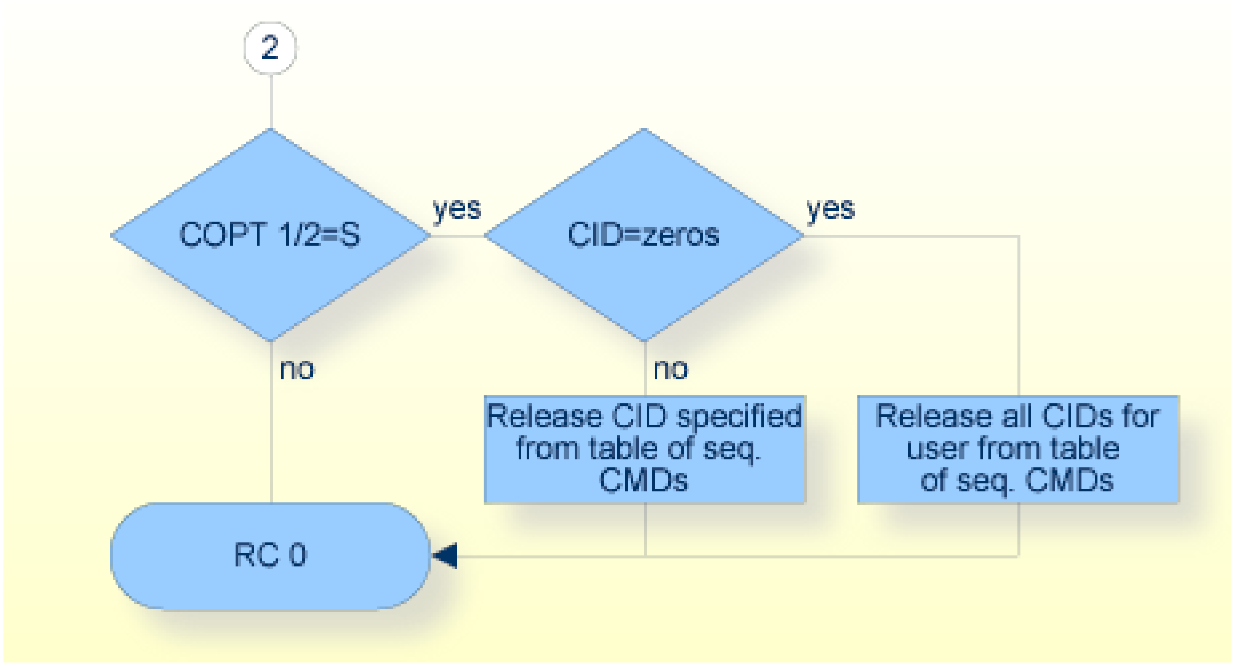
- The user has completed the processing of an ISN list which is stored on the Adabas temporary working space as a result of an Sx command in which the SAVE ISN LIST option was used. This will permit Adabas to reuse the space currently occupied by the list;
- The user wishes to terminate a sequential pass of a file (L2/L5, L3/L6, L9 command) before reaching an end–of–file condition;
- The user has completed a series of L1/L4, A1, N1/N2 commands in which a non-blank command ID was used.


ADABAS
Commmand option 1/2=blank or zeros
no
1
yes
no
CID = zeros
yes
Release specified CID from format pool and from table of sequential commands and from ISN list table
Release all CIDs assigned to user
RC 0

**RC Command, Procedure Flow**


1
COPT 1/2=F
yes
CID=zeros
yes
no
no
Release CID specified from internal format pool
Release all CIDs for user from internal buffer pool
COPT 1/2=I
yes
CID=zeros
yes
no
no
Release CID specified from ISN list table
Release all CIDs for user from ISN list table
COPT 1/2=L
yes
Release global format buffer ID
no
2

**RC Command, Procedure Flow (continued)**


2
COPT 1/2=S
yes
CID=zeros
yes
no
no
Release CID specified from table of seq. CMDs
Release all CIDs for user from table of seq. CMDs
RC 0

**RC Command, Procedure Flow (continued)**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| Command Option 1 | A | F/U |
| Command Option 2 | A | F/U |
| Additions 2 | A,B | -/A |
| Additions 5 | A | F/U |
| Command Time | B | -/A |
| User Area | | F/U |

### Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | –/– |
| Record Buffer | –/– |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | –/– |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
RC

**Command ID**
The local command ID or separate format buffer ID that is to be released is specified in this field. A value of binary zeros will release all of the command IDs currently assigned to the user.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**Command Option 1/2**
These fields are used to indicate that a command ID is to be released from the internal format buffer pool, the ISN list table or the table of sequential commands. Information about each of the above tables is contained in *Programming Considerations, Using Command IDs*.

An `F' in either of these fields indicates that the command ID(s) is/are to be released if contained in the internal format buffer pool.

An `S' in either of these fields indicates that the command ID(s) is/are to be released if contained in the table of sequential commands.

An `I' in either of these fields indicates that the command ID(s) is/are to be released if contained in the ISN list table.

If both of these fields are set to blanks or binary zeros, the command ID(s) will be released from all of the above tables in which it is or they are present.

An `L' in either of these fields indicates that a global format buffer ID is to be released. The format buffer ID is provided in the Additions 5 field.

**Additions 2**

For some response codes, Adabas returns detailed information in this field. See *Adabas Messages And Codes* for further information.

**Additions 5**

This field is used to provide the global format buffer ID that is to be released if the command option 1/2 field contains an `L'.

## Examples

**Example 1:**

The command ID X003 is to be released.

**Control Block**

```
Command Code          RC
```

```
Command ID            X003 (command ID X003 to be released)
```

```
Command Option 1/2    bb (all CID types to be released)
```

**Example 2:**

All command IDs currently assigned to the user are to be released.

## Control Block

```
Command Code         RC
```

```
Command ID           0x00000000       (binary zeros  indicate that
                     (or ^X00000000)   all command IDs are to be released)
```

```
Command Option 1/2    bb (all CID types to be released)
```

**Example 3:**

All of the command IDs assigned to the user and which are contained in the table of sequential commands or the internal format buffer pool are to be released.

**Control Block**

```
Command Code          RC
```

```
Command ID            0x00000000       (binary zeros  indicate that
                      (or ^X00000000)   all command IDs are to be released)
```

```
Command Option 1      F (F indicates that command IDs contained
                         in the internal format buffer pool are to
                         be released)
```

```
Command Option 2      S (S indicates that command IDs contained
                         in the table of sequential  commands  are
                         to be released)
```

# 24 RE Command (Read ET User Data)

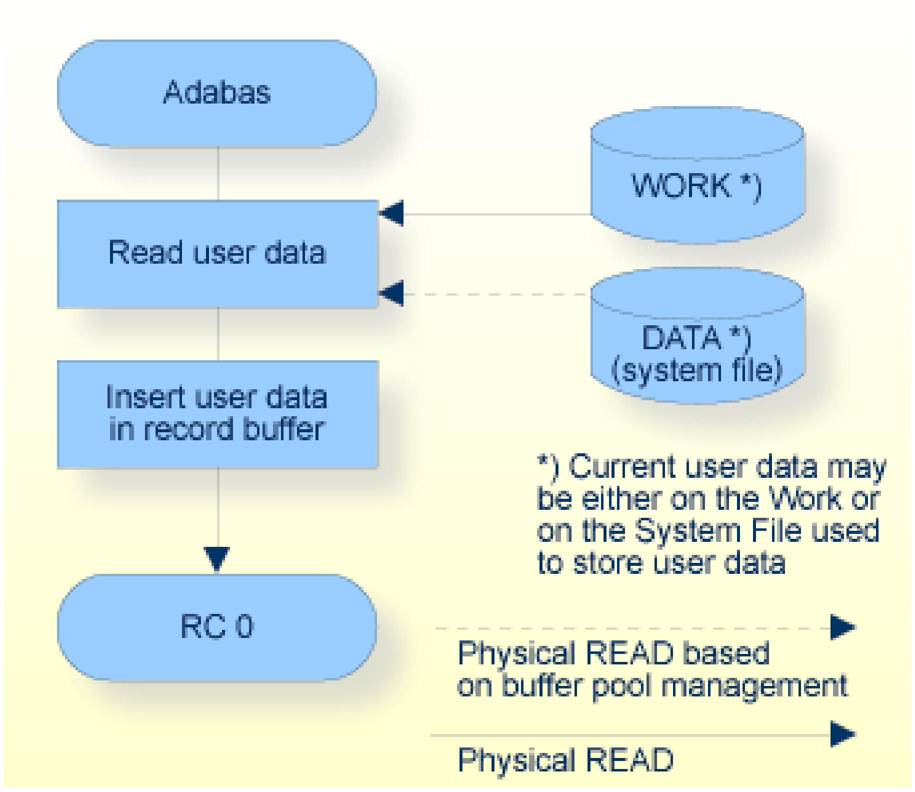
## Function and Use

The RE command is used to read user data which has been previously stored in an Adabas system file by a CL or ET command.

This user data may be needed for a user restart following abnormal termination of a user or Adabas session.

User data from a previous session may only be read if the user has specified the same USERID with the OP command for both this session and the session in which the user data was stored.

User data stored by another user may be read if the USERID of the user who stored the data is known.


Adabas
WORK *)
Read user data
DATA *) (system file)
Insert user data in record buffer
*) Current user data may be either on the Work or on the System File used to store user data
RC 0
Physical READ based on buffer pool management
Physical READ

**RE Command, Procedure Flow**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | -/A |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| Record Buffer Length (ACB only) | B | F/U |
| Command Option 1 | A | F/U |
| Additions 1 | A | F/U |
| Additions 2 | A,B | -/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | */– |
| Record Buffer | –/A |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | –/– |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |
| * | Not used but must be included in parameter list of CALL statement |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
RE

**Command ID**
If the user who stored the user data is an ET logic user, Adabas will return the transaction sequence number of the last successful logical transaction for the user in this field.

If the user is not active and the last session was terminated normally, a 0 is returned.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**Record Buffer Length (ACB only)**
The length of the record buffer. The length specified determines the number of bytes of user data to be returned.

If the length specified is less than the number of bytes of user data available, only the specified number of bytes will be inserted in the record buffer and the rightmost bytes will be truncated. A response code 2 is returned.

**Command Option 1**
An `I' in this field indicates that user data stored by another user is to be read.

**Additions 1**
If user data stored by another user is to be read, this field must contain the USERID of the user who stored the data.

**Additions 2**
If the user who stored the data being read was an ET logic user, Adabas will return the transaction sequence number of the user's last successfully completed transaction for which user data was stored with an ET or CL command in this field.

## Record Buffer

Adabas returns the user data in this buffer.

If no ET data are found, the record buffer is filled with blanks.

If the record buffer is larger than the user data, the record buffer is padded with blanks. If the record buffer is smaller than the user data, the user data are truncated.

## Examples

**Example 1:**

The user wishes to read the user data which he previously stored with an ET command.

**Control Block**

```
Command Code          RE
```

```
Record Buffer Length  100 (100 bytes user data is to  be  read)
```

```
Command Option 1      b (the user data to be read was stored by
                         this user)
```

**Example 2:**

The user wishes to read user data stored by another user (USERID = USER6666).

**Control Block**

```
Command Code          RE
```

```
Record Buffer Length  150 (150 bytes of user data to be read)
```

```
Command Option 1      I (the user data to be read was stored by
                         another user)
```

```
Additions 1           USER6666  (USERID  of the user who stored
                                 the user data)
```

# 25 RI Command (Release Record)

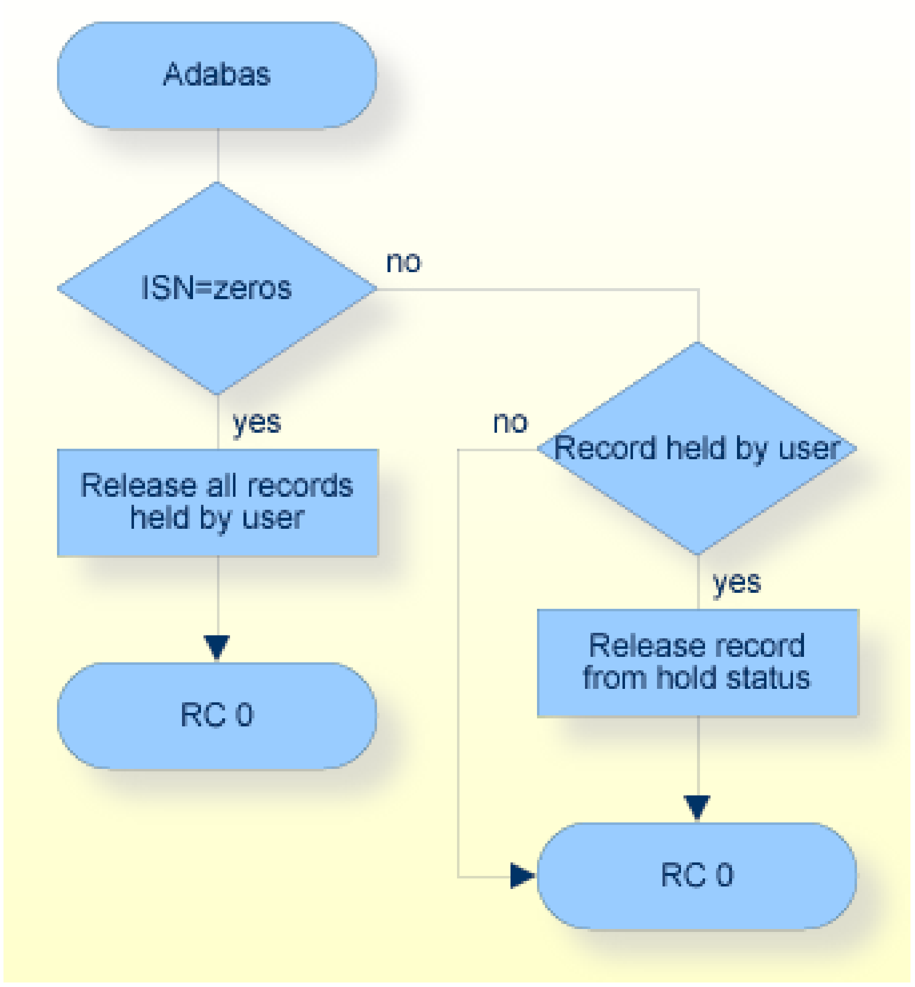
## Function and Use

The RI command is used to release the lock for a record or to downgrade the lock level from exclusive to shared.

