Adabas

Concepts and Facilities

Version 8.6.2

October 2026

ADABAS & NATURAL

This document applies to Adabas Version 8.6.2 and all subsequent releases.

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

Copyright © 1971-2026 Software GmbH, Darmstadt, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software GmbH product names are either trademarks or registered trademarks of Software GmbH and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software GmbH and/or its subsidiaries is located at https://softwareag.com/licenses.

Use of this software is subject to adherence to Software GmbH's licensing conditions and terms. These terms are part of the product documentation, located at https://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software GmbH Products / Copyright and Trademark Notices of Software GmbH Products". These documents are part of the product documentation, located at https://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software GmbH.

Document ID: ADAMF-CONCEPTS-862-20261013

Table of Contents

Preface v
1 About this Documentation 1
- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

2 Adabas Is 5
- Operational Highlights 6
- Operating Environments 7
- Supported Data Models 7
- Operating Structure 8
- Running Adabas 12

3 Adabas Design 15
- Adabas Entities 16
- Database Components 17
- Database Files 26
- Record and Field Definitions 30
- Parallel Participant Table (PPT) 52
- Spanned Records 54

4 Using Adabas 59
- Accessing a Database from Programs 60
- Maintaining Database Integrity 68

5 Adabas Utilities 77
- Initial Design and Load Operations 78
- Backup / Restore / Recovery Routines 81
- Database Modification Routines 84
- Audit / Control / Tuning/Reporting Procedures 92

6 Licensing Adabas 97
7 Adabas Security 99
- Security System User IDs 100
- Data Encryption 103
- Multiclient Files 103
- Adabas Security and ADASCR 104
- Adabas SAF Security 105
- Related Security Options 109

8 Optional Product Extensions 111
- Adabas Bridges 112
- Adabas Caching Facility 116
- Adabas Cluster Services 117
- Adabas Delta Save Facility 120
- Adabas Encryption 121
- Adabas Fastpath 121
- Adabas Native SQL 122
- Adabas Online System 123

Adabas Parallel Services 125
Adabas Review 125
Adabas SAF Security 128
Adabas SQL Gateway 129
Adabas Statistics Facility 130
Adabas Text Retrieval 132
Adabas Transaction Manager 133
Adabas Vista 134
Event Replicator for Adabas 135
Entire Net-Work Multisystem Processing Tool 139
Natural Application Development Environment 141
Predict Data Dictionary System 142
Index 143

Preface

This documentation provides a technical introduction to the principles and functions of Adabas, Software GmbH's adaptable data base management system. It provides an overview of Adabas for those who require a basic understanding of Adabas operating environments, design, use, and optional extensions.

Adabas is....	Presents a system overview and discusses the supported operating environments.
Adabas Design	Describes the structures used by Adabas and their functions and interactions.
Using Adabas	Describes accessing the database and maintaining its integrity.
Adabas Utilities	Briefly describes the Adabas utilities.
Licensing Adabas	Describes the Adabas licensing concept.
Adabas Security	Describes the security features available with Adabas.
Optional Product Extensions	Briefly describes the optional product extensions available from Software GmbH to add functionality to the Adabas core product.

1 About this Documentation

- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

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: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. 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

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

Product Training

You can find helpful product training material on our Learning Portal at **https://learn.softwareag.com**.

Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software GmbH news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software GmbH resources.

Product Support

Support for Software GmbH products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

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.

2 Adabas Is . . .

- Operational Highlights 6
- Operating Environments 7
- Supported Data Models 7
- Operating Structure 8
- Running Adabas 12

Adabas, the adaptable data base, is a high-performance, multithreaded, database management system for mainframe platforms where database performance is a critical factor. It is interoperable, scalable, and portable across multiple, heterogeneous platforms including mainframe, midrange, and PC.

Operational Highlights

High Availability

Adabas is designed for operation 7 days a week and 24 hours a day. Space is managed dynamically (read ***Adabas Space Management***, elsewhere in this guide), files can be loaded and unloaded, backed up and restored, and system performance can be analyzed without interrupting the active database.

Storage Space Optimization

Adabas stores data in compressed form to reduce space requirements. Since modern databases are measured in gigabytes (1000 megabytes) or even terabytes (1000 gigabytes), the savings in disk space can be considerable. Reduced space requirements also mean that the input/output (I/O) system is more efficient.

Performance

Performance is the key factor of Adabas, which includes a number of features to enhance it. For instance, a number of set-up parameters are available for fine-tuning the database operating environment, and many of these can be modified while the database is active.

Fault Tolerance

Adabas recovers automatically after an abnormal database or system termination. Each time an Adabas database is started, an automatic check is initiated to determine whether the database previously terminated cleanly or an active transaction was interrupted. If a transaction was interrupted, Adabas automatically resets all changes of the uncompleted transaction so that the database is consistent.

Operating Environments

Adabas supports a variety of operating environments and can be used in distributed environments. To see a complete list of the operating platforms supported by Adabas, read *Supported Platforms*, in the *Adabas Release Notes*. For general information regarding Software GmbH product compatibility with other platforms and their requirements for Software AG products, visit Software GmbH's ***Hardware Supported*** web page; for specific information regarding Software GmbH product compatibility with IBM platforms and any IBM requirements for Software GmbH products, visit Software GmbH's ***IBM Supported Platforms*** web page.

As the telecommunication interface, mainframe Adabas supports TP monitors such as Software GmbH's Com-plete as well as other popular monitors such as TSO, CICS, and IMS/DC.

Software GmbH's multisystem processing tool Entire Net-Work provides the benefits of distributed processing by allowing you to communicate across a network with Adabas and other service tasks. Support for network access methods is implemented in the form of line drivers. Mainframe Entire Net-Work provides drivers for VTAM, CTCA and FCTC (channel-to-channel adapters), TCP/IP, and XCF.

Supported Data Models

Adabas is a relational-like database in that:

- it stores information in tables in which rows represent individual data *records* and columns represent *fields*; and
- separate Adabas files can be linked logically by a common field.

Adabas differs from true relational databases in that it:

- stores many data relationships physically, resulting in fewer demands on CPU resources than true relational databases, which create all relationships logically at runtime.
- supports repeating groups of fields.

Adabas separates data relationships, management, and retrieval from the actual physical data and stores the physical data independently. It provides flexible access techniques and performs both simple and complex searches quickly and efficiently. The independence of the data from the program minimizes the need to reprogram when the database structure changes.

Logical data relationships can be created as needed. Adabas can accommodate any representational and access requirements dictated by the user environment. Each individual corporate user can decide how to view data in the system, and can alter data relationships dynamically -without altering the database or existing programs.

In contrast to systems that require a single model for all data, Adabas allows you to choose any structure your application requires. You can access the same data using your choice of data-model perspective:

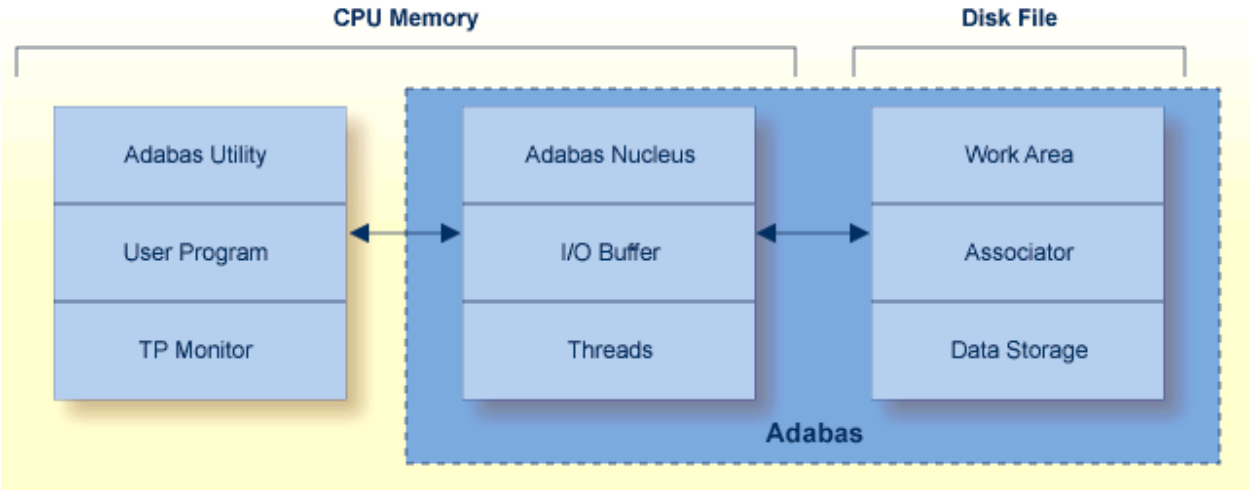
- relational including nested relational (tables within tables)
- entity relationship, with proven ability to support structural objects
- hierarchical; network
- geographical
- text

These data models can be combined within a single business solution; multiple solutions can view Adabas data using different data models.

As new requirements develop, Adabas evolves in both scope and complexity without redesign of the database or reprogramming of application systems. For example, field and access keys may be added to an Adabas file at any time without reloading or reorganizing the file.

Operating Structure

The following figure shows the operating structure of the Adabas system.



- Nucleus, I/O Buffer, and Threads
- Data Storage, Associator, and Work

- Utilities, User Programs, and TP Monitors

Nucleus, I/O Buffer, and Threads

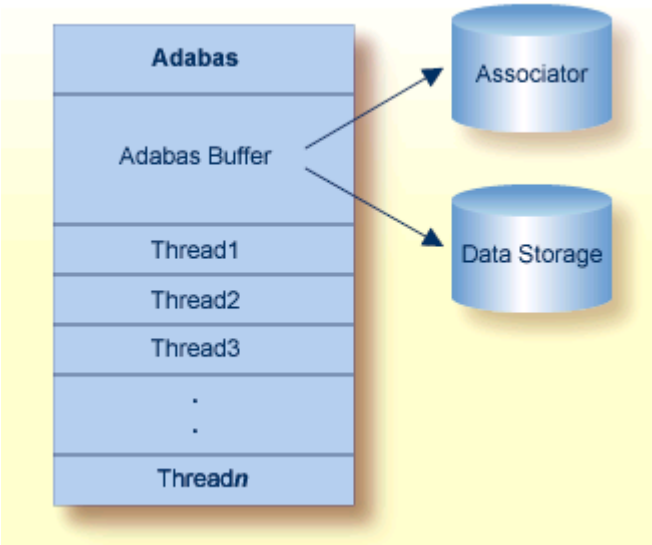
The Adabas nucleus and input/output (I/O) buffer are loaded into main memory at startup. The *nucleus* is a set of programs that drives Adabas, coordinates all work, and translates user program statements into Adabas commands. All programs access Adabas files through the nucleus. All database activities such as data access and update are managed by the Adabas nucleus. In most cases, a single nucleus is used to manage a single physical database.

Note: For information about running multiple nuclei against a single physical database under a single operating system image, read ***Adabas Parallel Services***, elsewhere in this guide or under multiple z/OS images, read ***Adabas Cluster Services***, also in this guide.

The Adabas I/O buffer area, which can be resized for each Adabas session, contains the most frequently used data and data relationships; it helps to minimize physical input/output (I/O) activity and thus saves computer time. It contains blocks read from the database and blocks to be written to the database:

- For blocks read from the database, a buffer algorithm ensures that the most frequently accessed blocks stay in memory. When a block from the database is needed, the buffer content is checked to determine if the block is already in memory, thus avoiding unnecessary reads.
- Multiple updates are accumulated in a block before being written (flushed) to the database.

Adabas provides multithreaded processing to maximize throughput. If I/O activity suspends command processing in an active thread, Adabas automatically switches to another thread. The user may set the number of 8-kilobyte threads to be used for an Adabas session up to a maximum of 250.



Data Storage, Associator, and Work

The Data Storage, Associator, and Work components are physical disk areas:

- Data Storage contains raw data, generally in compressed form.
- The Associator contains information about data relationships.
- The Work area contains the data protection area and temporary storage for intermediate results during complex search operations or distributed transaction processing.

Read ***Adabas Design***, elsewhere in this guide for more information about these database components.

Utilities, User Programs, and TP Monitors

Utilities

Database services such as loading or deleting files are handled by an integrated set of online and batch-mode utility programs. Most utilities can be run in parallel with normal database activity to preclude interruption of daily production.

Adabas utilities provide initial design and load operations, backup/restore/recovery routines, database modification routines, and audit/control/tuning procedures. Read ***Adabas Utilities***, elsewhere in this guide, for a brief explanation of each utility.

User Programs

The nucleus is called using a batch or online user program written in:

- Natural, Software GmbH's fourth-generation application development environment, or some other fourth-generation language
- Assembler, or a third-generation programming language such as FORTRAN, COBOL, or PL/I that uses the powerful and flexible Adabas direct call interface. Each Adabas call is accompanied by a parameter list identifying buffers defined in the user program that are used to transfer information to and from Adabas.

Note: Read ***Adabas Native SQL***, elsewhere in this guide, for information about Adabas Native SQL, a precompiler for Ada, COBOL, FORTRAN, and PL/I programs. Read ***Adabas SQL Gateway***, elsewhere in this guide, for information about Adabas SQL Gateway, an SQL interface to Adabas.

Data from VSAM, DL/I, or IMS/DB database structures can be transferred to and stored in Adabas using Adabas bridge products. The original, unmodified programs continue operating with their original data access commands while the bridge products intercept the data access commands and translate them to Adabas direct calls. Read ***Adabas Bridges***, elsewhere in this guide, for more information about Adabas bridges.

Special User Programs: Triggers and Stored Procedures

The Adabas triggers and stored procedures facility, an integral part of Adabas, can be used with Natural (read ***Natural Application Development Environment***, elsewhere in this guide) to write and manage *triggers* and *stored procedures* in the Adabas server environment. *Triggers* are systematically used programs that are started automatically based on an event; they can be used to ensure referential integrity, for instance. *Stored procedures* are programs used by a number of different clients that are executed by Adabas as a result of a special user call. Storing these programs in an Adabas file on the server reduces the amount of data traffic to and from the server. Read ***Using Triggers and Stored Procedures***, elsewhere in this guide, for more information about Adabas triggers and stored procedures.