The user specifies the file and ISN of the record to be released.

An option is available which permits the user to release all of the records currently being held by the user.

This command will be rejected for records which have been updated.


Adabas
ISN=zeros
no
yes
Release all records held by user
RC 0
no
Record held by user
yes
Release record from hold status
RC 0

**RI Command, Procedure Flow**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN | B | F/A |
| Command Option 3 (ACBX only) | A | F/U |
| Additions 2 | A,B | -/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | –/– |
| Record Buffer | –/– |
| Search Buffer | –/– |
| Value Buffer | –/– |
| ISN Buffer | –/– |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
RI

**File Number**
The number of the file which contains the record to be released.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully. Response code 0 will also be returned if the specified record was not in hold status for the current user. If the specified record is in hold status for another user, it will not be released, but response code 0 will be returned.

**ISN**
The ISN of the record to be released.

If all the records held by the user are to be released, this field must be set to binary zeros.

**Command Option 3 (ACBX only)**
A blank in this field releases the record completely from hold status. An 'S' in this field downgrades the locking mode from exclusive to shared.

## Example

The record identified by ISN 3 in file 2 is to be released from hold status.

**Control Block**

```
Command Code          RI
```

```
File Number           2 (record to be released is in file 2)
```

```
ISN                   3 (record with ISN 3 is to be released)
```

# 26 S1/S2/S4 Command (Find Records)

## Function and Use

The S1/S2/S4 commands are used to select a set of records which satisfy given search criteria.

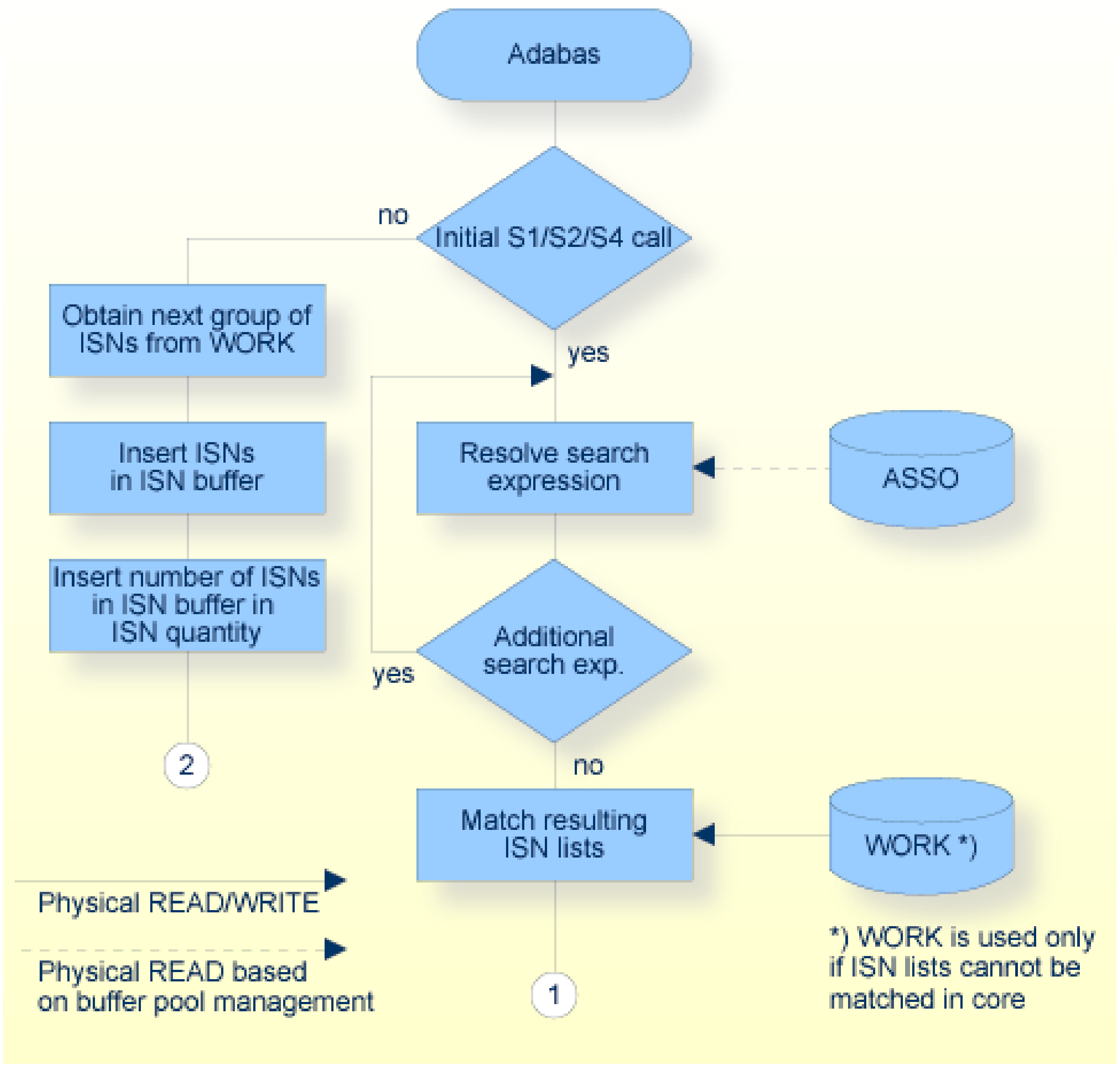
The result of an S1/S2/S4 command is the set of records which satisfy the query along with a list of the ISNs of the records. If the S1/S4 command is used, the ISNs are returned in ascending sequence in the ISN buffer. If the S2 command is used, the ISNs are returned according to a user-specified sort sequence. The S4 command places the first ISN in the resulting ISN list in hold status, i.e. lock the record for other users. The S4 command should be used if the user needs to prevent other users from updating the record, e.g. because he wants to update the record. Please refer to *Concepts and Facilities, Competitive Updating, Shared Locks* for further information. If the user already holds a lock of a lower locking mode for the record the lock is upgraded to the locking mode requested. If the user already holds a higher locking mode for the re-cord, the locking mode remains unchanged.

Adabas will store (if requested) on the Adabas temporary working space any ISNs which could not be inserted in the ISN buffer on the initial S1/S2/S4 call. These overflow ISNs may be retrieved subsequently by using further S1/S2/S4 calls. See *Programming Considerations, ISN List Processing* for additional information.

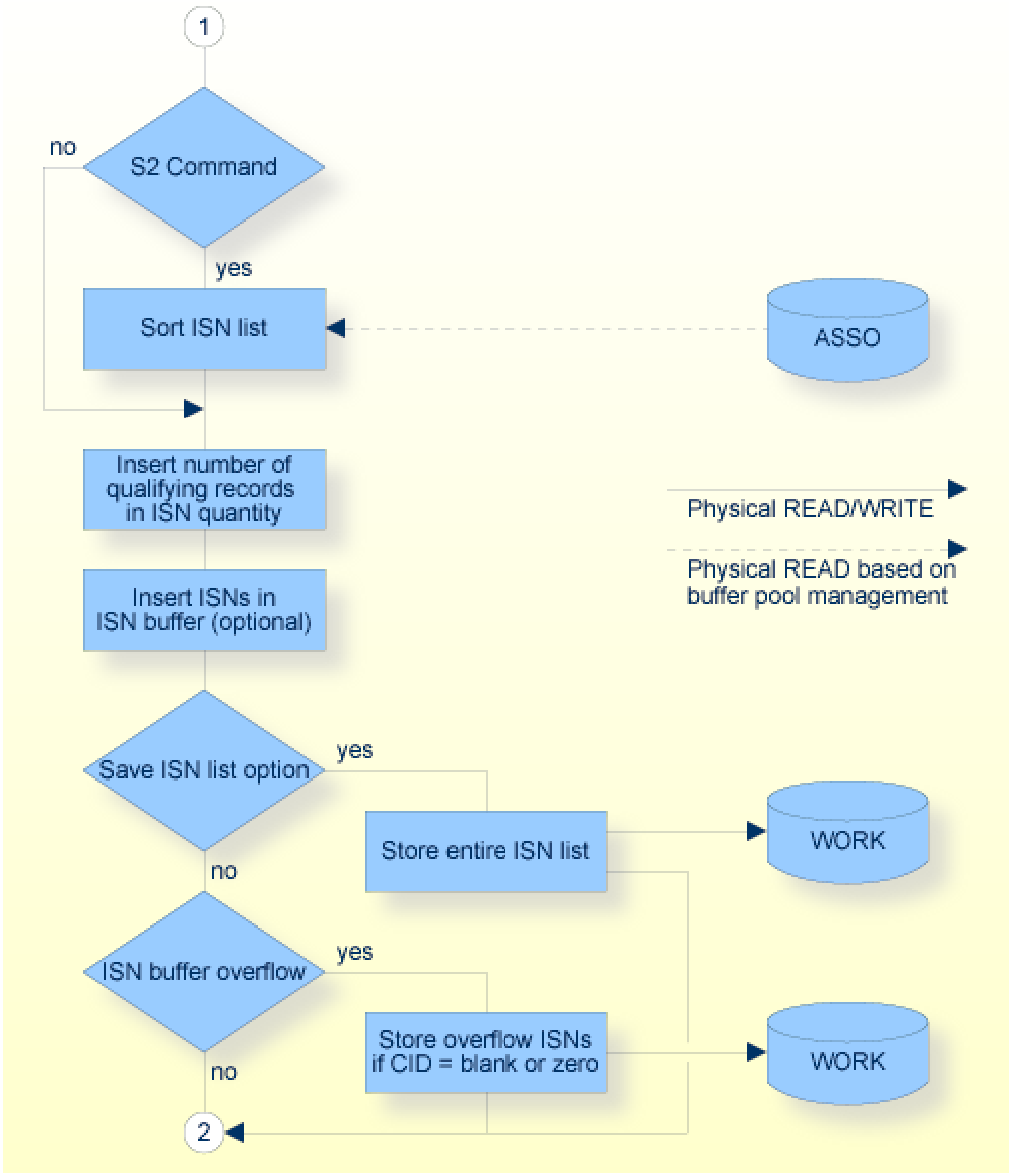
Adabas will release an overflow ISN list when the last ISN in the list has been returned to the user. If the user needs to retain the entire ISN list indefinitely, the SAVE ISN LIST option may be used. If this option is specified, the entire ISN list is stored on the Adabas temporary working space and is not released until an RC or CL command is issued (or the Adabas session is terminated).

If the user intends to read the records identified by the ISNs in the list by using the GET NEXT option of the L1/L4 command, the ISN buffer is not required. ISNs are obtained automatically from the stored ISN list by Adabas when this option is used.

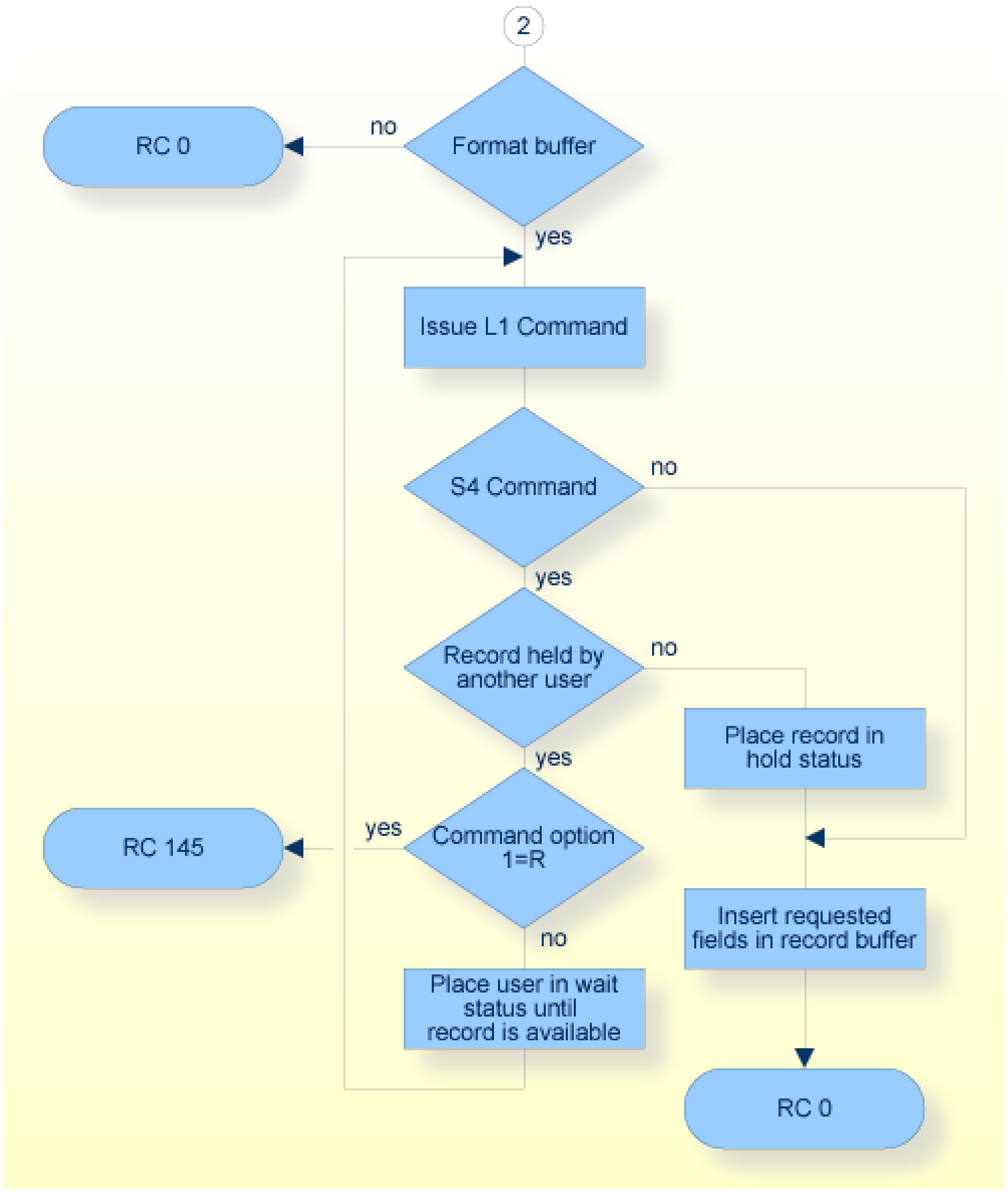
The record identified by the first ISN in the resulting ISN list may optionally be read from Data Storage with the S1/S2/S4 command by providing a format buffer.


Adabas
no
Initial S1/S2/S4 call
Obtain next group of ISNs from WORK
yes
Insert ISNs in ISN buffer
Resolve search expression
ASSO
Insert number of ISNs in ISN buffer in ISN quantity
Additional search exp.
yes
2
no
Match resulting ISN lists
WORK *)
Physical READ/WRITE
Physical READ based on buffer pool management
1
*) WORK is used only if ISN lists cannot be matched in core

**S1/S2/S4 Command, Procedure Flow**


1
S2 Command
no
yes
Sort ISN list
ASSO
Insert number of qualifying records in ISN quantity
Physical READ/WRITE
Physical READ based on buffer pool management
Insert ISNs in ISN buffer (optional)
Save ISN list option
yes
Store entire ISN list
WORK
no
ISN buffer overflow
yes
Store overflow ISNs if CID = blank or zero
WORK
no
2

**S1/S2/S4 Command, Procedure Flow (continued)**


2
RC 0
no
Format buffer
yes
Issue L1 Command
S4 Command
no
yes
Record held by another user
no
Place record in hold status
yes
RC 145
yes
Command option 1=R
Insert requested fields in record buffer
no
Place user in wait status until record is available
RC 0

**S1/S2/S4 Command, Procedure Flow (continued)**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN | B | -/A |
| ISN Lower Limit | B | F/U |
| ISN Quantity | B | -/A |
| Format Buffer Length (ACB only) | B | F/U |
| Record Buffer Length (ACB only) | B | F/U |
| Search Buffer Length (ACB only) | B | F/U |
| Value Buffer Length (ACB only) | B | F/U |
| ISN Buffer Length (ACB only) | B | F/U |
| Command Option 1 | A | F/U |
| Command Option 2 | A | F/U |
| Command Option 3 (ACBX only) | A | F/U |
| Additions 1 | A | F/U |
| Additions 2 | A,B | -/A |
| Additions 3 | A | F/A |
| Additions 5 | A | F/U |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | F/U |
| Record Buffer | –/A |
| Search Buffer | F/U |
| Value Buffer | F/U |
| ISN Buffer | –/A |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
S1/S2/S4

**Command ID**
This field is used to provide a value to identify the resulting ISN list if it is to be stored on the Adabas temporary working space.

If the SAVE ISN LIST option is to be used, or if overflow ISNs are to be stored, a non-blank, non-zero value must be provided in this field.

The high-order byte of this field must not be set to hexadecimal `FF', except when automatic command ID generation is used (see *Programming Considerations, Using Command IDs* for additional information).

See *Programming Considerations, ISN List Processing* for additional information.

**File Number**
The number of the file from which the ISNs are to be selected.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**ISN**
Adabas returns the first ISN of the resulting ISN list in this field. If there were no resulting ISNs, this field is not modified. This applies to both the initial call and any subsequent calls which are used to retrieve ISNs from the Adabas temporary working space.

**ISN Lower Limit**

This field may be used in an initial Sx call to limit the resulting ISN list to those ISNs which are greater than the ISN specified in this field. If this field is set to zeros, Adabas will return all qualifying ISNs.

In a subsequent Sx call, this field is used when a group of ISNs from a saved ISN list is being retrieved from the Adabas temporary working space, in order to determine the first ISN to be used.

**ISN Quantity**

Adabas returns, as a result of an initial Sx call, the number of records which satisfy the search criteria in this field.

Adabas returns, as a result of a subsequent Sx call used to retrieve ISNs from the Adabas temporary working space, the number of ISNs returned in the ISN buffer.

**Format Buffer Length (ACB only)**

The format buffer length (in bytes). The format buffer area defined in the user program must be as large as (or larger than) the length specified.

**Record Buffer Length (ACB only)**

The record buffer length (in bytes). The record buffer area defined in the user program must be as large as (or larger than) the length specified.

**Search Buffer Length (ACB only)**

The search buffer length (in bytes). The search buffer area defined in the user program must be as large as (or larger than) the length specified.

**Value Buffer Length (ACB only)**

The value buffer length (in bytes). The value buffer area defined in the user program must be as large as (or larger than) the length specified.

**ISN Buffer Length (ACB only)**

The ISN buffer length (in bytes). This length is used to determine the number of ISNs placed in the ISN buffer.

If this field is set to zeros, no ISNs will be inserted in the ISN buffer. This field should be set to zeros if the resulting ISN list is to be read with the GET NEXT option of the L1/L4 command, or if the command is being issued only to determine the number of qualifying records.

If a non-zero value is specified, it should be a multiple of 4. If it is not, Adabas will reduce the length to the next-lowest integer which is a multiple of 4.

**Command Option 1**

An `H' in this field invokes the SAVE ISN LIST option. This option causes Adabas to store the entire resulting ISN list on the Adabas temporary working space.

An `R' in this field indicates that the return option is to be used. If an S4 command is issued and the record to be read and held is currently held by another user, Adabas will return response code 145 rather than place the user in hold status until the record becomes available.

**Command Option 2**

If the S2 command is being used, this field is used to indicate whether the resulting ISNs are to be sorted in ascending or descending sequence.

A `D' indicates that descending sequence is to be used.

A 'T' indicates that the index truncation check is to be preformed. If this option is specified, the search buffer must contain a descriptor with the TR option, and the value buffer must contain the corresponding value. In this case, no search operation is performed, there is only a check made to see whether or not the specified descriptor value is truncated in the index: response code 0 indicates no truncation, response code 2 indicates truncation.

**Command Option 1/2**

An `I' in either of these fields causes the release of the CID value specified in the command ID field as the first action taken during command execution.

**Command Option 3 (ACBX only)**

The command option 3 is relevant only for an S4 command.

A ‘C’ in this field indicates that a shared lock is to be acquired for the first record found for only as long as the command is active. If the record was already locked before, the record remains locked. Using this option avoids dirty reads: you see only the committed states of the record.

An ‘S’ in this field indicates that the first record found is to be placed in shared hold status. The lock will be released again when the current transaction is committed or backed out. If the command belongs to a subtransaction, the lock is also released when the current subtransaction is backed out. You can also release the lock with an RI command.

A ‘Q’ in this field is allowed only for an S4 command with a command ID and ISN buffer length 4, and indicates that the first record found is to be placed in shared hold status. The lock is released again at start of next sequential read command for this read sequence, or when one of the events happens, which releases a record read with ‘S’ option, whichever happens first. If the same record is read by more than one command with the ‘Q’ option, the record is only released when, for all these command sequences, either the next record has been read or an RC command has been issued. If the same record has also been read by another command with the ‘S’ option, or the record has been locked exclusively, the record is not released by reading the next record of the command sequence. The 'Q' option is not allowed in combination with command option 1 = ‘M’ (multifetch feature).

**Note:** If an S4 command finds only one record and the command option 1 is not equal to 'H', no ISN list entry is generated. Therefore you cannot perform a sequential read command for the command ID, which would release the record if it was read with the 'Q' option. This means that the record read with the 'Q' option remains in shared hold status until an event occurs that would also release a record that is read with the 'S' option.

A blank in this field indicates that the record is locked exclusively.

**Additions 1**

If the S2 command is being used, this field is used to specify the field name(s) to be used to control the sort sequence. From one to four names may be specified. Descriptor or non-descriptor field names may be used, but they must not be derived from a periodic group. Hyperdescriptors may be specified but not in combination with non-descriptor field names. Phonetic descriptors must not be specified. Subdescriptors and superdescriptors may be specified if they are not derived from a PE group. A multiple-value field may be specified, in which case the ISNs will be sorted corresponding to the highest value present within a given record.

The descriptors are specified beginning with byte 1 (left-justified) and any remaining positions must be set to blanks ("b" in the following example).

**Example:**

```
 XXYYbbbb
```

```
  XX = major sort descriptor
  YY = minor sort descriptor
```

If using non-descriptor fields, all of their values must be within the standard FDT length if the length is not equal to zero, otherwise response 1 may be returned.

If a descriptor defined with the NU option and containing null values is used for the sort sequence, the null values appear at the beginning of the sorted list.

**Example:**

```
Sorted list with NU option     :  0, -1, +1

Sorted list without NU option  :  -1, 0, +1
```

**Additions 2**

If at least one record is found, and at least one Adabas field is requested in the format buffer, Adabas will return the compressed record length of the record which has been read in this field. The length is provided in the first two bytes in binary format.

The last two bytes of this field contain the length of the decompressed fields selected by the Format buffer, in binary format.

For some response codes, Adabas returns detailed information in this field. See Adabas Messages And Codes for further information.

**Additions 3**

This field is used to provide a security password.

If the file to be used is not security protected, this field should be set to blanks. If the file is security protected, the user must provide a valid password.

Adabas sets this field to blanks during command processing to protect the integrity of any password provided.

**Additions 5**

This field may be used to provide a separate format buffer ID that is used to identify the internal format buffer used for this command, or to provide a global format buffer ID.

As long as the first byte of the Additions 5 field is not alphanumeric, the value provided in the command ID field will also be used as the format buffer ID.

If the first byte is a lower case character, the bytes 5 to 8 of the Additions 5 field will be used as the separate local format buffer ID.

If the first byte is a digit or an upper case character, the Additions 5 field (8 bytes) will be used as a separate global format buffer ID, which means that the format buffer ID can be used by several users in parallel.

See *Programming Considerations, Using Command IDs* for additional information and examples.

## Format Buffer

If the record identified by the first ISN in the resulting ISN list is to be read from Data Storage, the fields within the record for which values are to be returned must be specified in this buffer.

The syntax and examples of format buffer construction are provided in *Calling Adabas, Format and Record Buffers*.

If no read is to be performed, the first non-blank character in this buffer must be a period.

## Record Buffer

If a format buffer is provided, Adabas returns the requested field values in this buffer.

The values are returned according to the standard length and format of the field, unless the user has explicitly requested a different length and/or format in the format buffer.

## Search and Value Buffers

These buffers are used to define the search criterion. The search expression (or expressions) is provided in the search buffer, and the values which correspond to the search expressions are provided in the value buffer.

The syntax and examples of search and value buffer construction are provided in *Calling Adabas, Search and Value Buffers*.

## ISN Buffer

Adabas places the list of resulting ISNs in this buffer. Each ISN is returned as a four-byte binary number.

The ISNs are returned in ascending ISN sequence unless the S2 command is being used, in which case they are returned in the user-specified sort sequence.

If the ISN buffer length field is set to zeros, no ISNs will be returned in the ISN buffer.

If the ISN buffer length is not zero, the ISN buffer is not large enough to contain all the resulting ISNs and a non-blank, non-zero command ID was used, Adabas will store the overflow ISNs on the Adabas temporary working space. These ISNs may then be retrieved using further S1/S2/S4 calls in which the same command ID is used. See *Programming Considerations, ISN List Processing* for additional information.