TP Monitors, Adalinks, and the Adabas API

Since most systems do not allow a standard call to Adabas, Software GmbH provides an application program interface (API) to translate calls issued by an application program into a form that can be handled by Adabas. The Adabas API is available across all supported mainframe platforms for both batch and online operations.

Online operations are controlled by teleprocessing (TP) monitors, which serve as telecommunication interfaces to Adabas. Supported TP monitors are listed in ***Operating Environments***, elsewhere in this guide. Software GmbH provides versions of the Adabas API that are specific to particular TP monitors. *Adalink* is a generic term that refers to the portion of the API that is specific to a particular TP monitor.

Batch applications are supported in both single-user and multiuser mode (read ***Modes of Operation***, elsewhere in this guide, for a discussion of these modes). The Adabas batch API uses a standard calling convention that is supported by all major programming languages through their CALL mechanisms. Most mainframe operating systems allow batch application modules to be linked either with the batch API or with ADAUSER.

We strongly recommend linking batch application programs with the Adabas version-independent module ADAUSER. The ADAUSER module can optionally be linked with the Adabas API. ADAUSER provides upward compatibility with Adabas releases and a degree of isolation from future changes to the API or to mechanisms that handle interregion communication between the user and the nucleus (read ***Modes of Operation***, elsewhere in this guide).

A client running under IBM's OpenEdition can access Adabas. An OpenEdition application containing calls to Adabas can be linked with either the batch API or ADAUSER.

Running Adabas

- Session Types
- Storage Areas
- Modes of Operation
- ADARUN Startup Parameters
- Session Control

Session Types

Three types of sessions can be identified for Adabas:

- The *Adabas session* starts when the nucleus is invoked and ends when the nucleus is terminated. An Adabas nucleus is invoked using job control specific to a particular operating system that contains Adabas startup, or ADARUN, parameters.
- A *user session* is either a batch mode program or a person using a terminal. A user session can occur only during an Adabas session; that is, when the Adabas nucleus is active. It is a sequence of Adabas calls optionally starting with an open user session (OP) command and ending with a close user session (CL) command.
- A *utility session* is executed in batch, or online using the **Adabas Online System**. Some utilities require the Adabas nucleus to be active; others do not. ADARUN startup parameters are also used for executing utilities.

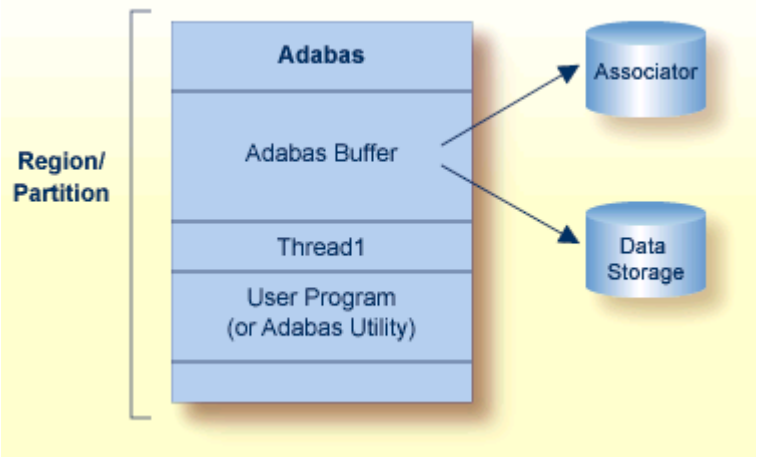
Storage Areas

The Adabas nucleus and each user program or Adabas utility is executed in separate storage areas defined by the operating system. On z/OS, the storage area can be an address space, dataspace, hiperspace, or 64-bit virtual space.

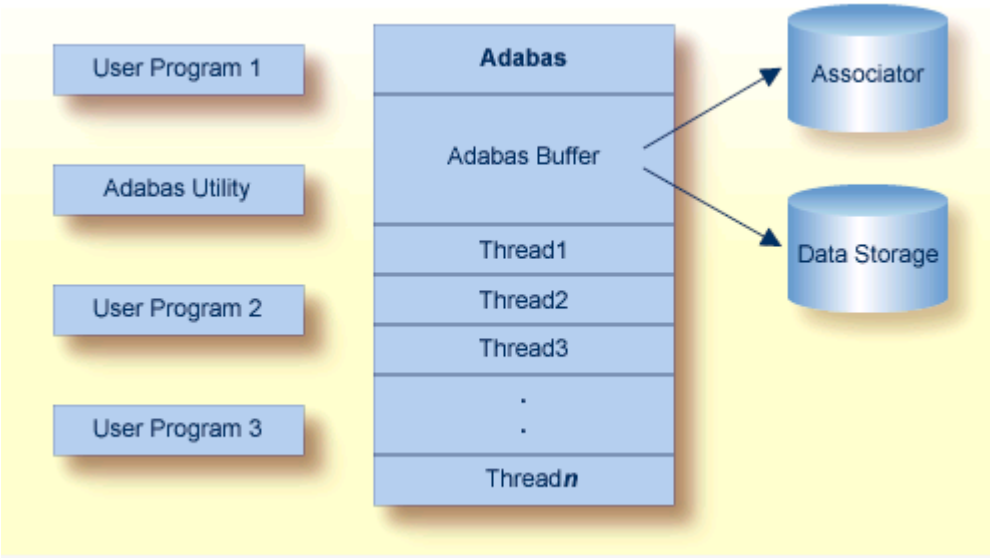
Modes of Operation

Adabas supports two modes of operation: single-user and multiuser.

Single-user mode is in effect when a user program (or Adabas utility) is executed in the same region as the Adabas nucleus.



Multiuser mode is in effect when the Adabas nucleus is located in a separate region. It is the most efficient and therefore the recommended mode of operation.



When using Adabas in multiuser mode, interregion communication is handled by Adabas in a manner that takes optimum advantage of the communications facilities offered by the various operating systems.

For single-user mode, the appropriate Adabas nucleus JCL must be included with the job control for the utility or user program.

ADARUN Startup Parameters

The ADARUN control statement defines and starts the Adabas operating environment. The ADARUN control statement also starts Adabas utilities. ADARUN

- loads the ADAIOR module, which performs all database I/O and other operating-system-dependent functions;
- interprets the ADARUN parameter statements; then loads and modifies the appropriate Adabas nucleus or utility modules according to the ADARUN parameter settings; and
- transfers control to Adabas.

The ADARUN statement, normally a series of entries each specifying one or more ADARUN parameter settings, is specified in the DDCARD data set.

Session Control

Adabas provides several ways to monitor and control Adabas, user, and utility sessions:

- Adabas *operator commands* can be entered from the operator console during an Adabas session or during utility operation.
- The ADADBS OPERCOM utility function can issue operator commands to the Adabas nucleus. Adabas then issues a message to the operator confirming the command execution.
- For those using Adabas Online System (demo or full version), you may be able to execute functions corresponding to operator commands while an Adabas session is active using menu options or direct commands.

Operator commands can be used to terminate an Adabas or user session; display nucleus or utility information; log commands; and change Adabas operating parameters or conditions.

Adabas *direct call commands* can also be used to open and close a user session. Read ***Direct Call Interface***, elsewhere in this guide, for more information about direct call commands.

3 Adabas Design

- Adabas Entities 16
- Database Components 17
- Database Files 26
- Record and Field Definitions 30
- Parallel Participant Table (PPT) 52
- Spanned Records 54

Database systems often involve complex data structures and data handling procedures that can be designed and used only by persons with extensive knowledge and experience. Adabas has a remarkably simple structure by comparison, yet it provides significant advantages for operational efficiency, ease of design, definition, and database evolution.

Adabas Entities

In Adabas, a *field* is the smallest logical unit of information (e.g., current salary) that may be defined and referenced by the user. A *record* is a collection of related fields that make up a complete unit of information (e.g., all the payroll data for a single employee). A *file* is a group of related records that have the same format (with some exceptions; read ***Multiple Record Types in One File***, elsewhere in this guide). A *database* is a group of related files.

- Adabas Limits
- Adabas Space Management

Adabas Limits

The table below shows the maximum number that mainframe Adabas supports for each entity:

Entity	Maximum
Databases	65,535
Blocks per database	2,147,483,646 using 4-byte RABNs
Files per database	the lower of 5,000 or the Associator block size minus one
Records per file	4,294,967,294 using 4-byte ISNs
Fields per record	3214
Uncompressed record length	depends on the operating system
Compressed record length	Data Storage block size *Spanned records*, supported in Adabas version 8 (or later), split a logical record into multiple physical records, each smaller than one Data Storage (DS) block. For more information, read ***Spanned Record Support***, elsewhere in this section.

Adabas Space Management

The disk storage space allocated to a single Adabas database is segmented into *logical* Adabas files. A certain part of the overall space within the database is allocated to each logical file. When the space is filled with records from the file, Adabas automatically allocates more space to the file from the common free space pool. This dynamic space allocation, together with the dynamic recovery of released space, allows Adabas databases to run without intervention for long periods of time.

The distribution of database space across disk drives can be controlled by physically segmenting it into multiple independent data sets. When all physical database space is filled, more data sets can be allocated dynamically, or the size of existing data sets can be increased so that new physical files can be loaded without reorganizing the entire database.

Database Components

To support the separation of data and access structures, the Adabas nucleus uses three database components:

- Data Storage for compressed data
- Associator for data management and retrieval
- Work, a scratch area for complex search criteria, etc.

This section describes each of these database components.

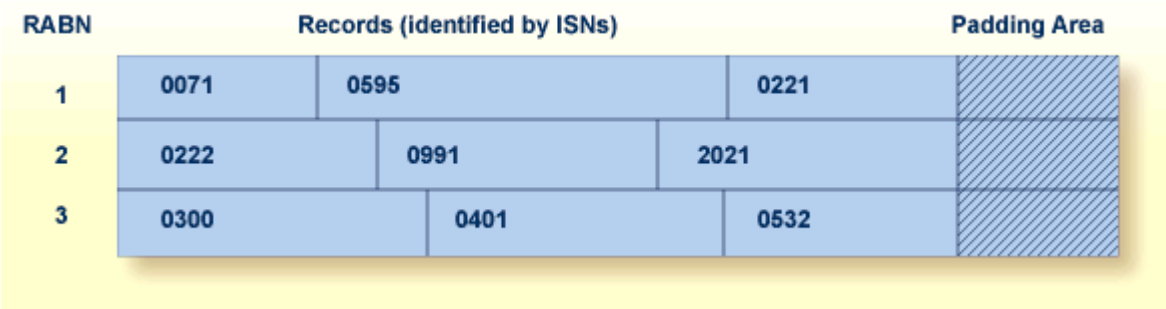
- Data Storage
- Associator
- Work
- Other Components

Data Storage

Data Storage is divided into blocks, each identified by a 3- or 4-byte relative Adabas block number, or RABN, that identifies the block's physical location relative to the beginning of the component. Data Storage blocks contain one or more physical records and a padding area to absorb the expansion of records in the block.

A logical identifier stored in the first four bytes of each physical record is the only control information stored in the data block. This internal sequence number or ISN uniquely identifies each record and never changes. When a record is added, it is assigned an ISN equal to the highest existing ISN plus one. When a record is deleted, its ISN is reused only if you instruct Adabas to do so. Reusing ISNs reduces system overhead during some searches and is recommended for files with records that are frequently added and deleted.

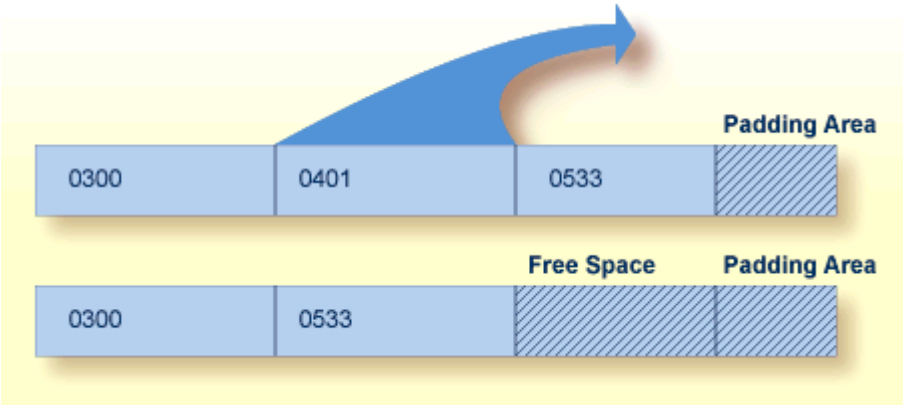
For each file, between 1-90 percent (default 10%) of each block can be allocated as padding based on the amount and type of updating expected. This reserved space permits records to expand without migrating to another block and thus helps to minimize system overhead.



- Free Space and Space Reusage
- Compression
- Dynamically Increasing Data Storage Space

Free Space and Space Reusage

If records become too large for their blocks, they migrate to new locations. When a record migrates or is deleted, free space is opened in the data block between the last record and the padding area. The following figure shows free space created when the record with ISN 0401 becomes too large for the block and migrates to another block:



You can instruct Adabas to reuse free space. Reusing space saves computer time, since Adabas then reads fewer physical blocks during searches. It is recommended for all files.

Compression

Data compression significantly reduces the amount of storage required. It also permits the transmission of more information per physical transfer, resulting in greater I/O efficiency.

Adabas retains data records in compressed form. Several compression options are supported:

- default compression;
- null suppression; and
- fixed format; and
- forward or prefix index compression.

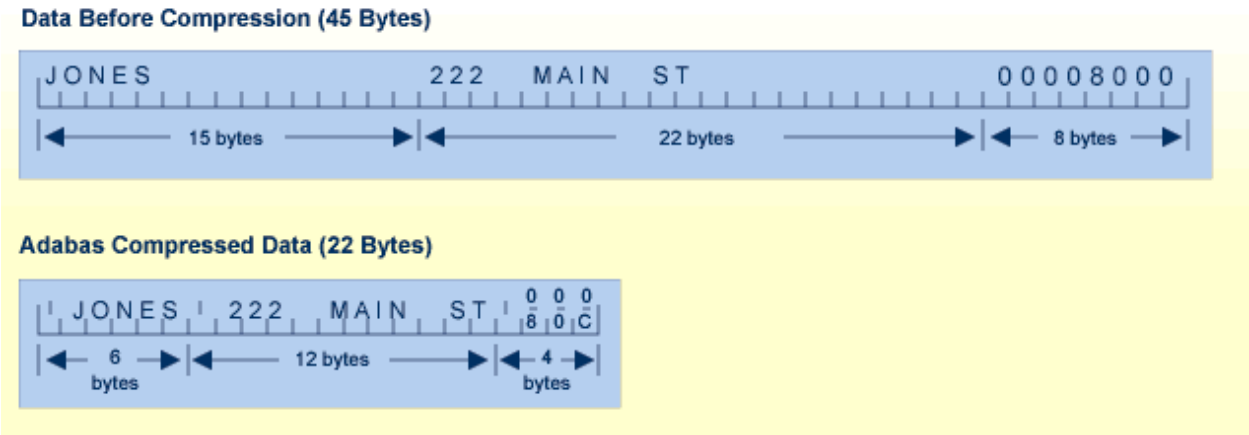
The first three options define and execute compression at the field level, with null suppression and fixed format compression added as field options.

The fourth option, forward or prefix index compression, compresses the descriptor values in the Associator's inverted list. It can be implemented at the file or the database level, in which case specific files can be set differently; the file-level setting overrides the database setting. The forward index compression option is set using the ADALOD utility and can be changed using the ADAORD utility. This compression option is more fully described in ***Inverted Lists***, elsewhere in this section.

The null suppression and fixed format options are added as field options and are discussed in ***Data Compression Options FI and NU***.

Default compression deletes trailing blanks in alphanumeric fields and leading zeros in binary fields. An inclusive length byte (ILB) at the beginning of the field indicates the total number of stored bytes, including the ILB. Thus, if "Susan" is entered in a "first-name" field defined with a 20-character length and default compression, its stored size will be six bytes: five bytes for the letters of the name, plus one byte for the ILB. In addition, empty fields in a record are not stored; an empty field is replaced by a one-byte empty field counter (EFC). Adabas can store up to 63 contiguous empty fields in a single hexadecimal byte.

Many Adabas files require only 50% to 60% of the space used for the raw data. Even with the addition of approximately 25% for the access structures stored in the Associator, Adabas storage requirements are still less than those required for traditional file storage or for DBMSs that do not use data compression.



Dynamically Increasing Data Storage Space

You can dynamically increase the size of your Data Storage space. This makes it possible to increase the size of the database without incurring a database outage, which can be critical if your database needs to run 24x7 (24 hours a day, 7 days a week). You can do this in one of two ways:

1. You can increase the size of the last existing Data Storage data set for the database by extending the data set and formatting the appended space, and then allocating it to the database.
2. You can add a new Data Storage data set to the database by allocating the data set to the operating system, formatting the data set, and then allocating it to the database.

Both of these methods can be accomplished without restarting the Adabas nucleus.

For more information about dynamic increases to the Data Storage space, read *Dynamically Increasing Associator and Data Storage Space*, in the *Adabas DBA Tasks Manual*.

Associator

The Associator is an organizational unit used for storing the structures required to access data in Data Storage. It contains the following elements:

1. Two *general control blocks (GCBs)* for the database. The GCBs provide information regarding the physical characteristics of the database, such as the database ID (DBID), the number of files loaded, the number of Associator, Data Storage, and Work extents, the Associator, Data Storage, and Work device types, system file information, Data Storage Space Table (DSST) extents, and the database version indicator.
2. Individual *file control blocks (FCBs)* for each file. The FCBs identify the physical characteristics and associated RABNs of database files. The contents include the file name, file number, current file status, the ISN reuse settings, the space reuse settings, MINISN and MAXISN settings, the first free ISN, and the number of updates against the file. In addition, the first RABN, last RABN, and first unused RABN are stored in the FCB.

3. All tables needed to control and maintain the database including these:
   - A field definition table (FDT) for each file. For more information about the FDT, read ***Records and Field Definitions***, elsewhere in this guide.
   - A free space table (FST) that lists the unused Associator and Data Storage blocks. These are shown in the database report (ADAREP utility) under the heading "Unused Storage".
   - A parallel participant table (PPT), used to track the active Adabas nucleus (or nuclei in a clustered environment) that manages a particular database. For more information about the PPT, read ***Parallel Participant Table (PPT)***, elsewhere in this guide.
   - Coupling lists for physically coupled files. For more information about physically coupled files, read ***Coupled Files***, elsewhere in this guide.
4. An **inverted list** for each descriptor in each file of the database and an **address converter** for each file.
5. If **spanned records** are used in a file, a secondary address converter for the file.

Inverted Lists

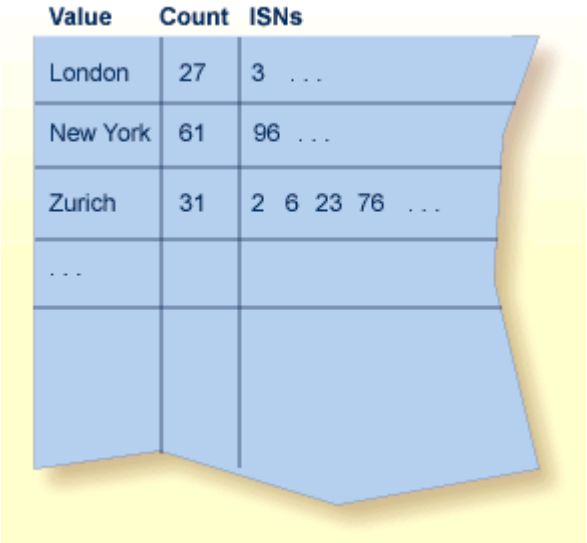
An inverted list, which is used to resolve Adabas search commands and read records in logical sequence, is built and maintained for each field in an Adabas file that is designated as a key field or *descriptor* (read ***Descriptor Options DE, UQ, and XI*** , elsewhere in this guide). It is called an *inverted list* because it is organized by descriptor value rather than by ISN. The list comprises the normal index (NI) and as many as 14 upper indexes (UI).

The normal index (NI) of the inverted list for a particular descriptor has an entry for each value. The entry contains the value itself, the number of records in which the value occurs, and the ISNs of those records.

To increase search efficiency, upper index (UI) levels are automatically created by Adabas as required, each level to manage the next lower level index. The first level UI, like the NI it manages, contains entries for only one descriptor in each index block. All other UI levels contain entries for all descriptors in each index block. UIs require a minimal amount of space: two blocks is the minimum.

Note: The Adabas direct access method (ADAM) facility permits the retrieval of records directly from Data Storage without accessing the inverted lists. The Data Storage block number in which a record is located is calculated using a randomizing algorithm based on the ADAM key of the record. The use of ADAM is completely transparent to application programs and query and report writer facilities. See ***Random Access Using the Adabas Direct Access Method (ADAM)***, elsewhere in this guide, for more information.

The following figure shows a typical normal index for the descriptor CITY in a customer file.



The example indicates that there are 31 records with the CITY Zurich (the ISNs of these records are 2,6,23,76...).

Forward (or *front* or *prefix*) index compression removes redundant prefix information from index values. Within one index block, the first value is stored in full length. For all subsequent values, the prefix that is common with the predecessor is compressed. An index value is represented by:

```
<l,p,value>
```

-where

p	is the number of bytes that are identical to the prefix of the preceding value.
l	is the exclusive length of the remaining value including the p-byte.

For example:

Before Compression	After Compression
ABCDE	6 0 ABCDE
ABCDEF	2 5 F
ABCGGG	4 3 GGG
ABCGGH	2 5 H

The decision to compress index values is based on the similarity of index values and the size of the file:

- the more similar the index values, the better the compression results.
- small files are not good candidates because the absolute amount of space saved would be small whereas large files are good candidates for index compression.

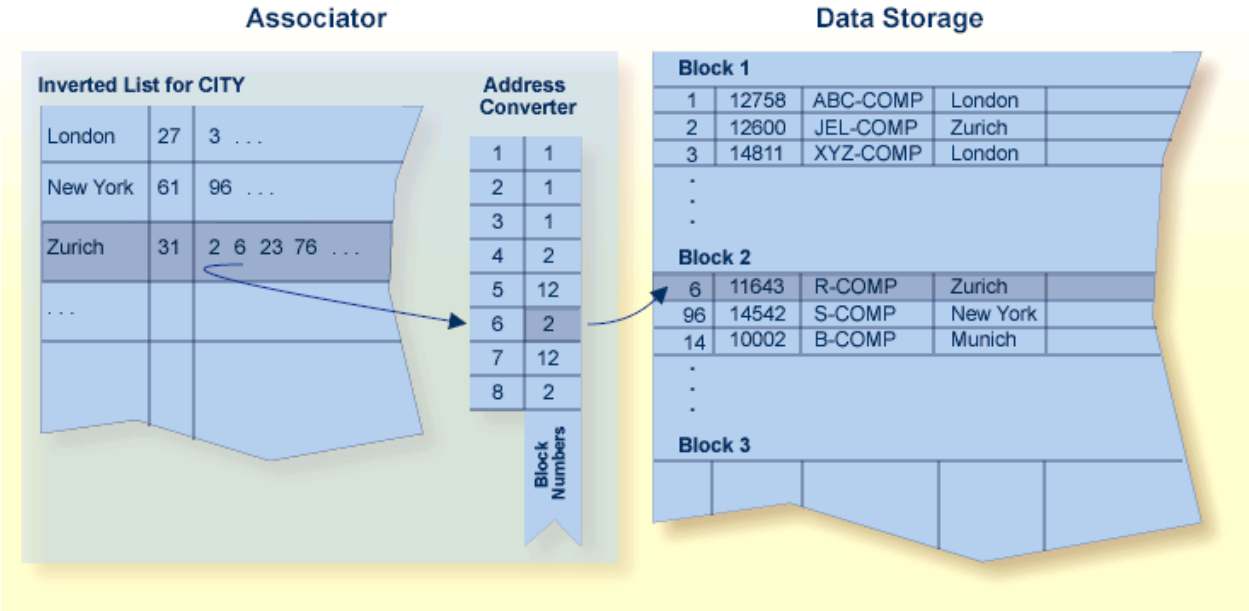
Even in a worst case scenario where the index values for a file do not compress well, a compressed index will not require more index blocks than an uncompressed index.

Address Converter

The address converter determines the physical location of a record. It is an index that maps the logical identifier of a record (that is, the ISN) to the relative Adabas block number (RABN) of the Data Storage block where the record is stored. If spanned records are used, a secondary address converter is used to map the secondary ISNs to the RABNs of the Data Storage blocks where the secondary records are stored. For more information about spanned records, read *Spanned Records*, elsewhere in this guide.

The address converter contains a list of RABNs in ISN order. Only the RABNs are actually stored in the address converter; the ISNs are identified by their relative position.

The following figure shows the relationship between an inverted list, the address converter, and Data Storage. For example, to determine the physical location of the record whose ISN is 6, Adabas uses the ISN as an index into the address converter. The sixth entry in the address converter is 2. Therefore, ISN 6 is located in physical block 2 in Data Storage for this file.



When a record moves or is deleted, Adabas updates the address converter automatically and transparently.

Since the ISN for a record never changes, and its physical block address is stored only in the address converter entry, the record itself may be moved in Data Storage with only one update to the address converter required and with no extension to the access path of the record.

Even if a record has many descriptors defined, the inverted list for each descriptor need not be modified because it contains ISNs.

This process explains how Adabas is able to perform simple and complex searches quickly and efficiently without storing pointer information in Data Storage.

Dynamically Increasing Associator Space

You can dynamically increase the size of your Associator space. This makes it possible to increase the size of the database without incurring a database outage, which can be critical if your database needs to run 24x7 (24 hours a day, 7 days a week). You can do this in one of two ways:

1. You can increase the size of the last existing Associator data set for the database by extending the data set and formatting the appended space, and then allocating it to the database.
2. You can add a new Associator data set to the database by allocating the data set to the operating system, formatting the data set, and then allocating it to the database.

Both of these methods can be accomplished without restarting the Adabas nucleus.

For more information about dynamic increases to the Associator space, read *Dynamically Increasing Associator and Data Storage Space*, in the *Adabas DBA Tasks Manual*.

Work

The Work area stores information in four parts:

Part	Stores . . .
1	data protection information required by the routines for autorestart and autobackout. Read ***Backout, Recovery, and Restart***, elsewhere in this guide, for more information.
2	intermediate results (ISN lists) of search commands.
3	final results (ISN lists) of search commands.
4	data related to two-phase commit processing.

Other Components

- Sort and Temp Areas

- Logs

Sort and Temp Areas

Certain Adabas utilities (ADAINV, ADALOD) require two additional data sets, sort and temp, for sorting and intermediate storage of data. Certain functions of other utilities require the temp data set for intermediate storage.

The size of the temp and sort data sets varies according to the utility function to be executed. These data sets can be allocated during the job and then released, or permanent data sets can be allocated and reused.

Logs

Adabas uses the following optional logs:

- The Command log (CLOG) records information from the control block of each Adabas command that is issued. The CLOG provides an audit trail and can be used for debugging and for monitoring the use of resources. Single, dual, or multiple (2-8) data sets can be used (multiple data sets are recommended).