## Examples

**Note:** All the examples in this section refer to the files in *Appendix A*.

**Example 1:**

The set of records in file 1 which contain a value in the range A to J for the descriptor AA is to be selected.

**Control Block:**

| | |
|---|---|
| Command Code | S1 |
| Command ID | bbbb (no ISNs are to be stored on the Adabas temporary working space) |
| File Number | 1 |
| ISN Lower Limit | 0 (all qualifying ISNs are to be returned) |
| Format Buffer Length | 1 (or larger) |
| Search Buffer Length | 12 (or larger) |
| Value Buffer Length | 2 (or larger) |
| ISN Buffer Length | 200 (no more than 50 ISNs are expected) |
| Command Option 1 | b (SAVE ISN LIST option not used) |
| Additions 3 | bbbbbbbb (the file is not security protected) |

**Buffer Areas:**

| | |
|---|---|
| Format Buffer | . (no read to be done) |
| Search Buffer | AA,1,S,AA,1. |

```
Value Buffer          C'AJ'
```

**Example 2:**

FIND with READ option. The ISN of the record containing the value ABCDEFGH for the field AA in file 1 is to be selected. The record is also to be read from Data Storage with the value for the field AC to be returned.

**Control Block:**

```
Command Code          S1
```

```
Command ID            bbbb  (no  ISNs  are to  be stored on the
                             Adabas temporary working space)
```

```
File Number           1
```

```
ISN Lower Limit       0 (all qualifying ISNs are to be returned)
```

```
Format Buffer Length  3 (or larger)
```

```
Record Buffer Length  20 (or larger)
```

```
Search Buffer Length  3 (or larger)
```

```
Value Buffer Length   8 (or larger)
```

```
ISN Buffer Length     4 (no more than 1 ISN expected)
```

```
Command Option 1      b (the Save ISN List option is not to be
                         used)
```

| | |
|---|---|
| Additions 3 | bbbbbbbb (file is not security protected) |

**Buffer Areas:**

| | |
|---|---|
| Format Buffer | AC. (the value for field AC is to be returned) |
| Search Buffer | AA. |
| Value Buffer | "ABCDEFGH"<br>0x6162636465666768<br>(or ^X6162636465666768) |

**Example 3:**

FIND with ISN buffer overflow. The set of records which contain any value in the range A to D for field AA in file 1 are to be selected. ISN buffer overflow handling is to be used.

**Control Block:**

| | |
|---|---|
| Command Code | S1 |
| Command ID | ABCD (a non blank Command ID is required) |
| File Number | 1 |
| ISN Lower Limit | 0 (all qualifying ISNs are to be returned) |
| Format Buffer Length | 1 (or larger) |
| Record Buffer Length | 0 (or larger) |

```
Search Buffer Length  12 (or larger)
```

```
Value Buffer Length   2 (or larger)
```

```
ISN Buffer Length     100  (up to 25 ISNs will be returned with
                            each call)
```

```
Command Option 1      b (SAVE ISN LIST option not used)
```

```
Additions 3           bbbbbbbb (file is not security protected)
```

**Buffer Areas:**

```
Format Buffer         .  (no read to be performed)
```

```
Search Buffer         AA,1,S,AA,1.
```

```
Value Buffer          "AD"
                      0x6164
                      (or ^X6164)
```

Adabas will return, as a result of the initial S1 call, a maximum of 25 ISNs in the ISN buffer. If more than 25 ISNs resulted from the query, the remaining ISNs will be stored on the Adabas temporary working space under the command ID ABCD. These overflow ISNs may be retrieved by repeating the S1 call using the same command ID.

**Example 4:**

FIND with SAVE ISN LIST option. All the records containing the value +80 for the field XB in file 2 are to be selected. The entire resulting ISN list is to be stored on the Adabas temporary working space.

**Control Block:**

```
Command Code          S1

Command ID            BCDE (a non blank  CID  is required  when
                            using the Save ISN List option)

File Number           2

ISN Lower Limit       0 (all qualifying ISNs are to be selected)

Format Buffer Length  1 (or larger)

Record Buffer Length  0 (or larger)

Search Buffer Length  3 (or larger)

Value Buffer Length   2 (or larger)

ISN Buffer Length     200  ( a  maximum  of  50  ISNs  will  be
                             returned on each call)

Command Option 1      H (Save ISN List option is to be used)

Additions 3           Password (file is security protected)
```

**Buffer Areas:**

```
Format Buffer         .  (no read is to be done)
```

```
Search Buffer         XB.
```

```
Value Buffer          0x080C
                      (or ^X080C)
```

The user may retrieve any group of ISNs from the ISN list which is stored as a result of this call by repeating the S1 command using the command ID BCDE. Adabas will insert as many ISNs as can be accommodated in the ISN buffer, starting with the first ISN which is greater than the ISN specified in the ISN Lower Limit field.

**Example 5:**

FIND with SORT. All records containing a value in the range A to F for the field AA in file 1 are to be selected. The resulting ISN list is to be returned in the ascending order of the values for field AB.

**Control Block:**

```
Command Code          S2
```

```
Command ID            CDEF (a non blank Command ID  is required
                            when using the S2 command)
```

```
File Number           1
```

```
ISN Lower Limit       0 (all qualifying ISNs are to be selected)
```

```
Format Buffer Length  1 (or larger)
```

```
Record Buffer Length  0 (or larger)
```

```
Search Buffer Length  12 (or larger)
```

```
Value Buffer Length   2 (or larger)
```

```
ISN Buffer Length     100  ( a  maximum  of  25  ISNs  will  be
                             returned on each call)
```

```
Command Option 1      b (the Save ISN List option is not used)
```

```
Command Option 2      b (the descending sort option is not used)
```

```
Additions 1           ABbbbbbb (resulting ISNs are to be sorted
                                on the values of field AB)
```

```
Additions 3           bbbbbbbb (file is not security protected)
```

**Buffer Areas:**

```
Format Buffer         .  (no read is to be done)
```

```
Search Buffer         AA,1,S,AA,1.
```

```
Value  Buffer         "AF"
                      0x6166
                      (or ^X6166)
```

**Example 6:**

FIND with HOLD. The record in file 1 containing the value 87654321 for field AA is to be selected. The record is also to be read and placed in hold status. The values for fields AB and AC are to be returned.

**Control Block:**

```
Command Code          S4

Command ID            bbbb  (blank Command ID may be used since
                             SAVE ISN LIST option is not  to  be  used
                             and no overflow ISNs are expected)

File Number           1

ISN Lower Limit       0 (all qualifying ISNs are to be selected)

Format Buffer Length  6 (or larger)

Record Buffer Length  22 (or larger)

Search Buffer Length  3 (or larger)

Value Buffer Length   8 (or larger)

ISN Buffer Length     4 (only 1 ISN is expected)

Command Option 1      b (the Save ISN List option is not to  be
                         used)

Additions 3           bbbbbbbb (file is not security protected)
```

**Buffer Areas:**

```
Format Buffer         AB,AC. (the record which is identified by
                              the  first  ISN is to be read, values for
                              fields AB and AC are to be returned)
```

```
Search Buffer         AA.
```

```
Value Buffer          "87654321"
                      0x6867666564636261
                      (or ^X6867666564636261)
```

**Example 7:**

Find using multiple search criteria (complex search). The set of records in file 2 containing a value of ABCD for subdescriptor SA, a value less than 80 for field XB and a value in the range MMMMM through ZZZZZ (but not Sbbbb through TZZZZ) for field XE is to be selected.

**Control Block:**

```
Command Code          S1
```

```
Command ID            GGGG (a  non blank  Command  ID  is  used
                            since Save ISN List option is to be used)
```

```
File Number           2
```

```
ISN Lower Limit       0 (all qualifying ISNs are to be selected)
```

```
Format Buffer Length  1 (or larger)
```

```
Record Buffer Length  0 (or larger)
```

```
Search Buffer Length  35 (or larger)
```

```
Value Buffer Length   27 (or larger)
```

```
ISN Buffer Length     0 (No ISNs  to  be  returned  in  the ISN
                         Buffer).
```

```
Command Option 1      H (Save ISN List option is to be used)
```

```
Additions 3           Password (file 2 is security protected)
```

**Buffer Areas:**

```
Format Buffer         .  (no read is to be done)
```

```
Search Buffer         SA,D,XB,3,U,LT,D,XE,S,XE,N,XE,S,XE.
```

```
Value Buffer          C'ABCD080MMMMMZZZZZSbbbbTZZZZ'
```

# 27 S8 Command (Process ISN Lists)

## Function and Use

The S8 command may be used to perform logical processing on two ISN lists which have been previously created with Sx commands.

The ISN lists to be processed must be in ISN sequence. ISN lists resulting from an S2 or S9 command which are not in ISN sequence may not be used.

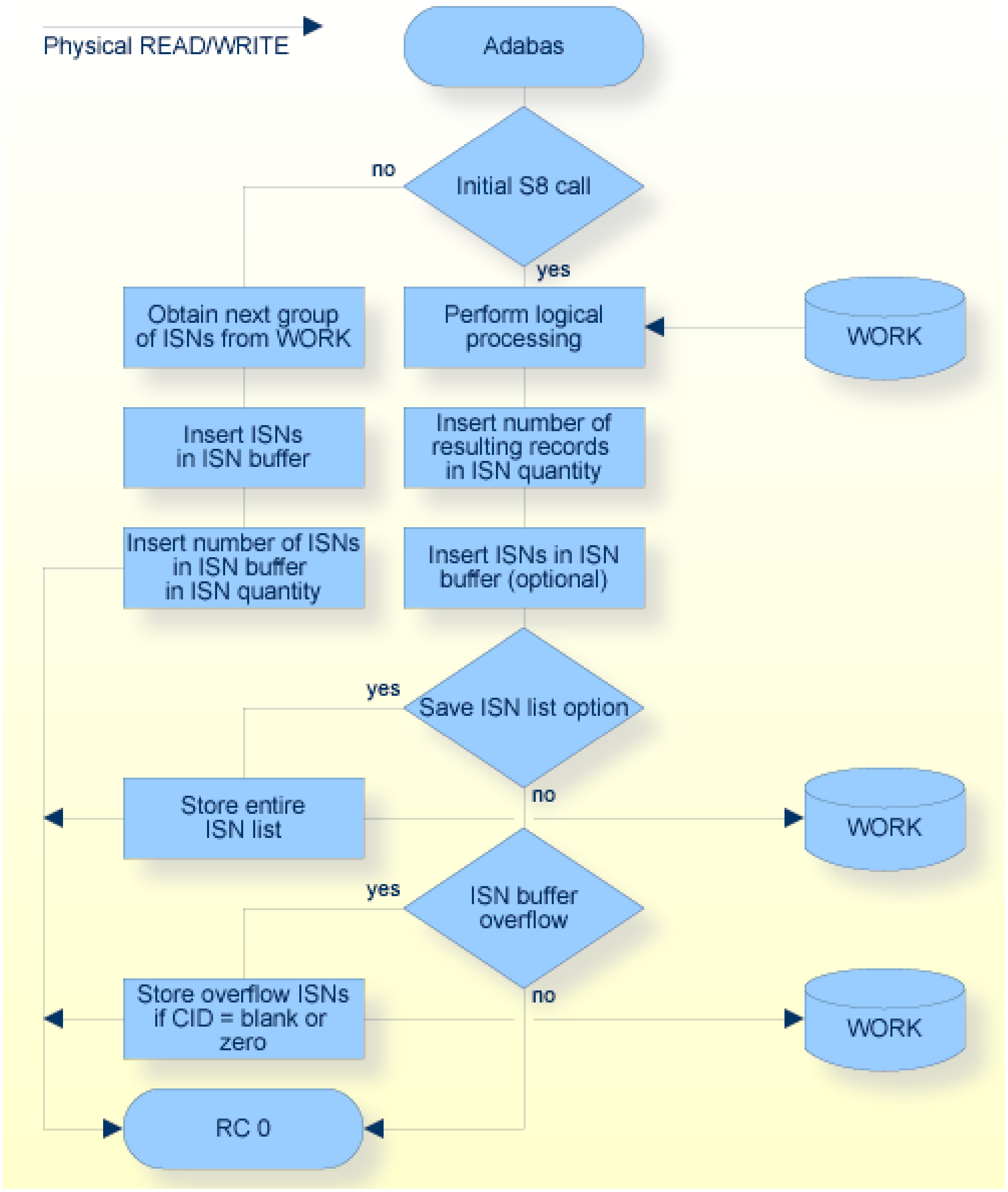
The ISN lists to be used should contain ISNs from a single Adabas file.

The S8 command can be used to perform the following logical operations:

- AND. The resulting ISN list will contain those ISNs which are present in both ISN lists;
- OR. The resulting ISN list will contain those ISNs which are present in either of the ISN lists;
- NOT. The resulting ISN list will contain those ISNs which are present in the first ISN list but not present in the second ISN list.

The resulting ISNs are returned in the ISN buffer in ascending sequence.

If one or both of the original ISN lists were stored without the SAVE ISN LIST option, the list(s) will automatically be released at the end of the S8 command.