  Timestamps in an Adabas 8 command log created using the ADARUN CLOGLAYOUT=8 parameter are stored in machine time (GMT), whereas CLOGLAYOUT=5 timestamps are stored, as always, in local time. The LORECX record layout that describes the CLOGLAYOUT=8 command log includes a differential time field that stores the difference between machine time and local time at the time the CLOG record is written. This field allows you to calculate the local time of a command log record.

  Because of the difference in timestamp formats, we do not recommend that you mix or merge command logs created using different CLOGLAYOUT settings. This is especially true for Adabas nuclei in a cluster environment. For more information, read *CLOGLAYOUT : Command Logging Format* , in *Adabas Operations Manual*.
- The Protection log (PLOG) records before- and after-images of records and other elements when changes are made to the database. It is used to recover the database (up to the last completed transaction or ET) after restart. Single, dual, or multiple (2-8) data sets can be used (multiple data sets are recommended).
- The Recovery log (RLOG) records additional information that the Adabas Recovery Aid uses to construct a recovery job stream. Read the **ADARAI utility discussion**, elsewhere in this guide, for more information.

Database Files

Each database contains system files and data files. A data file is generally created for each record structure required; that is, for each set of related fields identified.

Files are loaded into the database using the ADALOD utility. A file number must be unique in the database and not greater than the maximum file number defined for the database in the MAXFILES parameter. Checkpoint, security, trigger, and system files can have two-byte file numbers, but cannot be greater than 5000. Physically coupled files cannot include files with numbers greater than 255. File numbers are assigned by the user in any sequence.

This section describes the different types of database files.

- System Files
- Coupled Files
- Structuring Files to Enhance Performance

System Files

Adabas uses certain files to store system information. Using the ADALOD utility's FILE parameter, you can identify an Adabas *system* file as one of the following:

CHECKPOINT	Adabas checkpoint system file
REPLICATOR	Event Replicator system file
SECURITY	Adabas security system file
SYSFILE	Adabas system file
SLOG	Event Replicator subscription logging system file
TRIGGER	Adabas trigger system file

Coupled Files

File coupling allows you to select, using a single search command, records from one file that are related (coupled) to records containing specified values in a second file.

- Physical Coupling

- Logical or Soft Coupling

Physical Coupling

Any two files with file numbers 255 or lower may be physically coupled if a common descriptor (read ***Descriptor Options DE, UQ, and XI***, elsewhere in this guide) with identical format and length definitions is present in both files. A single file may be coupled with up to 18 other files, but only one coupling relationship may exist between any two files at any one time. A file may not be coupled to itself.

When files are coupled, coupling lists are created in the Associator for each file being coupled. File coupling is bidirectional rather than hierarchical in that two coupling lists are created for each coupling relationship with each list containing the ISNs that are coupled to the other file.

Once the physical coupling lists have been created, any key field in either file may be used within a search criteria.

Physical coupling may add a considerable amount of overhead if the files involved are frequently updated. The coupling lists must be updated if a record in either of the files is added or deleted, or if the descriptor used as the basis for the coupling is updated in either file.

Physical coupling may be useful for information retrieval systems in which

- files seldom change;
- the additional overhead of the coupling lists is insignificant compared with the increased ease of formulating queries; or
- files are small and primarily query-oriented.

Logical or Soft Coupling

Multiple files may also be queried by specifying the field to be used for interfile linkage in the search criteria. Adabas then performs all necessary search, read, and internal list matching operations.

This technique is called logical or soft coupling because it does not require the files to be physically coupled. Although logical coupling requires read commands, it is normally more efficient because it avoids the increased overhead of coupling lists.

Structuring Files to Enhance Performance

An Adabas database with one file for each record type supports any application functions required of it and is the easiest to manipulate for interactive queries, but it may not yield the best performance:

- As the number of Adabas files increases, the number of Adabas calls increases. Each Adabas call requires interpretation, validation and, in multiuser mode, supervisor call (SVC) and queuing overhead.
- In addition to the input/output (I/O) operations necessary for accessing at least one index, address converter, and Data Storage block from each file, the one-file-per-record-type structure requires buffer pool space. If sufficient buffer space is not available, blocks are overwritten that may be needed for a later request.

The number of Adabas files used by critical programs can be reduced by

- using multiple-value fields and periodic groups (read ***Field Levels***, elsewhere in this guide);
- linking physical files into a single logical (expanded) file;
- including more than one type of record in an Adabas file;
- including records for more than one category of user in an Adabas (multiclient) file; and
- controlling data duplication and the resulting high resource usage.

This section describes the following topics:

- Expanded Files
- Multiple Record Types in One File
- Multiclient Files
- Controlled Data Redundancy

Expanded Files

If you have a large number of records of a single type, you may need to spread the records over multiple physical files.

To reduce the number of files accessed, Adabas allows you to link multiple physical files containing records of the same format together as a single logical file. This file structure is called an expanded file and the physical files comprising it are the component files. An expanded file can comprise up to 128 component files, each with a unique range of logical ISNs. An expanded file cannot exceed 4,294,967,294 records.

Note: Since Adabas now supports larger file sizes and a greater number of Adabas physical files and databases, the need for expanded files has, in most cases, been removed.

Although an application program addresses the logical file (the address of the file is the number of the expanded file's base component or anchor file), Adabas selects the correct component file

based on the data in a field defined as the criterion field. The data in this field has characteristics unique to records in only one component file. When an application updates the expanded file, Adabas looks at the data in the criterion field in the record to be written to determine which component file to update. When reading expanded file data, Adabas uses the logical ISN as the key to finding the correct component file.

Multiple Record Types in One File

Multiple record types can be defined within a single physical record; each record type is a logical record composed of a subset of the fields defined for the file. Fields that do not belong to a given type are null-suppressed.

Record types can be identified to Adabas by

- defining a record type field with values to differentiate one type from another; or
- using values of an existing field to differentiate type; for example, to differentiate two types, a value of zero for a field common to both types might identify one type and any nonzero value for the same field might identify the other type.

Multiclient Files

Records for multiple users or groups of users can be stored in a single Adabas physical file defined as multiclient. The multiclient feature divides the physical file into multiple logical files by attaching an internal owner ID to each record.

The owner ID is assigned to a user ID. A user ID can have only one owner ID, but an owner ID can belong to more than one user. Each user can access only the subset of records that is associated with the user's owner ID.

Note: For any installed external security package such as RACF, CA-ACF2, or CA-Top Secret, a user is still identified by either Natural ETID or LOGON ID.

All database requests to multiclient files are handled by the Adabas nucleus.

Controlled Data Redundancy

Physical redundancy increases storage requirements but may also enhance performance and decrease complexity. For example, if a database stores customer and order information in a customer-orders file and product descriptions in an inventory file, and a program that generates invoices requires product descriptions in addition to customer-order data, it might enhance performance to store a duplicate copy of the product descriptions in the customer-orders file.

Logical redundancy also increases storage demands while decreasing complexity. It involves storing in one file the results of a process on data in another file; thus, the duplicate data is implied by the content of another file, rather than being physically stored in two places.

Physical and logical redundancy cause update programs to run more slowly. The duplicate updates required when changes in one file affect records in another file may degrade performance severely. Redundancy should be used only for static data or data that is updated rarely. You can control data redundancy by using multiple-value fields, periodic groups, and multiple record types within a file.

Record and Field Definitions

In Adabas, the record structure and the content of each field in a physical file are described in a Field Definition Table, or FDT, which is stored in the Associator. There is one FDT for each database file. The FDT is used by Adabas during the execution of Adabas commands to determine the logical structure and characteristics of any given field (or group) in the file.

Spanned records, supported in Adabas version 8 (or later), split a logical record into multiple physical records, each smaller than one Data Storage (DS) block. For more information, read ***Spanned Record Support***, elsewhere in this section.

This section covers the following topics:

- Record Structure and the FDT
- Field Levels and Group Fields
- System Fields
- Field Names
- Field Length and Data Format
- Field Options
- Special Fields and Descriptor Fields

Record Structure and the FDT

The FDT lists the fields of the file in physical record order, provides a quick index to the file's records, and defines the file's fields, subfields, superfields, and descriptors (including collation descriptors, subdescriptors, superdescriptors, hyperdescriptors, and phonetic descriptors). A minimum of one and a maximum of 3214 field definitions may be specified.

Information about each field includes the level, name, length, format, options, and special field and descriptor attributes.

```
FIELD DESCRIPTION TABLE

       I      I        I        I              I
 LEVEL I NAME I LENGTH I FORMAT I  OPTIONS     I         PARENT OF
       I      I        I        I              I
 ------I------I--------I--------I--------------I----------------------------I
       I      I        I        I              I                            I
   1   I  AA  I    8   I    A   I DE,UQ        I                            I
   1   I  AB  I        I        I              I                            I
   2   I  AC  I   20   I    A   I NU           I                            I
   2   I  AE  I   20   I    A   I DE           I  SUPERDE,PHONDE            I
   2   I  AD  I   20   I    A   I NU           I                            I
   1   I  AF  I    1   I    A   I FI           I                            I
   1   I  AG  I    1   I    A   I FI           I                            I
   1   I  AH  I    6   I    U   I DE           I                            I
   1   I  A2  I        I        I              I                            I
   1   I  AO  I    6   I    A   I DE           I  SUBDE,SUPERDE             I
   1   I  AQ  I        I        I PE           I                            I
   2   I  AR  I    3   I    A   I NU           I  SUPERDE                   I
   2   I  AS  I    5   I    P   I NU           I  SUPERDE                   I
   1   I  A3  I        I        I              I                            I
   2   I  AU  I    2   I    U   I              I  SUPERDE                   I
   2   I  AV  I    2   I    U   I NU           I  SUPERDE                   I
```

The order of the fields listed in the FDT determines the structure of the record and the efficiency of retrieval. The following factors should be considered when ordering fields:

- Fields that will be accessed frequently should be ordered first in the FDT. This technique reduces CPU time because Adabas does not have to read the whole record when retrieving a field.
- Fields that will frequently be accessed together should be assigned to a group field.
- Fields that will *always* be accessed together should be defined as a single field. This technique may inhibit compression and query language use; however, it decreases processing time by providing more efficient internal processing and shorter format buffers.
- If appropriate, fields that will frequently be empty should be ordered together in the FDT and set to use default compression or null suppression.
- Numeric fields should be loaded in the format in which they will be used most often.

Field Levels and Group Fields

When two or more consecutive fields in the FDT are frequently accessed together, you can reference them together by defining a group field. Other than its level and Adabas short name, a group field has no attributes defined. It immediately precedes its member fields in the FDT. A higher field level number is used to assign the member fields to the group field. Adabas supports up to seven field levels. User programs can access each member field individually, or all member fields together by referencing the group field.

For example, in the illustration of the Field Definition Table (FDT) in the section ***Records and Field Definitions***, field AB is defined as a group field and assigned to level 1. Fields AC, AE, and AD are assigned to level 2, indicating that they belong to group field AB. The next field, AF, is assigned to level 1, indicating that it is not part of the AB group. User programs can access AC, AE, and AD individually, or together by referencing the group field AB.

A group field can be assigned as a *periodic group field* if it is comprised of fields that can have more than one value (for example, group field AQ in the figure).

System Fields

Adabas allows you to define *system fields* in your Adabas files.

A system field is a field in an Adabas file whose value is automatically set by the Adabas nucleus when records are inserted or updated on the file. Optionally, you can specify that some system field values only be set when records are inserted. System fields are fields that store information such as the job name of the user making the update, the eight-byte user ID of the user, the session ID of the user, or the time at which the update was made.

The value of a system field refers to an insert or update of an entire record. You cannot define a system field that refers to only a portion of a record.

Values for system fields are saved in the compressed storage record in the same manner that other Adabas fields are stored.

This section covers the following topics:

- Allowed Types of System Fields
- Defining System Fields
- System Fields as MU Fields
- System Field Rules

- System Field Processing by an Adabas Nucleus

Allowed Types of System Fields

System fields containing the following types of information can currently be defined in an Adabas file:

- Job name: The job name of the user inserting or updating a record.
- ETID: The eight-byte user ID of the user inserting or updating a record. This is the user ID set in the Additions 1 field of an OP (open) command for the user session.
- Session ID: The 28-byte user ID of the user inserting or updating a record.
- Session user: The last eight bytes of the 28-byte session ID of the user inserting or updating a record.
- **Security system user ID**: The eight-byte security system ID of the user inserting or updating a record.
- Time: The date or date and time at which a record is inserted or updated.

Note: Information about record deletion is not recorded in system fields for the simple reason that the value of the system field is itself deleted along with the record.

Defining System Fields

System fields may *not* be part of a periodic group. Otherwise, they can be simple fields or MU fields (although not MU fields in a periodic group).

System fields are defined in the same manner as other database fields using FNDEF definitions in ADACMP COMPRESS utility runs, except each system field definition must include an **SY field option**. If you only want the system field values updated when a record is inserted (and not when it is updated), you would specify the **CR field option** in the system field definition. If the system field is a multiple occurrence (MU) field, it *cannot* be defined with the **CR field option**. If the system field is not defined with the CR option, it *must* be defined as an MU field.

System fields can also be added to a file using the ADADBS NEWFIELD utility function. The same SY and CR field options are supported by ADADBS NEWFIELD.

The SY and CR options cannot be changed for a field using the ADADBS CHANGE utility function. In other words, a field defined as a system field cannot later be changed from a system field to a non-system field. However, a system field may be logically deleted. Note that system field values for logically deleted fields are still set when record insertions or updates occur; this ensures that the system field values are correct if they are later no longer logically deleted

For more information about the SY and CR field options, read *Field Options*, in the *Adabas Utilities Manual*.

System Fields as MU Fields

A system field cannot be part of a periodic group. In addition, if the system field is defined with the CR field option, it cannot also use the MU option. Likewise, if the system field is *not* defined with the CR option, it *must* be defined as an MU field.

If a system field is an MU (multiple occurrence) field, Adabas sets the field values on record insertion and record update in a specific way.