Physical READ/WRITE
Adabas
no
Initial S8 call
yes
Obtain next group of ISNs from WORK
Perform logical processing
WORK
Insert ISNs in ISN buffer
Insert number of resulting records in ISN quantity
Insert number of ISNs in ISN buffer in ISN quantity
Insert ISNs in ISN buffer (optional)
yes
Save ISN list option
Store entire ISN list
no
WORK
yes
ISN buffer overflow
Store overflow ISNs if CID = blank or zero
no
WORK
RC 0

**S8 Command, Procedure Flow**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN | B | -/A |
| ISN Lower Limit | B | F/A |
| ISN Quantity | B | -/A |
| ISN Buffer Length (ACB only) | B | F/U |
| Command Option 1 | A | F/U |
| Command Option 2 | A | F/U |
| Additions 1 | A | F/U |
| Additions 2 | A,B | -/A |
| Additions 3 | A | F/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | * |
| Record Buffer | * |
| Search Buffer | * |
| Value Buffer | * |
| ISN Buffer | –/A |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |
| F | To be filled in by User |

| | |
|---|---|
| U | Unchanged after Adabas call |
| - | Not used |
| * | Not used but must be included in parameter list of CALL statement |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
S8

**Command ID**
This field is used to provide a value to identify the resulting ISN list if it is to be stored on the Adabas temporary working space.

If the SAVE ISN LIST option is to be used, or if overflow ISNs are to be stored, a non-blank, non-zero value must be provided in this field.

The high-order byte of this field must not be set to hexadecimal `FF', except when automatic command ID generation is used (see *Programming Considerations, Using Command IDs* for additional information).

**File Number**
The number of the file from which the ISN lists that are to be processed were obtained.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**ISN**
Adabas returns the first ISN of the resulting ISN list in this field. If there were no resulting ISNs, this field is not modified. This applies to both the initial call and any subsequent calls which are used to retrieve ISNs from the Adabas temporary working space.

**ISN Lower Limit**
This field may be used in an initial Sx call to limit the resulting ISN list to those ISNs which are greater than the ISN specified in this field. If this field is set to zeros, Adabas will return all qualifying ISNs.

This field is also used when a group of ISNs from a saved ISN list is being retrieved from the Adabas temporary working space.

**ISN Quantity**
As a result of an initial S8 call, Adabas returns the number of records which are contained in the resulting ISN list in this field.

As a result of a subsequent S8 call used to retrieve ISNs from the Adabas temporary working space, Adabas returns the number of ISNs returned in the ISN buffer.

**ISN Buffer Length (ACB only)**

The ISN buffer length (in bytes). This length is used to determine the number of ISNs placed in the ISN buffer.

If this field is set to zeros, no ISNs will be inserted in the ISN buffer. This field should be set to zeros if the resulting ISN list is to be read with the GET NEXT option of the L1/L4 command, or if the command is being issued only to determine the number of qualifying records.

If a non-zero value is specified, it should be a multiple of 4. If it is not, Adabas will reduce the length to the next-lowest integer which is a multiple of 4.

**Command Option 1**

An `H' in this field invokes the SAVE ISN LIST option. This option causes Adabas to store the entire resulting ISN list on the Adabas temporary working space.

**Command Option 2**

This field is used to indicate the logical operation that is to be performed on the ISN lists.

A `D' indicates that an AND operation is to be performed. The resulting ISN list will contain those ISNs which are present in both ISN lists.

An `O' indicates that an OR operation is to be performed on the ISN lists. The resulting ISN list will contain those ISNs which are present in either ISN list.

An `N' indicates that a NOT operation is to be performed on the ISN lists. The resulting ISN list will contain those ISNs which are present in the first list but not in the second list.

An `I' releases the CID value specified in the Command ID field as the first action taken during command execution.

**Additions 1**

The command IDs which identify the ISN lists to be processed must be specified in this field (four bytes per command ID). Each ISN list must be currently stored on the Adabas temporary working space and should contain ISNs from the same file.

**Additions 2**

For some response codes, Adabas returns detailed information in this field. See *Adabas Messages And Codes* for further information.

**Additions 3**

This field is used to provide a security password.

If the file to be used is not security protected, this field should be set to blanks. If the file is security protected, the user must provide a valid password.

Adabas sets this field to blanks during command processing to protect the integrity of any password provided.

## ISN Buffer

Adabas places the list of resulting ISNs in this buffer. Each ISN is returned as a four-byte binary number.

If the ISN buffer length is not zero, and the ISN buffer is not large enough to contain all of the resulting ISNs, and a non–blank, non–zero command ID was used, Adabas will store the overflow ISNs on the Adabas temporary working space. These ISNs may then be retrieved using further Sx calls in which the same command ID is used. See *Programming Considerations, ISN List Processing*, for additional information.

## Example

A logical OR operation is to be performed on two ISN lists in order to produce a third ISN list which contains ISNs present in either list. The ISN lists to be processed were stored on the Adabas temporary working space under the command IDs U020 and U021. The resulting ISN list is to be stored on the Adabas temporary working space under the Command ID U999. The SAVE ISN LIST option is to be used.

**Control Block:**

```
Command Code          S8
```

```
Command ID            U999  (the  resulting  ISN  list is to be
                             stored under the command ID U999)
```

```
ISN Lower Limit       0 (all of the resulting  ISNs  are to  be
                         selected)
```

```
ISN Buffer Length     0  (no ISNs are to be returned in the ISN
                          Buffer)
```

```
Command Option 1      H (Save ISN List option to be used)
```

```
Command Option 2      O (an OR operation is to be performed)
```

```
Additions 1           U020U021 (the ISN lists identified by the
                               command  IDs  U020  and  U021  are  to be
                               processed)
```

```
Additions 3           bbbbbbbb (file is not security protected)
```

# 28 S9 Command (Sort ISN List)

## Function and Use

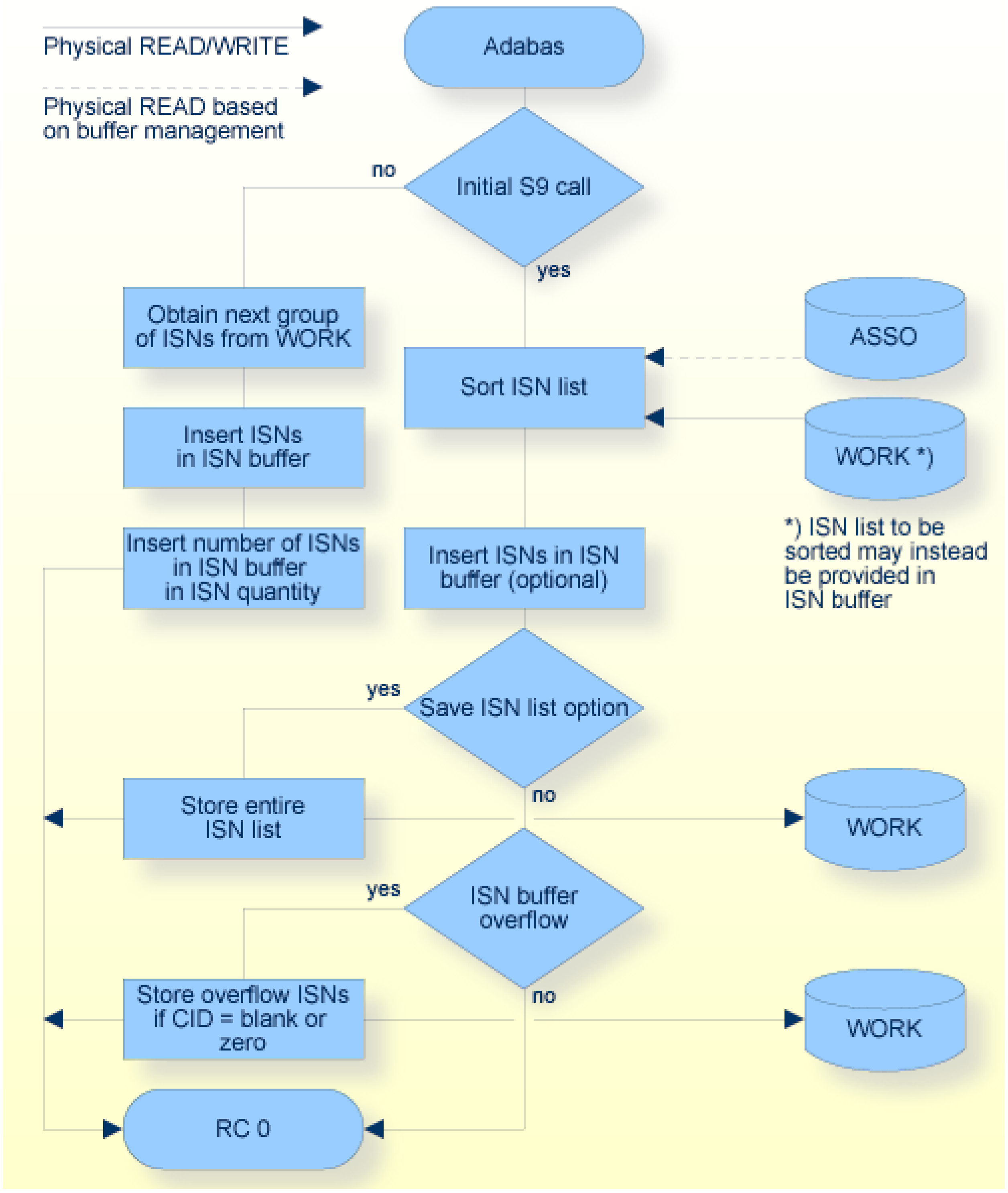
The S9 command is used to sort an ISN list which was created using an Sx command, or to sort a list of ISNs provided by the user.

Two types of sorting may be done:

- In the order of ISN values (ascending ISN sequence);
- In the order of the values of a user-specified descriptor (or descriptors). The user may specify from one to four descriptors which are to be used to control the sort sequence. Either ascending or descending sequence may be specified.

The ISN list to be sorted may either be stored on the Adabas temporary working space or may be provided in the ISN buffer.

The resulting ISN list is returned to the ISN buffer.


Physical READ/WRITE
Physical READ based on buffer management
Adabas
Initial S9 call
no
yes
Obtain next group of ISNs from WORK
Insert ISNs in ISN buffer
Insert number of ISNs in ISN buffer in ISN quantity
Sort ISN list
ASSO
WORK *)
*) ISN list to be sorted may instead be provided in ISN buffer
Insert ISNs in ISN buffer (optional)
Save ISN list option
yes
no
Store entire ISN list
WORK
ISN buffer overflow
yes
no
Store overflow ISNs if CID = blank or zero
WORK
RC 0

**S9 Command, Procedure Flow**

## Control Block

| Field | Format | |
|---|---|---|
| Call Type | B | F/U |
| Reserved (internal use) | | -/- |
| Command Code | A | F/U |
| Command ID | B | F/U |
| File Number | B | F/U (1) |
| Response Code | B | F/A (1) |
| ISN | B | -/A |
| ISN Lower Limit | B | F/U |
| ISN Quantity | B | F/A |
| ISN Buffer Length (ACB only) | B | F/U |
| Command Option 1 | A | F/U |
| Command Option 2 | A | F/U |
| Additions 1 | A | F/U |
| Additions 2 | A,B | -/A |
| Additions 3 | A | F/A |
| Additions 4 | A | F/A |
| Command Time | B | -/A |
| User Area | | F/U |

## Buffer Areas

| Buffer | |
|---|---|
| Format Buffer | * |
| Record Buffer | * |
| Search Buffer | * |
| Value Buffer | * |
| ISN Buffer | F/A |

**Formats:**

| | |
|---|---|
| A | alphanumeric |
| B | binary |

**x/y before/after Adabas call - x and y can take the values:**

| | |
|---|---|
| A | Filled in by Adabas |

|   |   |
|---|---|
| F | To be filled in by User |
| U | Unchanged after Adabas call |
| - | Not used |
| * | Not used but must be included in parameter list of CALL statement |

(1) The meaning of this field depends on the value specified for "Call Type". See *Calling Adabas, The Control Block* for details.

## Control Block

**Command Code**
S9

**Command ID**
A non-blank, non-zero value may be specified in this field.

The high-order byte of this field must not be set to a hexadecimal `FF' except when automatic command ID generation is used (see *Programming Considerations, Using Command IDs* for additional information).

**File Number**
The number of the file from which the ISN list to be sorted was obtained.

**Response Code**
Adabas returns the response code for the command in this field. Response code 0 indicates that the command was executed successfully.

**ISN**
Adabas returns the first ISN of the resulting ISN list in this field. If there were no resulting ISNs, this field is not modified. This applies to both the initial call and any subsequent calls which are used to retrieve ISNs from the Adabas temporary working space.

**ISN Lower Limit**
This field may be used in an initial S9 call to limit the resulting ISN list to those ISNs which are greater than the ISN specified in this field. If this field is set to zeros, Adabas will return all qualifying ISNs.

This field is also used when a group of ISNs from a saved ISN list is being retrieved from the Adabas temporary working space. See *Programming Considerations, ISN List Retrieval* for additional information.

**ISN Quantity**
As a result of an initial S9 call, Adabas returns the number of records which are contained in the resulting ISN list in this field.

As a result of a subsequent S9 call used to retrieve ISNs from the Adabas temporary working space, Adabas returns the number of ISNs returned in the ISN Buffer.

If the ISN list which is to be sorted is being provided in the ISN buffer, this field must contain the number of ISNs that are to be sorted - if you specify a value larger than the number of ISNs provided in the ISN buffer, it is reduced to the number of these ISNs.

**ISN Buffer Length (ACB only)**

The ISN buffer length (in bytes). This length is used to determine the number of ISNs placed in the ISN buffer.

If this field is set to zeros, no ISNs will be inserted in the ISN buffer. This field should be set to zeros if the resulting ISN list is to be read with the GET NEXT option of the L1/L4 command, or if the command is being issued only to determine the number of qualifying records.

If a non-zero value is specified, it should be a multiple of 4. If it is not, Adabas will reduce the length to the next-lowest integer which is a multiple of 4.

If the ISNs to be sorted are contained in the ISN buffer, this field must contain a value equal to or larger than the number of ISNs to be sorted multiplied by 4.

**Command Option 1**

An `H' in this field invokes the SAVE ISN LIST option. This option causes Adabas to store the entire resulting ISN list on the Adabas temporary working space.

An ‘S’ in this field tells ADABAS that the ISN list is already sorted. The ‘S’ option includes the SAVE ISN LIST option. If you specify the ‘S’ option, but the ISN list is not sorted, you get a response code 24. The ‘S’ option is only allowed if the ISN list is provided in the ISN buffer - Additions 4 must be set to blanks.

**Command Option 2**

This field may be used to invoke the descending sort option. A `D' indicates that the ISN list is to be sorted in descending order. The descending option may not be specified when sorting by ISN values.

**Command Option 1/2**

An `I' in either of these fields causes the release of the CID value specified in the command ID field as the first action taken during command execution.

**Additions 1**

The value ISNbbbbb indicates that the ISN values are to be used as the sorting sequence.

This field may also contain the name(s) that is/are to be used to control the sort sequence.

From one to four names may be specified. These may be descriptors or non descriptor field names, but they must not be derived from a periodic group. Hyperdescriptors may be specified, but not in conjunction with non-descriptor field names. Phonetic descriptors must not be specified. Subdescriptors and superdescriptors may be specified if they are not derived from a PE group. A multiple value field may be specified, in which case the ISNs will be sorted corresponding to the highest value present within a given record.

The descriptors are specified beginning with byte 1 (left-justified) and any remaining positions must be set to blanks ("b" in the following example).

**Example:**

```
  XXYYbbbb
```

```
   XX = major sort descriptor
   YY = minor sort descriptor
```

If using non-descriptor fields, all of their values must be within the standard FDT length if the length is not equal to zero, otherwise response 1 may be returned.

If a descriptor defined with the NU option and containing null values is used for the sort sequence, the null values appear at the beginning of the sorted list.

**Example:**

```
Sorted list with NU option     :  0, -1, +1

Sorted list without NU option  :  -1, 0, +1
```

**Additions 2**

For some response codes, Adabas returns detailed information in this field. See Adabas Messages And Codes for further information.

**Additions 3**

This field is used to provide a security password.

If the file to be used is not security protected, this field should be set to blanks. If the file is security protected, the user must provide a valid password.

Adabas sets this field to blanks during command processing to protect the integrity of any password provided.

**Additions 4**

If the ISN list to be sorted is contained on the Adabas temporary working space, the command ID under which the list is stored must be specified in the first 4 bytes of this field.

If the ISN list to be sorted is being provided in the ISN buffer, this field must be set to blanks.

## ISN Buffer

The ISN list that is to be sorted may be provided by the user in this buffer. If the ISNs that are to be sorted are being provided in this buffer, it must be large enough to initially contain all ISNs to be sorted.

Adabas places the list of resulting ISNs in this buffer. Each ISN is returned as a four-byte binary number.

If the ISN buffer length is not zero, and the ISN buffer is not large enough to contain all the resulting ISNs, and a non-blank command ID was used, Adabas will store the overflow ISNs on the Adabas temporary working space. These ISNs may then be retrieved using further S9 calls in which the same command ID is used. *See Programming Considerations, ISN List Processing* for additional information.

## Examples

**Example 1:**

An ISN list contained in the ISN buffer is to be sorted in ISN sequence. 622 ISNs are to be sorted.

**Control Block:**

```
Command Code          S9
```

```
Command ID            S901 (a non blank, non zero command ID is
                            required)
```

```
File Number           1  (the  ISN list to be sorted is derived
                          from file 1)
```

```
ISN Quantity          622 (622 ISNs are to be sorted)
ISN Lower Limit       0 (all ISNs are to be selected)
```

```
ISN Buffer Length     2488 (or larger) (each ISN  to be  sorted
                                    requires 4 bytes)
```

```
Command Option 1      H (the Save ISN List option is to be used)
```

```
Command Option 2      b (ascending sequence is to be used)
```

```
Additions 1           ISNbbbbb (the ISN values are  to be  used
                                as the sorting sequence)
```

```
Additions 3           bbbbbbbb (file is not security protected)
```

```
Additions 4           bbbbbbbb  (the ISN  list  to be sorted is
                                 contained in the ISN Buffer)
```

**Buffer Areas:**

```
ISN Buffer            The ISNs to be  sorted  are  provided  in
                      this buffer. Each ISN must be provided as
                      a four byte binary number.
```

**Example 2:**

An ISN list that is stored on the Adabas temporary working space is to be sorted. The command ID under which the ISN list is stored is U066. The list is to be sorted using the descriptors AA and AB as the major and minor sequence fields. The descending option is to be used.

**Control Block:**

```
Command Code          S9
Command ID           S902 (a non blank command ID is required)

File Number          1 (the ISN list was derived from file 1)

ISN Lower Limit      0 (All ISNs are to be selected)

ISN Buffer Length    0 (no ISNs are to be returned in the  ISN
                     Buffer)

Command Option 1     H (the Save ISN List option is to be used)

Command Option 2     D (the descending option is to be used)

Additions 1          AAABbbbb  (AA  is to be used as the major
                                sequence field; AB is to be  used  as the
                                minor sequence field)

Additions 3          bbbbbbbb (file is not security protected)

Additions 4          U066bbbb (the  ISN  list  to be sorted is
                               stored  on the Adabas temporary working
                               space under the command ID U066)
```

# 29 Appendix A - File Definitions (Examples)

The following file definitions are used in all examples in this manual.

In these examples, SL means standard length, and SF means standard format.

File 1:

```
01,GA              ; Group GA, consisting of fields AA and AB.
```

```
02,AA,8,A,DE,NU    ; Elementary field AA; SL is 8 bytes, SF is
                   ; Alphanumeric, Descriptor, Null Value
                   ; Suppression.
```

```
02,AB,2,P,DE,NU    ; Elementary field AB; SL is 2, SF is Packed,
                   ; Descriptor, Null Value Suppression.
```

```
01,AC,20,A,NU      ; Elementary field AC; SL is 20, SF is
                   ; Alphanumeric, Null Value Suppression.
```

```
01,MF,3,A,MU,DE,NU ; Multiple value field MF; SL is 3, SF is
                   ; Alphanumeric, Descriptor, Null Value
                   ; Suppression.
```

```
01,GB,PE            ; Periodic group GB.
```

```
02,BA,1,B,DE,NU     ; Elementary field BA (within periodic group
                    ; GB); SL is 1, SF is Binary, Descriptor, Null
                    ; Value Suppression.
```

```
02,BB,5,P,NU        ; Elementary field BB (within periodic group
                    ; GB); SL is 5, SF is Packed, Null Value
                    ; Suppression.
```

```
02,BC,10,A,NU       ; Elementary field BC (within periodic group
                    ; GB); SL is 10, SF is Alphanumeric, Null Value
                    ; Suppression.
```

```
01,GC,PE            ; Periodic group GC.
```

```
02,CA,7,A,DE,NU     ; Elementary field CA (within periodic group
                    ; GC); SL is 7, SF is Alphanumeric, Descriptor,
                    ; Null Value Suppression.
```

```
02,CB,10,A,MU,NU    ; Multiple value field CB (within periodic group
                    ; GC); SL is 10, SF is Alphanumeric, Null Value
                    ; Suppression.
```

```
01,DA,4,F,NC        ; Elementary field DA; SL is 4; SF is Fixed Point;
                    ; may contain SQL NULL values
```

```
01,W1,0,W           ; Elementary field W1
                    ; SL is variable
                    ; SF is Unicode
```

File 2:

```
01,L1,0,A,NU,NV,NB,LB    ; LOB field
```

```
01,RG                  ; Group RG, consisting of all other fields in the
                       ; record.
```

```
02,RA,8,A,DE,NU        ; Elementary field RA; SL is 8, SF is Alphanumeric,
                       ; Descriptor, Null Value Suppression.
```

```
02,RB,10,A,DE          ; Elementary field RB; SL is 10, SF is
                       ; Alphanumeric, Descriptor.
```

```
02,GX                  ; Group GX, consisting of the fields XA, XB, XC,
                       ; XD, and XE.
```

```
03,XA,10,A             ; Elementary field XA; SL is 10, SF is
                       ; Alphanumeric.
```

```
03,XB,2,P,DE           ; Elementary field XB; SL is 2, SF is Packed,
                       ; Descriptor.
```

```
03,XC,6,U              ; Elementary field XC; SL is 6, SF is Unpacked.
```

```
03,XD,8,A,DE,NU        ; Elementary field XD; SL is 8, SF is
                       ; Alphanumeric, Descriptor, Null Value
                       ; Suppression.
```

```
03,XE,5,A,DE,NU        ; Elementary field XE; SL is 5, SF is
                       ; Alphanumeric, Descriptor, Null Value
                       ; Suppression.
```

```
SA=RA(1,4)             ; Subdescriptor SA; derived from bytes 1 through
                       ; 4 of field RA, format is Alphanumeric.
```

```
SB=RA(1,8),RB(1,4) ; Superdescriptor SB; derived from bytes 1
                   ; through 8 of field RA and bytes 1 through 4 of
                   ; field RB, format is Alphanumeric.
```

```
SC=XB(1,2),XC(1,6) ; Superdescriptor SC; derived from bytes 1
                   ; through 2 of field XB and bytes 1 through 6 of
                   ; field XC, format is Binary.
```

```
SD,B,UQ = XC(1,1),XC(5,6)
                   ; Superdescriptor SD; derived from the first byte
                   ; and the bytes 5 through 6 of the field XC, the
                   ; resulting format is binary, unique descriptor
```

# 30 Appendix B - File Definition For Example Programs

This file definition is used for the following example program.

```
1    ,  A0                                ; Personnel data
  2   ,  AA ,   8 ,    A  , DE,UQ         ; Personnel ID
  2   ,  AB                               ; ID data
   3   ,  AC ,   4 ,    F  , DE           ; Personnel no.
   3   ,  AD ,   8 ,    B  , NU,HF        ; ID card
   3   ,  AE ,   0 ,    A  , LA,NU,NV,NB  ; Signature
 1    ,  B0                               ; Full name
  2   ,  BA ,  40 ,    W  , NU            ; First name
  2   ,  BB ,  40 ,    W  , NU            ; Middle name
  2   ,  BC ,  50 ,    W  , NU ,DE        ; Name
 1    ,  CA ,   1 ,    A  , FI            ; Maritial status
 1    ,  DA ,   1 ,    A  , FI            ; Sex
 1    ,  EA ,   4 ,    P  , DE,NC         ; Birth
 1    ,  F0 ,             PE              ; Private address
  2   ,  FA ,  60 ,    W  , NU,MU         ; Address line
  2   ,  FB ,  40 ,    W  , DE,NU         ; City
  2   ,  FC ,  10 ,    A  , NU            ; Post code
  2   ,  FD ,   3 ,    A  , NU            ; Country
  2   ,  F1                               ; Phone email
   3   , FE ,   6 ,    A  , NU            ; Area code
   3  ,  FF,   15 ,    A  , NU            ; Private phone
   3  ,  FG ,  15 ,    A  , NU            ; Private fax
   3  ,  FH ,  15 ,    A  , NU            ; Private mobile
   3  ,  FI ,  80 ,    A  , NU,MU,DE      ; Private email
 1    ,  I0 ,            PE               ; Business address
  2   ,  IA ,  40 ,    W  , NU,MU         ; Address line
  2   ,  IB ,  40 ,    W  , DE,NU         ; City
  2   ,  IC ,  10 ,    A  , NU            ; Post code
  2   ,  ID ,   3 ,    A  , NU            ; Country
  2   ,  IE ,   5 ,    A  , NU            ; Room number
  2   ,  I1                               ; Phone email
   3  ,  IF ,   6 ,    A  , NU            ; Area code
   3  ,  IG ,  15 ,    A  , NU            ; Business phone
   3  ,  IH ,  15 ,    A  , NU            ; Business fax
   3  ,  II ,  15 ,    A  , NU            ; Business mobile
   3  ,  IJ ,  80 ,    A  , NU,MU,DE      ; Business email
 1    ,  JA ,   6 ,    A  , DE            ; Department
 1    ,  KA ,  66 ,    W  , DE,NU         ; Job title
 1    ,  LO               , PE            ; Income
  2   ,  LA ,   3 ,    A  , NU            ; Currency code
  2   ,  LB ,   6 ,    P  , NU            ; Salary
  2   ,  LC ,   6 ,    P  , NU,MU,DE      ; Bonus
 1    ,  MA ,   4 ,    G  , NU            ; Total income
 1    ,  NO                               ; Leave data
  2   ,  NA ,   2 ,    U                  ; Leave due
  2   ,  NB ,   3 ,    U  , NU            ; Leave taken
 1    ,  OO               , PE            ; Leave booked
  2   ,  OA ,   8 ,    U  , NU            ; Leave start
  2   ,  OB,    8 ,    U  , NU            ; Leave end
 1    ,  PA ,   3 ,    A  , DE,NU,MU      ; Language
 1    ,  QA ,   7 ,    P                  ; Last update
 1    ,  RA ,   0 ,    A  , LB,NU,NV,NB   ; Picture
 1    ,  SO ,     PE                      ; Documents
  2   ,  SA ,  80 ,    W  , NU            ; Document description
  2   ,  SB ,   3 ,    A  , NU            ; Document type
  2   ,  SC ,   0 ,    A  , LB,NU,NV,NB,MU ; Document

CN,HE=COLLATING(BC,'de__PHONEBOOK',PRIMARY)
H1=NA(1,2),NB(1,3)
S1=JA(1,2)
S2=JA(1,6),BC(1,40)
S3=LA(1,3),LB(1,6)
```

# 31 Appendix C - C Examples

This Appendix contains examples in C using direct Adabas calls.