- When a record is inserted, an appropriate value is set in the first occurrence of the MU system field. The MU system field will contain only a single value (the value set in the first occurrence of the field).
- When a record is updated, an appropriate value is set in the first occurrence of the MU system field. Values held previously in occurrences 1 to N of the MU system field will be shifted to occurrences 2 to N+1. If the maximum number of MU system field values is reached for the file, the oldest value is dropped from the file. For example, if a file is defined with a maximum number of MU system field values set to 5 and 5 values are already present in the file, an update to the MU system field will drop the value in occurrence 5 of the field, shift the others down and insert the new value in occurrence 1.

The maximum number of MU system field values allowed in a file is stored in the File Control Block (FCB) for the file and is set when you load the file using ADALOD LOAD. Thereafter, the SYFMAXUV value can be modified using the ADADBS MODFCB utility function. This setting applies to all MU system fields in the file. For more information, read *MODFCB: Modify File Parameters* , in the *Adabas Utilities Manual*.

The following are examples of valid system fields. Note that the fourth example from the top is *not* an MU field, but is defined with the CR option.

```
01,ET,8,A,NU,MU,SY=OPUSER
01,SU,8,A,MU,NU,NV,SY=SESSIONUSER
01,SI,28,A,NU,NV,MU,SY=SESSIONID
01,D1,8,U,NU,DT=E(DATE),SY=TIME,CR
01,TI,14,U,MU,NU,DT=E(DATETIME),SY=TIME
01,TZ,14,U,MU,NU,DT=E(DATETIME),TZ,SY=TIME
01,Z3,8,A,MU,SY=JOBNAME
01,SC,8,A,NU,MU,SY=SECUID
```

System Field Rules

The following rules apply to system fields:

1. The field format for JOBNAME, OPUSER, SESSIONID, SESSIONUSER, and **SECUID** system fields must be A (alphanumeric).
2. The format and length of a TIME system field will be enforced based on the rules set for the date-time edit mask specified for the field.

   A date-time edit mask (DT field option) must be specified for time system fields. However, the DT field option is not valid for any other type of system field.
3. System fields cannot be a periodic group field (the PE field option *cannot* be specified), nor can it be a member of a periodic group.
4. A system field can be a descriptor (the DE field option can be specified) or a unique descriptor field (the UQ field option can be specified).
5. The system field may be the parent of a superdescriptor field or of a special descriptor.
6. A system field *cannot* be a long alpha or wide-character field or a large object field (the LA and LB field options *cannot* be specified).
7. Security-by-value is allowed for system fields.
8. The CR field option can only be specified if the SY field option is also specified for a field.
9. A system field can be defined to have a fixed storage length (the FI field option can be specified). If FI is specified, the field length must exactly match the lengths shown in the SY field option description. If it is not specified, any field length is allowed in the field definition; the length of the data stored for each field will match the lengths shown in the SY field option description.
10. A system field must have either the MU option *OR* the CR option specified (but not both). The MU and CR options are mutually exclusive.
11. Null values can be suppressed for a system field (the NU field option can be specified).
12. Alphanumeric system fields can be processed in the record buffer without being converted (the NV field option can be specified).
13. The value of a system field refers to the insert or update of an entire record. You cannot define a system field that refers to only a portion of a record.
14. Information about record deletion is not recorded in system fields for the simple reason that the value of the system field is itself deleted along with the record.
15. System fields are not required in a file (none can be specified). In addition, one or more system fields of the same type can be defined for a file.
16. The SY and CR options cannot be changed for a field using the ADADBS CHANGE utility function. In other words, a field defined as a system field cannot later be changed from a system field to a non-system field.

17. A system field can be logically deleted. Note that system field values for logically deleted fields are still set when record insertions or updates occur; this ensures that the system field values are correct if they are later no longer logically deleted.

System Field Processing by an Adabas Nucleus

When a record is inserted or updated in an Adabas database file with system fields, the system field values are set by the nucleus. If system fields are specified one or more times in the format buffers and record buffers of an insert or update command, the values passed by the user are ignored.

The following processing occurs by the Adabas nucleus for system fields:

1. If the system field is *not* defined with the CR option, the system field value is set by the nucleus when an insert command and when an update command is issued.
2. If the system field is defined with the CR option, the system field value is set by the nucleus when an insert command is issued, and is left as is when an update command is issued.

Field Names

A field is identified to Adabas by a two-character Adabas short name that must begin with an alphabetic character (either upper- or lowercase) and can be followed by a numeral or an alphabetic character (either upper-or lowercase) and must be unique within a file. The combinations E0-E9 are reserved and special characters are not allowed. Adabas assigns short names to fields automatically, although you can choose to assign them yourself. Adabas uses the short names internally and actually accesses fields by their short names.

Note: Lowercase fields will not display correctly (they will be converted to uppercase) if you use the ADARUN parameter settings MSGCONSL=UPPER, MSGDRUCK=UPPER, or MSGPRINT=UPPER.

Field Length and Data Format

Field values are fixed or variable in length and can be in alphanumeric, binary, fixed-point, floating-point, packed/unpacked decimal, or wide character formats.

The length (expressed in bytes) and format (expressed as a one-character code) of a field define the standards (defaults) to be used by Adabas during command processing. They are used when the field is read/updated unless the user specifies an override.

If standard length is zero for a field, the field is assumed to be a variable-length field. Standard format must be specified for a field. The format specified determines the type of default compression to be performed on the field.

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

Format	Format Description	Maximum Length
A	Alphanumeric (left-justified): see also the long alphanumeric (LA) option in *Long Alpha Option LA* and the large object (LB) option in *Large Object Option LB*, elsewhere in this guide	253 bytes
B	Binary (right-justified, unsigned/positive)	126 bytes
F	Fixed point (right-justified, signed, positive value in normal form; negative value in two's complement form)	8 bytes (always exactly 2, 4, or 8 bytes)
G	Floating point (normalized form, signed)	8 bytes (always exactly 4 or 8 bytes)
P	Packed decimal (right-justified, signed)	15 bytes
U	Unpacked decimal (right-justified, signed)	29 bytes
W	Wide character (left-justified): see also the long alphanumeric (LA) option in *Long Alpha Option LA*, elsewhere in this guide	253 bytes

Field Options

Field options are specified using two-character codes, which may be specified in any order, separated by a comma.

Code	Option	Read Section
CR	A system field will not be modified when updates occur to the record, but only when the record is first inserted.	*System Field Options SY and CR*
DE	The field is to be a descriptor (key).	*Descriptor Options DE, UQ, and XI*
DT	A date-time edit mask is specified for the binary, fixed point, packed decimal, or unpacked decimal field.	*Date-Time Edit Mask Option DT*
FI	The field is to have a fixed storage length; values are stored without an internal length byte, are not compressed, and cannot be longer than the defined field length.	*Data Compression Options FI and NU*
LA	An alphanumeric or wide-character, variable-length field may contain a value up to 16,381 bytes long.	*Long Alpha Option LA* and *Comparing LA and LB Fields*
LB	An alphanumeric field may contain up to 2,147,483,643 bytes (about 2 GB) of data.	*Large Object Option LB* and *Comparing LA and LB Fields*
MU	The field may contain up to about 65,534 values in a single record.	*MU and PE Options and Field Types*
NB	Trailing blanks should not be removed (compressed) from the LA or LB fields. Specification of this option requires the specification of NU or NC as well.	*Blank Compression Option NB*
NC	Field may contain a null value that satisfies the SQL interpretation of a field having no value; that is, the field's value is not defined (not counted).	*SQL Compatibility Options NC and NN*
NN	Field defined with NC option must always have a value defined; it cannot contain an SQL null (not null).	*SQL Compatibility Options NC and NN*

Code	Option	Read Section
NU	Null values occurring in the field are to be suppressed.	*Data Compression Options FI and NU*
NV	An alphanumeric or wide-character field is to be processed in the record buffer without being converted.	*Encoding Conversion Option NV*
PE	This group field is to define consecutive fields (which may include one or more MU fields) in the FDT that repeat together (up to about 65,534 times) in a record.	*MU and PE Options and Field Types*
SY	The field is a system field.	*System Field Options SY and CR*
TZ	The date-time field value is presented in the user's local time and stored in UTC time, allowing for differences in time zones.	*Time Zone Option TZ*
UQ	The field is to be a unique descriptor; that is, for each record in the file, the descriptor must have a different value.	*Descriptor Options DE, UQ, and XI*
XI	For this field, the occurrence (index) number is to be excluded from the unique descriptor (UQ) option set for a periodic group (PE).	*Descriptor Options DE, UQ, and XI*

Descriptor Options DE, UQ, and XI

A *descriptor* is a search key. The DE option indicates that the field is to be a descriptor. The UQ option can only be specified if DE is also specified; it indicates that the DE field is to have a different (i.e., unique) value for each record in the file. If the UQ field is also an MU field or a field in a periodic group, the same value for the field may occur multiple times in the same record, but must be unique in different records. Entries are made in the Associator's inverted list for DE fields, adding disk space and processing overhead requirements.

Any field can be used within a selection criterion. When a field that is used extensively as a search criterion is defined as a descriptor (key), the selection process is considerably faster since Adabas is able to access the descriptor's values directly from the inverted list without reading any records from Data Storage.

A descriptor field can be used as a sort key in a search command, as a way of controlling a logical sequential read process (ascending or descending values), or as the basis for file coupling.

Any field and any number of fields in a file can be defined as descriptors. When a multiple-value field or a field in a periodic group is defined as a descriptor, multiple key values are generated for the record. Key searches may be limited to particular occurrences of a periodic group.

For descriptor fields that are part of a periodic group (PE field), the group index is considered part of the descriptor value in the index. This makes it possible to search for a value plus a group index. By default, a given value plus the group index of one occurrence of a record is considered different than the same value plus the different group index of a second record. Because the group indexes are different, these two occurrences do not violate the “uniqueness” criteria. If you want to eliminate the group index from the uniqueness criteria, use the XI option. The XI option is used for

unique descriptors in periodic groups to exclude the occurrence (index) number from the definition of uniqueness.

Because the inverted list requires disk space and update overhead, the descriptor option should be used judiciously, particularly if the file is large and the field that is being considered as a descriptor is updated frequently. For instance, the inverted list for a periodic group used as a descriptor may be very large because each occurrence is stored.

A descriptor may be defined at the time a file is created, or later by using an Adabas utility. Because the definition of a descriptor is independent of and has no effect on the record structure, descriptors may be created or deleted at any time without the need for database restructuring or reorganization.

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

A portion of a field may be defined as a *subdescriptor*; combinations of fields or portions thereof may be defined as a *superdescriptor*; a user-supplied algorithm may be the basis of a *collation descriptor* or *hyperdescriptor*; and a sounds-like encoding algorithm may be the basis of a *phonetic descriptor*, which may be customized for specific language requirements. Read ***Special Field and Descriptor Attributes***, elsewhere in this guide, for more information.

System Field Options SY and CR

A system field is a field in an Adabas file whose value is automatically set by the Adabas nucleus when records are inserted or updated on the file. Optionally, you can specify that some system field values only be set when records are inserted. A system field cannot be a PE field.

System fields containing the following types of information can currently be defined in an Adabas file:

- Job name: The job name of the user inserting or updating a record. When ADACMP COMPRESS is used to define this type of field, its field value will be the job name of the ADACMP COMPRESS job.
- ETID: The eight-byte user ID of the user inserting or updating a record. This is the user ID set in the Additions 1 field of an OP (open) command for the user session.
- Session ID: The 28-byte user ID of the user inserting or updating a record.
- Session user: The last eight bytes of the 28-byte session ID of the user inserting or updating a record.
- **Security system user ID**: The eight-byte security system ID of the user inserting or updating a record.
- Time: The date or date and time at which a record is inserted or updated.

Use the CR option to indicate that the system field value should only be maintained when a record is inserted and not when it is updated. The CR field option can only be specified for fields defined with the SY field option, but cannot be specified for an MU field.

For complete information about system fields, read ***System Fields***, elsewhere in this section. For more information about the SY and CR field options, read *Field Options*, in the *Adabas Utilities Manual*.

Date-Time Edit Mask Option DT

DT assigns a date-time edit mask to a binary, fixed point, packed decimal, or unpacked decimal field. This option cannot be specified for fields of other formats.

The syntax of the DT option is:

```
DT=E(edit-mask-name)
```

Valid values for edit-mask-name substitutions are described in the following table. It also shows the required minimum field lengths for the different formats of fields that can specify the DT option; the length of the field must be large enough to store the date-time values. Detailed discussions of each edit mask is provided in *Date-Time Edit Mask Reference*, in the *Adabas DBA Tasks Manual*.

Note: In the table, "YYYY" represents the 4-digit year (1-9999), "MM" represents the 2-digit month (1-12), "DD" represents the 2-digit day of the month (1-31), "HH" represents the 2-digit hour (0-23), "II" represents the 2-digit minute within the hour (0-59), "SS" represents the 2-digit second within the minute (0-59), and "XXXXXX" represents the 6-digit microsecond within the second.

edit-mask-name	Description	Minimum Field Length for Field Format			
		B	F	P	U
DATE	The date field is in the format `Z'YYYYMMDD'`.	4	4	5	8
TIME	The time field is in the format `Z'HHIISS'`.	3	4	4	6
DATETIME	The date and time field is in the format `Z'YYYYMMDDHHIISS'`	6	8	8	14
TIMESTAMP	The date and time field is in the format `Z'YYYYMMDDHHIISSXXXXXX'`, with microsecond precision	--	--	11	20
NATTIME	The time field is in Natural T format (tenths of seconds since year zero)	6	8	7	13
NATDATE	The date field is in Natural D format (days since year zero)	3	4	4	7
UNIXTIME	The time field is in UNIX time_t type format (seconds since January 1, 1970)	4	4	6	10

edit-mask-name	Description	Minimum Field Length for Field Format			
		B	F	P	U
XTIMESTAMP	The date and time field is in UNIX timestamp format, with microsecond precision, since January 1, 1970 (`UNIXTIME * 60**6 + microseconds`).	8	8	10	18

The following table contains some examples.

Example	The field contains...
`1,SD,8,U,DT=E(DATE)`	Numeric data in the form `Z'YYYYMMDD'`.
`1,TI,6,U,DT=E(TIME)`	Numeric time in the form `Z'HHIISSD'`
`1,DT,14,U,DT=E(DATETIME)`	A value composed of DATE and TIME
`1,TS,20,U,DT=E(TIMESTAMP)`	A value composed of DATETIME plus microseconds
`1,TT,7,P,DT=E(NATTIME)`	Natural T-format data (tenths of seconds since the year zero)
`1,DD,4,P,DT=E(NATDATE)`	Natural D-format data (days since the year zero)
`1,UU,4,F,DT=E(UNIXTIME)`	UNIX time_t-type data (seconds since January 1, 1970)
`1,XS,8,F,DT=E(XTIMESTAMP)`	UNIX time data (microseconds since January 1, 1970)

Time Zone Option TZ

The TZ field option identifies a date-time field that should be presented in the user's local time and stored in UTC time, allowing for differences in time zones. There is no specific syntax for the TZ field option as there are no parameters; simply specifying TZ in the field definition of a date-time field provides time zone support.

When TZ is specified, date-time values are converted and displayed in the user's local time, but are stored in *coordinated universal (UTC) time*. This allows users in different time zones to view the data in their individual local times, but still share the same data. Storing values in standardized UTC time makes them easily comparable.

Adabas uses the time zone data taken from the **tz database**, which is also called the *zoneinfo* or *Olson* database. The specific list of time zone names that Adabas supports in any given release can be found in the TZINFO member of the Adabas time zone library (ADA*vrs*.TZ00). For more information about the TZINFO member of the time zone library, read *Supported Time Zones*, in the *Adabas DBA Tasks Manual*.

The TZ option can be specified in field definitions that use the following **date-time edit masks**:

- DATETIME
- TIMESTAMP
- NATTIME
- UNIXTIME

- XTIMESTAMP

You cannot use the TZ option in field definitions that use the DATE, TIME, or NATDATE date-time edit masks because the timezone offsets depend on the presence of both date and time values in the data.

Note that UNIXTIME and XTIMESTAMP fields are by definition based on the UTC; standard conversion routines will perform time zone handling outside of Adabas. In other words, the TZ option has no effect when reading or writing fields with the UNIXTIME or XTIMESTAMP edit mask.

However, when the DATETIME, NATTIME, and TIMESTAMP edit masks are set in the format buffer, the TZ option will convert the times to local time; otherwise they will be converted and returned as UTC times.

For example, if a date-time field is stored in UTC format is February 14, 2009, 16:00 hours, user A in time zone America/New_York will see the field displayed as February 14, 2009, 11:00 hours or 10:00 (UTC time minus 5 or 6 hours, depending on the differences in daylight savings time). Alternately, user G in time zone Europe/Berlin will see the field as February 14, 2009, 17:00 hours (UTC time plus 1).

For information on the conversions between date-time fields defined with the TZ option, read *Conversions Between Date-Time Representations for Fields with the TZ option*, in the *Adabas DBA Tasks Manual*.

Data Compression Options FI and NU

Default data compression is described in the section ***Compression***. At the field level, additional compression can be specified (null suppression option) or all compression can be disabled (fixed storage option).

Null suppression (NU) differs from default compression in that searches on descriptor fields defined with null suppression do *not* return records in which the descriptor field is empty.

Fields defined as fixed format (FI) do not include a length byte and are not compressed. This option actually saves storage space for one-byte fields or fields that are nearly always full (e.g., a field containing the social security number).

Encoding Conversion Option NV

Alphanumeric (A) or wide-character (W) format fields with the NV option are processed in the record buffer without being converted to or from the user.

The field has the characteristics of the file encoding; that is, the default blank:

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

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

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

Long Alpha Option LA

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

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

In Adabas 8 (or later), the **NB (no blank compression) option** can be specified for LA fields to control blank suppression.

LA fields cannot be descriptors but parents of sub/superdesciptors.

LA fields cannot also be defined with the LB field option. To assist you in determining whether to define a field as an LA or an LB field, read ***Comparing LA and LB Fields***, elsewhere in this section.

Large Object Option LB

The large object (LB) option can be specified for some fields to identify them as *large object* fields. LB fields can contain up to 2,147,483,643 bytes (about 2 GB) of data.

The format of an LB field must be "A" (alphanumeric) and its default field length must currently be defined as zero.

LB fields *cannot* be:

- Descriptors or parents of a special (phonetic, sub-, super-, or hyper-) descriptor.
- Defined with the FI or LA options.

To assist you in determining whether to define a field as an LA or an LB field, read ***Comparing LA and LB Fields***, elsewhere in this section.

- Specified in a search buffer or in format selection criteria in a format buffer.

LB fields may be:

- Defined with any of the following options: MU, NB, NC, NN, NU, or NV
- Part of a simple group or a PE group.

The presence of the NB (no blank compression) field option in the LB field definition indicates whether on not Adabas removes trailing blanks in LB fields containing characters.

LB fields containing both binary and character data are supported. An LB field defined with both the NV and NB options can store binary large object data, as Adabas will not modify binary LB fields in any way. The identical LB binary byte string that was stored is what is retrieved when the LB field is read. In addition, because LB fields containing binary values are defined with the NV and the NB options, Adabas will not convert LB field binary values according to some character code page nor will it cut off trailing blanks in LB fields containing binary values.

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

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

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

Commands dealing with LB fields must always be directed to the *base file* of a *LOB file group*. User commands against *LOB files* are rejected.

For information on getting started using LB fields, read *Large Object (LB) Files and Fields*, in *Adabas DBA Tasks Manual*.

Comparing LA and LB Fields

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

Feature	LA Field Behavior	LB Field Behavior
Zero field length specification in format buffers	Two bytes in the corresponding record buffer area are used to store the actual length of the LA field.	Four bytes in the corresponding record buffer area are used to store the actual length of the LB field.
Data record storage	Alphanumeric and wide-character fields are stored within the compressed record. All long values must fit into the same compressed record. The maximum length of simple or spanned data records limits the number and lengths of long values that can be stored. This can be a problem if multiple long values are contained in a record.	Some LB field values (those larger than 253 bytes) are stored offline in a separate large object file (the LOB file) and only references to the LB field values in the LOB file are included in the data record. This allows for storing more long objects for a single data record than using normal or LA fields. However, the performance overhead at runtime and for file maintenance is increased for LB fields because of this behavior. Smaller LB field values (up to 253 bytes) are stored directly in the compressed record. This improves performance for small values, but also limits the number of small LB field occurrences that can be stored in the same compressed record.
Asterisk (*) field length notation in format buffers	Supported for LA fields of any length.	Supported for LB fields of any length.
Maximum length of any stored object does *not* exceed 16,381 bytes	Alphanumeric or wide-character LA field can be used. This avoids the overhead of LB fields, but limits the number of such fields that can be stored in a single record.	Alphanumeric LB field can be used.
Maximum length of any stored object exceeds 16,381 bytes	Not supported.	Supports objects with sizes larger than 16,381 bytes.
So many large objects that they will not fit in a single simple or spanned data record	Not supported.	Supports multiple large objects.
Parent of sub/superdescriptor	Supported.	Not supported.
Parent of sub/superfield	Supported.	Not supported.

MU and PE Options and Field Types

Adabas supports two basic field types: elementary fields and multiple-value fields. An *elementary* field has only one value per record. *Multiple-value* (MU) fields can have 191 up to about 65,534 values, or occurrences, in a single record. The use of more than 191 MU fields or PE groups in a file must be explicitly allowed for a file (it is not allowed by default). This is accomplished using the ADADBS MUPEX function or the ADACMP COMPRESS `MUPEX` and `MUPECOUNT` parameters. Each multiple-value field has a *binary occurrence counter* (BOC) that stores the number of occurrences.

A *periodic (PE) group field* defines consecutive fields in the FDT that repeat together in a record. Like the members of a non-periodic group field, PE members immediately follow the PE group field, have a higher level number than the PE field, and can be accessed both individually and as a group. Each PE has a BOC that stores the number of occurrences.

A periodic group may be repeated 191 or up to about 65,534 times per record and may contain one or more multiple-value fields. The use of more than 191 MU fields or PE groups in a file must be explicitly allowed for a file (it is not allowed by default). This is accomplished using the ADADBS MUPEX function or the ADACMP COMPRESS `MUPEX` and `MUPECOUNT` parameters. Occurrences or values that are not used require no storage space.

Adabas thus supports four field types:

	Single Value per Record	Multiple Values per Record
Single Field	Elementary	MU
Multiple Fields	Group	PE

The actual limit to the number of occurrences of MU fields and PE groups in a file is derived from the maximum data storage record length (the ADALOD MAXRECL parameter), which defaults to the size of the data storage block minus 4.

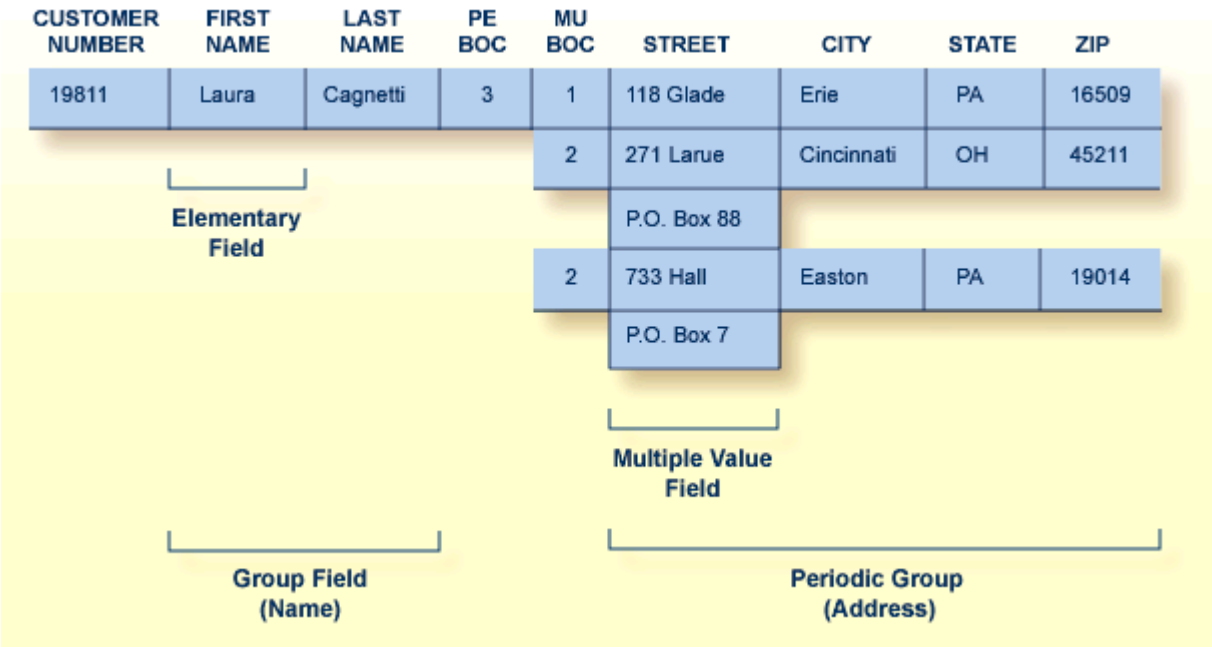
The number of occurrences of each MU field or each PE group in a record can be increased from 191 to about 65,534 using the ADADBS MUPEX function or the ADACMP COMPRESS `MUPEX` and `MUPECOUNT` parameters. However, the actual limit is derived from the maximum Data Storage record length (the ADALOD MAXRECL parameter), which defaults to the size of the Data Storage block minus 4, the device type, and the file type (spanned or unspanned). All MU fields and PE groups and other fields must fit into one compressed record. If you are using spanned records (introduced with Adabas 8), more MU fields and PE groups can be stored.

In addition, subdescriptors and superdescriptor definitions can affect the number of MU fields or PE groups in the record. For example, if a superdescriptor is created as a combination of a PE group and one or more MU fields and the number of occurrences is high, performance and resource problems can occur.

Note: Excessive use of extended MU and PE fields might cause performance and resource problems. These can result in a work storage overflow, resulting in response code 9 (ADARSP009). If this should happen, increase the ADARUN LP size for the database.

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

The following figure illustrates the four field types in a single record structure.



A PE field cannot be nested within another PE group. Nesting an MU field within a PE group, as shown in the figure above, is permitted but complicates programming by introducing a two-dimensional array. It also has implications for data access: when Adabas accesses the periodic group, it returns only the first occurrence of the MU for each occurrence of the PE returned.

The unique characteristic of the periodic group and the reason for choosing the periodic group structure is its ability to maintain the order of occurrences. If a periodic group originally contains three occurrences and the first or second occurrence is later deleted, those occurrences are set to nulls; the third occurrence remains in the third position. This contrasts with the way leading null entries are handled in multiple-value fields. The individual values in a multiple-value field do not retain positional integrity if one of the values is removed.

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

Blank Compression Option NB

The NB option can be used with **LA** and **LB** fields to control blank compression. When specified, the NB option indicates that Adabas should *not* remove trailing blanks for the field; when not specified, Adabas removes trailing blanks when storing an alphanumeric or wide-character field value. If you specify the NB option for a field, you must also specify the NU or NC option for the field; NB processing requires the use of NC or NU as well.

Note: Fields specified without the NB option can lead to differences in the stored and retrieved lengths of the fields. The retrieved length of a non-NB field is likely to be smaller than the length specified for the field when it is stored due to blank compression. This may matter if the value if not really a character string, but rather a binary value that happens to end with the character codes for a blank. Therefore, if you want the stored and retrieved lengths of a field to be the same, use the NB option.