## C Example

The Adabas file defined in Appendix B is used in this example.

```
/************* ADABAS  ************* (C) Copyright Software AG 2005
*
* File       : c_example.c
* Description: Example for ADABAS calls from C programs
*
*
*********************************************************************/



#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <adabasx.h>                /* include adabas definitions */



#define NULLPTR ((char *) 0)       /* define for NULL-pointer */

int dbid;                          /* default database */
int emp_file ;                     /* filenumber       */


#define FMTBUF  "LB1,4,F."         /* FB: get first salary field */
#define FBLEN   8                  /* format buffer's length    */


#define SEABUF  "BC,5."            /* SB: search NAME equal to  */
#define SBLEN   5                  /* search buffer's length    */


#define VALBUF  "SMITH"            /* VB: name to search for    */
#define VBLEN   5                  /* value buffer's length     */


#define RBLEN   4                  /* record buffer's length    */
#define IBLEN   4                  /* ISN buffer's length       */


char openrb[100];                    /* record buffer used for OPEN */

/*
```

```
        +-----------------+
        ! define buffers !
        +-----------------+
*/

static CB_PAR  cb;          /* control block */

static int     isn_buffer;      /* ISN buffer (1 ISN) */

static int     rb_salary;           /* record buffer (for salary field) */

/*************************************************************************/
/**                 Module Local Functions                             **/
/*************************************************************************/

int      usage ();
int      open_database ();
int      close_database();
void     response ();
int      update_record();
int      find_record();
int      issue_bt ();



usage()
{
    printf("usage: c_example <dbid> <employees file number>\n");
    exit(1);
}


/*
        +------------------+
        !   Main program   !
        +------------------+
*/

int main (int argc, char **argv )
{
   register int   upd = 0;           /* update counter */
   register int   old_salary;            /* saving salary field */


   printf ("\nSoftware AG - C example program for calling ADABAS\n\n") ;

   if (argc==3)
   {
       if (sscanf (argv[1],"%d",&dbid)==0)
       usage ();
       if (sscanf (argv[2],"%d",&emp_file)==0)
       usage ();
   }
   else
       usage();

   /*--------------*/
   /* Open session */
   /*--------------*/
   if ( open_database() != ADA_NORMAL )
      response ();              /* open command failed */
   else
   {
      /*------------------------------------*/
      /* Search all persons with name = SMITH */
      /*------------------------------------*/
      if ( find_record () == ADA_NORMAL )
      {
         printf ("Found  %ld records with name %s, increase salary by 10 %%\n",
         cb.cb_isn_quantity, VALBUF);

         while ( cb.cb_return_code == ADA_NORMAL  &&  cb.cb_isn_quantity != 0 )
         {
            old_salary = rb_salary;
            rb_salary += rb_salary / 10;

        /*------------------------*/
        /* Increase salary by 10 % */
        /*------------------------*/
            if ( update_record () != ADA_NORMAL )
               cb.cb_isn_quantity = 0;
            else
            {
               upd++;
               printf
                  ("%3d. ISN = %8d  old salary = %10ld   new salary = %10ld\n",
                    upd, cb.cb_isn, old_salary, rb_salary);

          /*-----------------*/
          /* Get next record */
          /*-----------------*/
               find_record ();
            }
         }
      }

      if ( cb.cb_return_code != ADA_NORMAL )    /* if response given: */
      {
         response ();               /* display response code */
         cb.cb_return_code = ADA_NORMAL;

         if ( upd != 0 )            /* if updates done */
     {
            if ( issue_bt () == ADA_NORMAL )    /* backout transaction */
```

```
               upd = 0;
            else
               response ();         /* BT failed */
     }
      }

      if ( close_database () != ADA_NORMAL )
         response ();
   }
}


/*
        +------------------------+
        ! Open the database for  !
        ! updating the EMPLOYEES !
        ! file                   !
        +------------------------+
*/

int open_database ()
{


   cb.cb_cmd_code[0] = 'O';
   cb.cb_cmd_code[1] = 'P';

   sprintf(openrb,"UPD=%d.",emp_file);

   cb.cb_rec_buf_lng = strlen(openrb);
   do
   {
      CB_SET_FD(&cb,dbid,0);
      adabas ( &cb , NULLPTR , openrb, NULLPTR, NULLPTR, NULLPTR );
   }
   while ( cb.cb_return_code == ADA_TABT );

   cb.cb_isn_quantity = 0;
   cb.cb_isn_ll = 0;

   return ( cb.cb_return_code );
}


/*
        +--------------------------------+
        !  Close the database and give   !
        !  implicit end of transaction.  !
        +--------------------------------+
*/

int close_database ()
```

```
{
   CB_SET_FD(&cb,dbid,0);

   cb.cb_cmd_code[0] = 'C';
   cb.cb_cmd_code[1] = 'L';

   adabas ( &cb, NULLPTR, NULLPTR, NULLPTR, NULLPTR, NULLPTR );

   return ( cb.cb_return_code );
}


/*
        +---------------------------------+
        ! Call ADABAS to find all people !
        ! named SMITH and read them      !
        +---------------------------------+
*/

int find_record ()
{

   CB_SET_FD(&cb,dbid,emp_file);

   if ( cb.cb_isn_quantity == 0 )       /* if first call:        */
   {                         /*   search records and */
      cb.cb_cmd_code[0]  = 'S';         /*   read first one      */
      cb.cb_cmd_code[1]  = '4';
      cb.cb_cmd_id[0]    = 'F';
      cb.cb_cmd_id[1]    = 'I';
      cb.cb_cmd_id[2]    = 'N';
      cb.cb_cmd_id[3]    = 'D';

      cb.cb_fmt_buf_lng  = FBLEN;
      cb.cb_rec_buf_lng  = RBLEN;
      cb.cb_sea_buf_lng  = SBLEN;
      cb.cb_val_buf_lng  = VBLEN;
      cb.cb_isn_buf_lng  = IBLEN;       /* read 1. record by search */
   }
   else                      /* if subsequent call: */
   {                         /*   read next record  */
      cb.cb_cmd_code[0] = 'L';
      cb.cb_cmd_code[1] = '4';
      cb.cb_cop2         = ADA_GET_NEXT;    /* activate get-next option */
   }

   adabas ( &cb , FMTBUF , &rb_salary , SEABUF , VALBUF , &isn_buffer );

   if ( cb.cb_return_code == ADA_EOF )
   {
      cb.cb_return_code   = ADA_NORMAL;     /* indicate success */
      cb.cb_isn_quantity  = 0;          /* force termination of loop */
   }
   return ( cb.cb_return_code );
}




/*
        +---------------------------+
        ! Change the value of salary !
        !  and update the record     !
        +---------------------------+
*/

int update_record ()
{
   CB_SET_FD(&cb,dbid,emp_file);

   cb.cb_cmd_code[0] = 'A';
   cb.cb_cmd_code[1] = '1';
   cb.cb_cop1 = cb.cb_cop2 = ' ';

   adabas ( &cb , FMTBUF , &rb_salary, NULLPTR, NULLPTR, NULLPTR );

   return ( cb.cb_return_code );
}




/*
        +----------------------+
        !  printf response code  !
        +----------------------+
*/

void response ()
{
   printf ("** Response code %d from ADABAS for Command %-2.2s\n",
            cb.cb_return_code , cb.cb_cmd_code);
   printf ("** Additions2 %d %d\n",
            cb.cb_add2[2] , cb.cb_add2[3]);

}




/*
        +--------------------+
        !  Issue BT command  !
        +--------------------+
*/

int issue_bt ()
```

```
{
   CB_SET_FD(&cb,dbid,0);

   cb.cb_cmd_code[0] = 'B';
   cb.cb_cmd_code[1] = 'T';
   cb.cb_cop1 = cb.cb_cop2 = ' ';

   adabas ( &cb, NULLPTR, NULLPTR, NULLPTR, NULLPTR, NULLPTR );

   return ( cb.cb_return_code );
}
```