Adabas does not allow the storing of a field with a length of zero unless the field has been defined with the NB option. One way to specify a length of zero for a large object (LOB) field L1 would be, for instance, via `FB='L1L,4,B,L1,*,A.'` and `RB=x'00000000'`. Adabas accepts this construction if field L1 has been defined with the NB-option (for example, `L1,0,A,LB,NU,NB`) and will return the length of zero if the field is later read. However, if the field has been defined *without* the NB option (for example, `L1,0,A,LB,NU`), Adabas rejects the attempt to store the zero-length value with response code 52 (ADARSP052), subcode 2 . Without the NB option, an empty field should be stored as a single blank (for example, `FB='L1L,4,B,L1,*,A.'` and `RB=x'0000000140'`).

For examples of this, read *Read Operations, Length Indicators, and the NB (No Blank Compression) Option*, in the *Adabas Command Reference Guide*.

SQL Compatibility Options NC and NN

Special data definition options are included in Adabas to accommodate Software GmbH's mainframe Adabas SQL Gateway (ACE) and other structured query language (SQL) database query languages that require SQL-compatible null representation.

A field designated with the NC (not counted) option may contain a null value that satisfies the SQL interpretation of a field having no value. An NC field containing a null means that *no field value has been entered*; that is, the field's value is not defined.

This undefined state differs from a null value assigned to a non-NC field for which no value has been specified: a non-NC field's null means the value in the field is either zero or blank, depending on the field's format.

The NN (not null) option can be specified only for NC-defined fields. It indicates that an NC field must always have a value defined; it cannot contain an SQL null. This ensures that the field cannot be left undefined when a record is either created or updated. The field value may be zero or blank, however.

Special Fields and Descriptor Fields

The FDT indicates whether a field is a parent field for a collation descriptor, subfield, superfield, subdescriptor, superdescriptor, hyperdescriptor, or phonetic descriptor. Information about any special fields and descriptors (collation descriptors, subdescriptors, subfields, superdescriptors, superfields, phonetic descriptors, and hyperdescriptors) in a file is maintained in the special descriptor table (SDT) part of the FDT.

```
SPECIAL DESCRIPTOR TABLE

        I      I        I        I                      I                I
  TYPE  I NAME I LENGTH I FORMAT I       OPTIONS        I   STRUCTURE    I
        I      I        I        I                      I                I
 -------I------I--------I--------I----------------------I----------------I
        I      I        I        I                      I                I
  SUPER I  H1  I    4   I    B   I DE,NU                I AU (  1 -  2)  I
        I      I        I        I                      I AV (  1 -  2)  I
  SUB   I  S1  I    4   I    A   I DE                   I AO (  1 -  4)  I
  SUPER I  S2  I   26   I    A   I DE                   I AO (  1 -  6)  I
        I      I        I        I                      I AE (  1 - 20)  I
  SUPER I  S3  I   12   I    A   I DE,NU,PE             I AR (  1 -  3)  I
        I      I        I        I                      I AS (  1 -  9)  I
        I      I        I        I                      I                I
  PHON  I  PH  I        I        I                      I PH =PHON(AE)   I
        I      I        I        I                      I                I
  COL   I  Y1  I   20   I    W   I DE                   I CDX  8,PA      I
  COL   I  Y2  I   12   I    A   I DE,NU,PE             I CDX  1,AR      I
        I      I        I        I                      I                I
        I      I        I        I                      I                I
 ---------------------------------------------------------------------------
```

Along with the name, length, format, and specified options of each special field and descriptor, this table provides the following information:

Column	Explanation	
TYPE	COL	Collation descriptor
	HYPER	Hyperdescriptor
	PHON	Phonetic descriptor
	SUB	Subfield/subdescriptor
	SUPER	Superfield/superdescriptor
STRUCTURE	The component fields and field bytes of the sub-, super-, or hyperdescriptor. Phonetic descriptors show the equivalent alphanumeric elementary fields. Collation descriptors show the associated collation descriptor user exit and the name of the parent field.	

This section describes the special fields and descriptors:

- Collation Descriptor
- Hyperdescriptor
- Phonetic Descriptor
- Subfield / Superfield
- Subdescriptor
- Superdescriptor

Collation Descriptor

An alphanumeric or wide-character field can be defined as a parent field of a collation descriptor. A collation descriptor is used to sort field values in a special user-defined sequence. The LF command reports the collation descriptor field information.

A collation descriptor is assigned a collation descriptor user exit (1-8) which encodes the collation descriptor value and decodes it back to the original field value. The ADARUN parameter CDXnn is used to specify collation descriptor user exits.

Hyperdescriptor

The hyperdescriptor option can be used to generate descriptor values based on a user-supplied algorithm. Up to 31 different hyperdescriptors can be defined for a single physical Adabas database. Each hyperdescriptor must be named by an appropriate HEXnn ADARUN statement parameter in the job where it is used.

With hyperdescriptors, fuzzy matching is possible; i.e., retrieving data based on *similar* rather than on *exact* search criteria. Hyperdescriptors allow multiple virtual indexes, meaning that several different search index entries can be made for a single data field.

Hyperdescriptors can be used to implement n-component superdescriptors, derived keys, or other key constructs. Using hyperdescriptors, it is possible to develop applications that are simpler and more flexible than applications based on a strictly normalized relational structure.

One application area for hyperdescriptors is name processing. For example, the name SCHROEDER could be stored not only with the index SCHROEDER itself, but also with the virtual indexes SCHRODER, SCHRADER, or any other variation of the name. Thus, although only the name SCHROEDER is physically stored in the data area of the database, multiple search indexes exist to the data. If, subsequently, a search is made for the name SCHRODER, the record SCHROEDER will be found.

A more sophisticated application area for hyperdescriptors is fingerprint matching, in which typical characteristics of fingerprints can form the basis of a fuzzy matching algorithm; i.e., the original fingerprint is stored in the database, but any number of search indexes can be made to the fingerprint, based on an algorithm that allows small-scale deviations from the original.

Phonetic Descriptor

A phonetic descriptor may be defined and used to search for all records that contain similar phonetic values. The phonetic value of a descriptor is determined by an internal algorithm based on the first 20 bytes of the field value with only alphabetic values being considered (numeric values, special characters and blanks are ignored).

Subfield / Superfield

A portion of a field (subfield) or any combination of fields (superfield) may be defined as an elementary field (read ***MU and PE Options and Field Types***, elsewhere in this guide). Subfields and superfields may be used for read operations only. They may only be changed by updating the original fields.

Subdescriptor

A subdescriptor is part of a single field used as a descriptor. The field from which the subdescriptor is derived may or may not be an elementary descriptor (read ***Descriptor Options DE, UQ, and XI***, elsewhere in this guide. If a search criteria involves a range of values contained in the first n bytes of an alphanumeric field or the last n bytes of a numeric field, a subdescriptor may be defined using only the relevant bytes of the field. A subdescriptor allows you to increase the efficiency of a search by specifying a single value rather than a range of values.

For example, if the first two bytes of a five-byte field refer to a geographical region and you want to retrieve all records for region 11 without using a subdescriptor, you would have to search for all records in the range 11000-11999. If you define a subdescriptor comprising the first two bytes of the field, you could search for all records with 11 in the subdescriptor.

Superdescriptor

A superdescriptor combines all or parts of 2-20 fields. The fields from which the superdescriptor is derived may or may not be elementary descriptors. When search criteria involve values for a combination of fields, using a superdescriptor is more efficient than using a combination of several elementary descriptors.

For example, to search for customers by last name within regions, you could create a superdescriptor by combining the first two bytes (i.e., the geographical region indicator) of the five-byte customer number field and the entire customer last name field.

For complete information about defining superdescriptors, read *SUPDE: Superdescriptor Definition* in the ADACMP documentation found in *Adabas Utilities Manual*.

Parallel Participant Table (PPT)

The parallel participant table (PPT) in the Associator tracks the active Adabas nucleus (or nuclei in a clustered environment) that manages a particular database. When the nucleus becomes active, it registers itself in the PPT.

The PPT comprises 32 blocks: one per possible nucleus in a cluster. A single (noncluster) nucleus always occupies the first block of the table and the nucleus ID is always zero (0).

The PPT is created by the ADADEF utility for a new database or by the ADACNV utility when an existing database, that does not yet have a PPT, is converted to the current version. Its location is stored in the GCB. PPT entries can be modified by the nucleus or by the ADARES PLCOPY/CLCOPY functions. The DPPT operator command displays the PPT contents; the ADACHK PPTPRINT utility functions print out the PPT contents in interpreted and dump format.

If the nucleus terminates normally, the PPT entry is retained and marked as inactive. If the nucleus terminates abnormally, its entry in the PPT remains unchanged until the recovery process is complete.

Information about protection log (PLOG) data sets, command log (CLOG) data sets, and the Work data set of a nucleus is logged in its PPT entry. As each PLOG and CLOG becomes active, it is registered in the PPT. Once all PLOGs or CLOGs have been copied after session end, the PPT entry is marked accordingly. ADARES and the nucleus both check the PPT. Sequential PLOGs and CLOGs are neither recorded nor maintained in the PPT.

The ADADBS RESETPPT utility function resets the PPT blocks on the Associator data set, but should only be used when extraordinary measures are necessary. For more information, read *RESETPPT: Reset PPT Blocks*, in the *Adabas Utilities Manual*

The PPT is used in the following situations:

- It is used when ADARES PLCOPY or CLCOPY is copying all active protection logs or command logs, respectively. ADARES only pulls the PLOG and CLOG information from the PPT if the PLCOPY or CLCOPY is for a cluster database; otherwise, it uses the JCL provided in the PLCOPY/CLCOPY job.
- It is used to warn when a nucleus session leaves the Work, PLOG, or CLOG data set non-empty and the next nucleus (i.e., the restart nucleus) starts with a different Work, PLOG, or CLOG data set.

This section consists of the following topics:

- Initialization Processing

- Dynamic Allocation

Initialization Processing

The PPT exists for both cluster and noncluster nuclei. It is used to determine which PLOGs, CLOGs, and Work data sets have been last used by each nucleus. The following discussion applies to the session start of noncluster nuclei.

The starting nucleus uses the PPT to determine if any PLOGs or CLOGs still contain uncopied data from a previous session. If they do, the nucleus calls UEX2 or UEX12, if specified, to submit ADARES PLCOPY or CLCOPY jobs that will copy these PLOGs and CLOGs (or merge them, if they were written by cluster nuclei). If the starting nucleus shares PLOG data sets with a previously run cluster nucleus and one of the PLOGs contains uncopied data, the nucleus either reports an error (if PLOGRQ=FORCE) or marks the PLOG as empty after giving UEX2/12 a chance to copy off the protection data (otherwise).

If the first block of the PPT has been used by a cluster nucleus, the starting noncluster nucleus gives UEX2/12 a chance to copy off any uncopied PLOGs and then overwrites the PPT entry with its own PLOG and CLOG data set names, printing a warning message if the previously recorded PLOGs have not been copied. However, if PLOGRQ=FORCE has been specified, the data set names of the uncopied PLOGs cannot be overwritten in the PPT and the session start fails.

In any case, once the starting nucleus prepares its own PLOGs and CLOGs (specified in its JCL) for logging, it inserts their data set names into the PPT and overwrites any previously used PLOG or CLOG data set names that may have been recorded there.

If uncopied PLOG or CLOG data sets remain, they can still be copied with the ADARES utility, by specifying the data sets in the ADARES JCL. Uncopied PLOGs from a cluster session can be copied and merged using the NOPPT option. Uncopied CLOGs from a cluster session can be copied and merged using the CLOG MERGE function.

Dynamic Allocation

The nucleus always writes its protection and command log data to the Work, PLOG, and CLOG data sets specified in its JCL. In a noncluster environment, the ADARES PLCOPY and CLCOPY functions always read from the PLOG and CLOG data sets specified in their JCL. In addition, the nucleus and the ADARES utility use dynamic allocation (i.e., access the data sets via the names recorded in the PPT, rather than via the JCL) to perform consistency checks if the PPT indicates that the old data sets still contain needed information.

Spanned Records

With Adabas 8, records can be spanned in a database. In the database, the logical record is split into a number of physical records, each part fitting into a single Data Storage (DS) block. Spanned records may be segmented at the field or byte level. The resulting physical records are each assigned individual ISNs. The first physical record is called the *primary record* and contains the beginning of the compressed record and is assigned a *primary ISN*. The remaining physical records are called *secondary records* and contain the rest of the data of the logical record. Secondary records are assigned *secondary ISNs*. These ISNs do not affect the user ISNs assigned when using the N2 command or the ISNs used when using the I option of the L1 command. If spanned records are used, a secondary address converter is used to map the secondary ISNs to the RABNs of the Data Storage blocks where the secondary records are stored.

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

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

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

Note: Spanned record support must be explicitly allowed for a file. You can do this using the ADADBS RECORDSPANNING function or the `SPAN` parameter of ADACMP COMPRESS. For more information, read the *Adabas Utilities Manual* documentation for the ADADBS and ADACMP utilities.

This section covers the following topics:

- Spanned Record Structure
- Allowing Spanned Records in Files
- Secondary Record Segmentation
- Padding Factors
- Spanned Record ISN Use
- ADARUN Parameters Affected
- Reporting on Spanned Records

- Securing Spanned Records

Spanned Record Structure

A spanned logical record is comprised of one or more physical records, including a single primary record and one or more secondary records. The number of records that comprise a spanned record is limited. The Adabas nucleus allows up to five physical records (one primary record and four secondary records) in a spanned record.

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

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

Allowing Spanned Records in Files

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

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

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

Secondary Record Segmentation

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

Padding Factors

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

Spanned Record ISN Use

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

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

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

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

ADARUN Parameters Affected

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

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

For complete information on these and other ADARUN parameters, read *Adabas Initialization (ADARUN Statement)*, in the *Adabas Operations Manual*.