## C Example for LOB Processing

This example works with any file that contains an elementary LOB field; the database ID, file number and name of the LOB field can be passed to the program as parameters.

```
/************* ADABAS  ************* (C) Copyright Software AG 2006 **********
*
* File       : lob_example.c
*
* Description: Example for ADABAS calls from C programs
*
*          This is an example how read and write large objects in a database
*          or to read an object from a database
*
*          Parameters:
*           function (add_lob or read_lob)
*           Database-ID
*           Filenr
*           Isn
*           fieldname
*           filename
*
*           The add_lob function reads a disk file and adds or updates the
*           content of a specified field in a database. This is done by an
*           ADABAS A1 call.
*           The record must exist and the given field is updated.
*
*           The read_lob function does two L1 calls to retrieve the size and
*           the content of a given field in a database. The content is then
*           written to a disk file.
*
*
*           adabasx()is required for buffers greater than 64 K.
*           The buffers are described by ABD structures (ADABAS Buffer
*           descritpions) that are passed as arguments to adabasx.
*           More than one pair of ABDs for record and format buffer can be used.
*           (See function add_lob)
*           This means that not all data has to be in one continuous memory
*           area.The ABDs are initialized by the SETABD macros.
*
*
*****************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include "adabasx.h"        /* include adabas definitions */


/* opening the database */
void open_database( int dbid );

/*close the database */
void close_database( int dbid );

/*usage descriptions */
void usage( );

/* display response code */
void response( ACBX *cbx );

/* read from database */
void read_lob( int dbid, int filenr, int isn, char *filename, char *field );

/* write in the database */
void add_lob( int dbid, int filenr, int isn, char *filename, char *field );

/* read from the file */
int  read_file( char *filename, void **obj);

/* write LOB to file */
void write_file( char *filename, char *obj, unsigned int objsize  );


int main ( int argc, char **argv )
/*
    Main function
    read the argument and pass control to one of the functions
*/

{
    int filenr, dbid, isn;    /* variables for adabas call        */
    char * command = NULL;      /* command to execute (read/write)*/
    char * filename = NULL;     /* filename                       */
    char *field;          /* field of the database                */
    printf("\nSoftware AG - C example program for calling ADABAS with large");
    printf(" objects\n\n    add/read a large object to/from a record \n\n");

    if( argc != 7 )
        usage( );
    else
    {
        command = &argv[1][0];

        if( ( strcmp( command ,"add_lob") != 0 )
              && ( strcmp( command, "read_lob") != 0 ) )
        {
            usage ( );
        }

        if( sscanf( argv[2],"%d",&dbid ) == 0 )
            usage( );

        if( sscanf (argv[3],"%d",&filenr ) == 0 )
            usage( );

        if( sscanf (argv[4],"%d",&isn ) == 0 )
            usage( );

        field = argv[5];


        filename = argv[6];
    }

    open_database( dbid );      /* Open database            */

                    /* read in the database         */
    if( strcmp( command, "add_lob" ) == 0 )
        add_lob( dbid, filenr, isn, filename, field );
    else                /* Write from database          */
        read_lob( dbid, filenr, isn, filename, field );

    close_database( dbid );     /* Close database           */

}



void usage( )
/*------------------------------------------
    Display how the program has to be used
------------------------------------------*/
{
printf("usage:\n");
printf("    lob_example (add_lob | read_lob) <dbid> <file> <isn> <field> <file ↩
name>");
```

```
printf("\n\n");
printf("    add_lob  : Read an object from <file name> and update <field> in ↩
record\n");
printf("    read_lob : Read from database, save to file on disk\n");
printf("    dbid     : Number of ADABAS database\n");
printf("    file     : Number of file\n");
printf("    isn      : ISN of record to be updated/read\n");
printf("    field    : Field where object is (to be) stored\n");
printf("    file name: Name of the disk file to be read or written\n\n");
printf("    ie.: lob_example add_lob 12 9 1100 RA picture.jpg \n");
printf("    ie.: lob_example read_lob 12 9 1100 RA picture.jpg\n");
    exit(1);
}




void open_database( int dbid )

/*--------------------------------------------------
  Open the database

  This call is done with conventional adabas
  function.
  All other calls are done via adabasx to show that
  The two types of calls can be mixed.
--------------------------------------------------*/

{

    CB_PAR cb;                          /* Control block        */
    char *open_rb = ".";

    memset(&cb,0,sizeof(cb));

    cb.cb_call_type=0;

    cb.cb_cmd_code[0] = 'O';            /* open options         */
    cb.cb_cmd_code[1] = 'P';
    CB_SET_FD(&cb,dbid,0);
    adabas( &cb, 0, open_rb , 0 ,0 ,0); /* adabas call          */


    if( cb.cb_return_code != 0 )
    {
        printf("Open of Database %d failed with response %d\n",
        dbid,
        cb.cb_return_code );
        exit(1);
    }
}
```

```
void read_lob( int dbid, int filenr, int isn, char *filename, char *field )
/*--------------------------------------------+
  Read a large object from database
  two L1 calls are done:
  the first to determine the size of the object.
  then a buffer with this size is allocated.
  the second reads the actual object.
+--------------------------------------------*/
{
#define abdcount 4
    char i=0;
    ACBX cbx;                  /* Control block                   */
    ABD  object_fb_abd;        /* Format buffer ABD for the object */
    ABD  object_rb_abd;        /* Record buffer ABD for the object */
    ABD  object_size_fb_abd;   /* Format buffer ABD for the size   */
    ABD  object_size_rb_abd;   /* Record buffer ABD for the size   */

    ABD* pabd[abdcount];       /* ABD array for Adabas call        */

    char size_fb[20];          /* format buffer for size           */
    char object_fb[30];        /*format buffer to retrieve the     */
                               /*object itself                     */
    unsigned int object_size;  /*The 4-Byte integer object_size    */
                               /*is our record buffer              */
                               /*for the first call                */
    unsigned int object_size_chk;/*store the size to check if object*/
                               /*changed between the calls         */



    SETACBX ( &cbx );

    cbx.acbxdbid   = dbid;
    cbx.acbxfnr    = filenr;
    cbx.acbxisn    = isn;
    cbx.acbxcmd[0] = 'L';      /* Command code         */
    cbx.acbxcmd[1] = '1';

    /*Initialize the abds*/
    pabd[0] = &object_size_fb_abd;
    pabd[1] = &object_size_rb_abd;
    pabd[2] = &object_fb_abd;
    pabd[3] = &object_rb_abd;

    /*Initialize abd*/
    for (i=0 ;i< abdcount;i++)
    {
        SETABD  ( pabd[i] );
        pabd[i]->abdloc = ABDQIND;
    /*  ABDQIND means that the buffer is found at abdaddr   */
    }
    object_size_fb_abd.abdid    = ABDQFB;
    sprintf( size_fb, "%sL.", field );
    object_size_fb_abd.abdaddr  = size_fb;
    object_size_fb_abd.abdsend  =
    object_size_fb_abd.abdsize  = strlen( size_fb );


    object_size_rb_abd.abdid    = ABDQRB;
    object_size_rb_abd.abdaddr  = &object_size;
    object_size_rb_abd.abdsize  = sizeof(object_size);

    adabasx ( &cbx, 2 , pabd );
    response( &cbx );

    printf("Size of Object in Database :%d     \n\n",object_size);
    if (object_size)
    {
        object_fb_abd.abdid    = ABDQFB;

        sprintf( object_fb, "%s,*.", field );
        object_fb_abd.abdaddr  = object_fb;
        object_fb_abd.abdsend = object_fb_abd.abdsize  = strlen(object_fb);

        object_rb_abd.abdid    = ABDQRB;


        do {
            object_size_chk=object_size;

            /*Now the memory for the object is allocated                */
            /*and the object itself is read by a second call            */
            /*the lenth is read again to verify that it is unchanged    */
            object_rb_abd.abdsize  = object_size_chk;
            if (object_rb_abd.abdaddr)
                free(object_rb_abd.abdaddr);
            object_rb_abd.abdaddr  = malloc ( object_size_chk);
            if (object_rb_abd.abdaddr)
            {
                adabasx ( &cbx , 4 , pabd );    /* Adabas call  */
                response( &cbx );

            }
            else
            {
                printf ("*** Memory allocation for Record Buffer failed");
                close_database(dbid);
                exit (-1);
            }
          } while (object_size>object_size_chk&&object_size);
    }
    if (object_size)
    {
write_file( filename, object_rb_abd.abdaddr, (unsigned int)object_rb_abd.abdsize );
printf("Object from field %s ISN %d file %d Database %d successfully written\nto ↩
file %s\n\n",
field,isn,filenr,dbid,filename);
    }
    else
    {
printf("No Object found in field %s ISN %d file %d Database %d\n",
field,isn,filenr,dbid);
printf("No file written\n\n");
    }

}



void write_file( char *filename, char *obj, unsigned int objsize  )
/*---------------------------------------------
    write LOB to disc file
----------------------------------------------*/
{
    FILE * fd;
    struct stat st;        /* to specify the exists of the file */

    if(stat(filename , &st) > 0)
    {
        printf("The file %s already exists!\n\n", filename);
        exit( 1 );
    }
                    /* open the file            */
    if( ( fd = fopen( filename, "wb+" ) ) == NULL )
    {
        printf("File %s could not be opened for writing!\n\n", filename);
        exit( 1 );
    }
                    /* write in the file        */
    else if ( fwrite ( obj, 1, objsize, fd ) < 0 )
    {
        printf("Cannot write to file %s\n\n", filename);
        exit( 1 );
    }
    fclose(fd);
}



void add_lob( int dbid, int filenr, int isn, char *filename, char *field)
/*------------------------------------------------------------
    add a large object to a record in a  database.
    for demonstration purposes two pairs of record and format
    buffer segments are used:
```

```
    one pair is used to specify the size of the object
    one pair is used for the object itself
 ----------------------------------------------------------*/
{
#define abdcount 4
    char i=0;
    ACBX cbx;                  /* Control block                    */
    ABD  object_fb_abd;        /* Format buffer ABD for the object */
    ABD  object_rb_abd;        /* Record buffer ABD for the object */
    ABD  object_size_fb_abd;   /* Format buffer ABD for the size   */
    ABD  object_size_rb_abd;   /* Record buffer ABD for the size   */

    ABD* pabd[abdcount];       /* ABD array for Adabas call        */

    char size_fb[20];          /* format buffer for size           */
    char object_fb[20];        /* format buffer for content        */
    int object_size;


    SETACBX ( &cbx );          /* Initialing abds                  */
    cbx.acbxdbid = dbid;
    cbx.acbxfnr  = filenr;
    cbx.acbxisn  = isn;
    cbx.acbxcmd[0] = 'A';      /* Record Update                    */
    cbx.acbxcmd[1] = '1';
    cbx.acbxcop1   = 'H';      /* Hold status database             */


    pabd[0] = &object_size_fb_abd;
    pabd[1] = &object_fb_abd;
    pabd[2] = &object_size_rb_abd;
    pabd[3] = &object_rb_abd;

    /*Initialize abd*/
    for (i=0 ;i< abdcount;i++)
    {
        SETABD  ( pabd[i] );
        pabd[i]->abdloc = ABDQIND;
    /*  ABDQIND means that the buffer is found at abdaddr          */
    }

    /*the object is stored in a pair of rb and fb                  */
    object_fb_abd.abdid     = ABDQFB;
    object_fb_abd.abdaddr   = object_fb;
    sprintf (object_fb,"%s,*.",field);
    object_fb_abd.abdsend = object_fb_abd.abdsize = strlen(object_fb);
    strlen(object_fb);

    object_rb_abd.abdid    = ABDQRB;
    /*store the size of the object in the abdsize field            */
    /*this field is also used as the second record buffer segment*/
    object_size = (int)read_file( filename, &object_rb_abd.abdaddr);
    object_rb_abd.abdsend = object_rb_abd.abdsize = object_size ;

    printf("** Size of Object to add:%d     \n\n",object_rb_abd.abdsize);


    /*the size of the object is stored in a separate pair           */

    object_size_fb_abd.abdid  = ABDQFB;
    object_size_fb_abd.abdaddr=size_fb;
    sprintf (size_fb,"%sL.",field);
    object_size_fb_abd.abdsend = object_size_fb_abd.abdsize
        = strlen(size_fb);

    object_size_rb_abd.abdid = ABDQRB;
    object_size_rb_abd.abdsend = object_size_rb_abd.abdsize
        = sizeof(object_size);

    object_size_rb_abd.abdaddr=&object_size;

    adabasx ( &cbx , abdcount,  pabd );    /* Adabas call           */

    response( &cbx );

printf("Object successfully added/updated in field %s ISN %d file %d Database %d\n\n",
 field,isn,filenr,dbid);

#undef abdcount
}




int  read_file( char *filename, void **obj)
/*--------------------------------------
 allocate memory for the record buffer and
 read a file into it
+--------------------------------------*/
{
    unsigned int filelength;
    FILE * fd;
    struct stat st;

    /*  determine the length of  file    */

    if( stat(filename , &st) < 0 )
    {
        printf("The file %s doesn't exist!\n\n", filename);
        exit (1);
    }
    if( ( filelength = st.st_size ) < 0 )
    {
        printf("File % could not be opened!\n\n", filename);
        exit( 1 );
    }

                    /* open the file            */
    if( !filename || ( fd = fopen( filename, "rb" ) ) == NULL )
    {
        printf("File % could not be opened!\n\n", filename);
        exit( 1 );
    }


    if (!(*obj=malloc (filelength)))
    {
        printf("Could not allocate memory\n\n");
        exit( 1 );
    }

    filelength=fread( *obj, sizeof(char), filelength / sizeof(char), fd );

    fclose ( fd );

    if( filelength < 0 )
    {
        printf("Could not read  from the file: %s\n\n", filename);
        exit( 1 );
    }

    return  filelength;
}


void close_database( int dbid )
/*---------------------------+
  Close the session and give
  implicit end of transaction.
+---------------------------*/

{
    ACBX cbx;               /* Control block            */

    SETACBX( &cbx );        /* Initialize the Control      Block*/
    cbx.acbxcmd[0] = 'C';       /* put close option         */
    cbx.acbxcmd[1] = 'L';
    cbx.acbxdbid = dbid;

    adabasx( &cbx , 0 , NULL);      /* adabas call              */

    response( &cbx );
    if( cbx.acbxrsp != 0 )
         printf("** Close failed with response %d\n\n", cbx.acbxrsp);
}






void response( ACBX *cbx )
/*-----------------+
print response code
+------------------*/

{
    if( cbx->acbxrsp != 0)
    {
        printf("** Response code from ADABAS for Command %-2.2s:     %d\n\n",
        cbx->acbxcmd, cbx->acbxrsp);
        if( cbx->acbxrsp == ADA_LOBERR)
        {
            printf("** subcommand response %d\n\n",cbx->acbxerrc);
        }
        if ( cbx->acbxrsp != ADA_ANACT)
        {
                close_database(cbx->acbxdbid);
        }
        exit( 1 );
    }

}
```

# 32 Appendix D - Example Files in the Adabas Kit

The Adabas kit contains the following example files:

| File Name | Description |
|---|---|
| $ACLDIR/examples/bin/c_example (UNIX)<br>%ACLDIR%\..\examples\client\c_example.exe (Windows) | Executable of the C example program |
| $ACLDIR/examples/bin32/c_example (UNIX, only on 64 bit systems where 32 bit mode is supported)<br>%ACLDIR%\..\examples\bin32\c_example.exe (Windows, only on 64 bit systems where 32 bit mode is supported) | Executable of the C example program in 32 bit mode |
| $ACLDIR/examples/src/c_example.c (UNIX)<br>%ACLDIR%\..\examples\src\c_example.c (Windows) | C example program of Appendix C |
| $ACLDIR/examples/src/c_example_w.c (UNIX)<br>%ACLDIR%\..\examples\src\c_example_w.c (Windows) | C example program using wide character sets |
| $ACLDIR/examples/bin/lob_example (UNIX)<br>%ACLDIR%\..\examples\client\lob_example.exe (Windows) | Executable of the C example program for LOB processing |
| $ACLDIR/examples/bin32/lob_example (UNIX, only on 64 bit UNIX systems where 32 bit mode is supported)<br>%ACLDIR%\..\examples\bin32\lob_example.exe (Windows, only on 64 bit systems where 32 bit mode is supported) | Executable of the C example program for LOB processing in 32 bit mode |
| $ACLDIR/examples/src/lob_example.c (UNIX)<br>%ACLDIR%\..\examples\src\lob_example.c (Windows) | C example program for LOB processing of Appendix C |
| $ACLDIR/examples/src/makefile (UNIX)<br>%ACLDIR%\..\examples\src\makefile (Windows) | Make file for C example program - for usage, see the comments in the make file |
| $ACLDIR/examples/src/security_example.c (UNIX)<br>%ACLDIR%\..\examples\src\security_example.c (Windows) | C example program using Adabas authentication |

**Note:** The C examples use the Employees file, which is one of the demo Adabas files delivered with the Adabas kit. For more information on the demo files see Utilities, Appendix A.