Reporting on Spanned Records

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

Securing Spanned Records

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

4 Using Adabas

- Accessing a Database from Programs 60
- Maintaining Database Integrity 68

Generally, Adabas uses between 10% and 50% of the data processing resources (disk storage, CPU time, elapsed processing time) used by other database management systems. Since fewer hardware facilities are used, more can be accomplished with less. Very large online applications using several terabytes of data have been successfully implemented, with thousands of terminal workstations, and with the response times and the cost of much smaller systems.

Accessing a Database from Programs

Adabas access is field-oriented: user programs access and retrieve only the fields they need. User program statements invoke Adabas search and retrieval operations automatically.

- Direct Call Interface
- Complex Searches
- Access Methods
- Using Triggers and Stored Procedures
- Universal Encoding Support (UES)

Direct Call Interface

Adabas provides a powerful and flexible set of direct call commands for performing database operations. Adabas direct call commands provide a direct interface to the Adabas database when Natural or another fourth-generation database language is not being used.

The commands can be categorized by function:

- database query
- read (Data Storage or Associator)
- database modification
- logical transaction processing
- special commands

Database Query Commands (Sx)

Database query commands (S1/S4, S2, S5)) search for and return the ISNs of specified records or record groups according to specified search criteria. Other commands in this category (S8, S9) sort the resulting ISN lists in preparation for later operations.

The ISN lists resulting from any S*x* command may be saved on the Adabas Work data set for later retrieval during the user session.

In most cases, these commands do not actually read the database; ISNs are read directly from the Associator's inverted lists. Options allow the ISN's record to be placed in hold status to prevent

it from being updated by other programs until the record is released; if desired, additional field values contained in the first ISN's record can be read from Data Storage.

Read Commands (Lx)

The L1 through L6 commands are used to read actual records from Data Storage. Depending on the specified command and its options, records are read individually

- in the sequence in which they are stored;
- in ascending or descending ISN sequence;
- in the order of an ISN list created by one of the database query commands; or
- in logical sequence according to a user-specified descriptor.

Read *Sequential Access*, elsewhere in this guide, for more information about sequential access methods.

A hold option allows the database records to be locked until released by a separate command or at transaction end.

The L9 and LF commands read information directly from the Associator inverted lists or field definition tables (FDTs), returning either the inverted list values for a specified descriptor or the field definitions for a specified file in the database.

Database Modification Commands (A1, E1, N1/N2)

Database modification commands (A1, E1, and N1/N2) change, delete, or add database records and update the related Associator lists accordingly. ISNs can be assigned to new records either by the user or by Adabas.

The inverted lists and the address converter are automatically maintained by Adabas. When the user supplies a new value for a descriptor, either in a new record or as part of a record update, Adabas performs all necessary maintenance of the inverted lists. When a new record is added or a record is deleted, the address converter is appropriately updated. These operations are completely transparent to the user.

Logical Transaction Control Commands (ET/BT)

An Adabas logical transaction defines the logical start (BT) and end (ET) of the database operation being performed. If the user operation or Adabas itself terminates abnormally, these commands provide the capability for restarting a user, beginning with the last unsuccessfully processed transaction. ET/BT commands

- define the transaction start and end;
- restore pretransaction conditions if a situation occurs that prevents successful completion of the transaction; and

- read program-specified user data written during the transaction sequence.

Programs that use these commands are called ET logic programs. Although not required, we recommend that you use ET logic. Read ***Transaction Logic***, elsewhere in this guide, for more information.

Special Commands

Special commands perform many of the housekeeping functions required for maintaining the Adabas database environment. Commands in this group

- open (OP) and close (CL) a user session (but do not control a transaction);
- write data protection information, user data, and checkpoints (C1, C3, C5); and
- set (HI) and release (RI) record hold status.

In addition, the RC command releases one or more command IDs currently assigned to a user, or deletes one or all global format IDs.

The RE command reads user data previously stored in an Adabas system file by C3, CL, or ET commands.

Complex Searches

In many large database systems, the time required to process complex searches is often excessive. Adabas efficiently solves this problem. Many Adabas applications are currently in use with up to 150 complex selection criteria created dynamically. Data is retrieved immediately from files with more than 50 million records.

Multifile Searching

It is often necessary to perform a multifile search in order to resolve an inquiry. Multifile searching can be accomplished using multiple search commands, physically coupled files, or soft file-coupling. Read ***Coupled Files***, elsewhere in this guide, for information about coupled files.

Multiple search commands may be used in which a value retrieved in one command is used as the search value for the next command. This process is not restricted to two files.

Multi-index Searching

Fuzzy matching (i.e., retrieving data based on *similar* rather than on *exact* search criteria) can be implemented using hyperdescriptors. Hyperdescriptors allow multiple virtual indexes, meaning that several different search index entries can be made for a single data field. Read ***Hyperdescriptor*** , elsewhere in this guide, for information about hyperdescriptors.

Access Methods

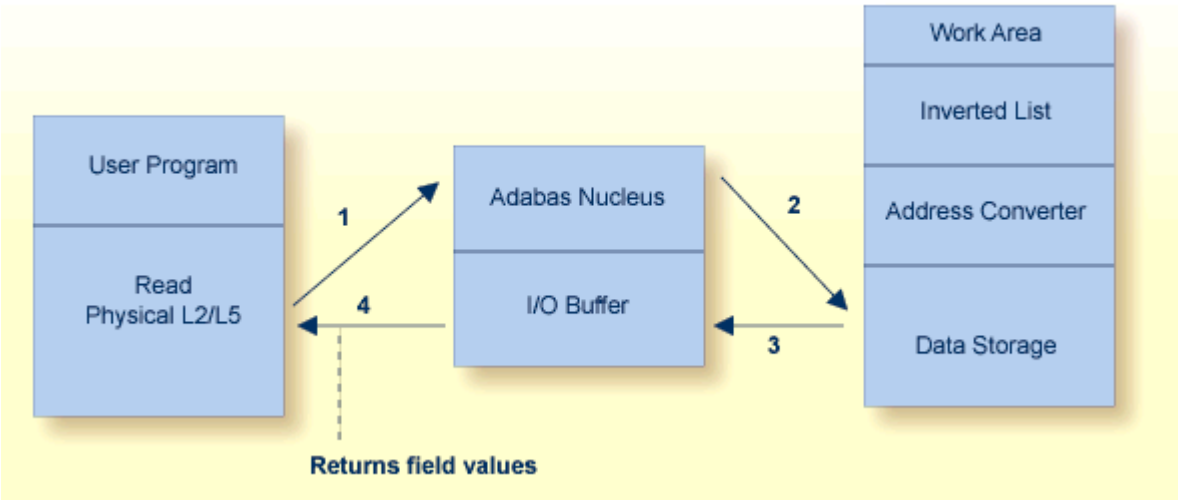
Adabas supports both sequential and random access methods. Different calls use different Adabas access paths and components; the most efficient method depends on the kind of information you want and the number of records you need to retrieve.

- Sequential Access
- Random Access
- Random Access Using the Adabas Direct Access Method (ADAM)

Sequential Access

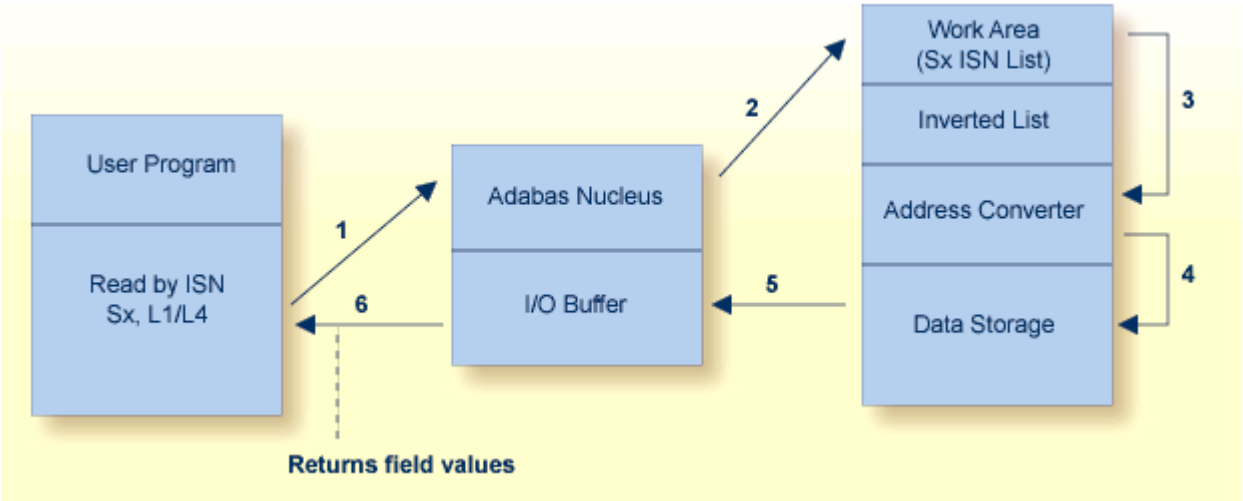
Physical sequence retrieves every record in a file in the order the records are stored in Data Storage. You can limit the fields within each record for which values are to be returned. You can also specify a starting ISN: the sequential pass then begins at the first record physically located after the record identified by the specified ISN.

Adabas bypasses the Associator and goes directly to Data Storage, reading the first data block (or the first record following a specified ISN) and continuing in consecutive sequence until the last block is read. Physical sequence is the fastest way to process a large volume of records.

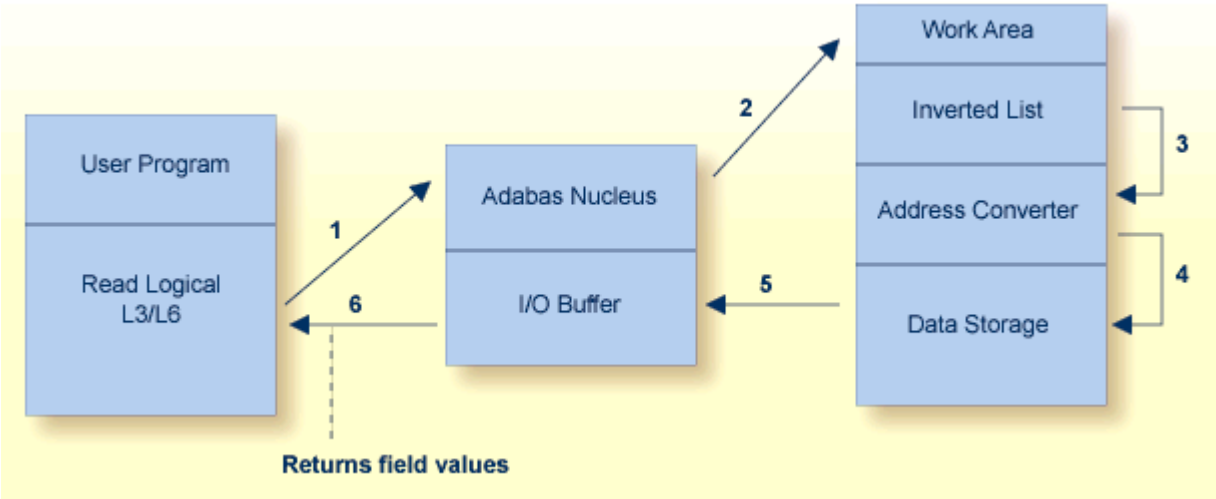


ISN sequence retrieves records in ISN order. Adabas uses database query commands (S*x*) to build and sort ISN lists, which can then be read using L1/L4 commands with the GET NEXT option.

When reading, Adabas uses the address converter to find the RABNs of each ISN in the list and then reads and returns the records from Data Storage.



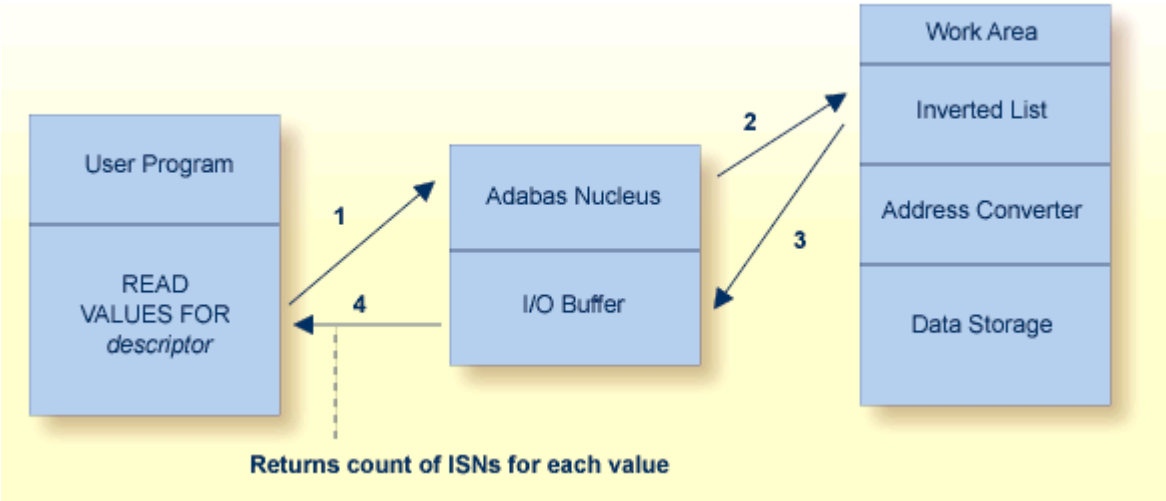
Logical sequence retrieves records by descriptor value. Adabas finds the value(s) in the inverted list, uses the address converter to find the RABNs of the ISNs related to the value, and retrieves the records from Data Storage.



Reading in logical sequence retrieves all the records related to a single value or a range of values of the specified descriptor. It returns the records sorted in ascending/descending order by descriptor value, and in ascending/descending ISN order within each value. You can specify a starting and ending value. You can also specify the field(s) for which values are returned. Read logical is useful when you want the returned records sorted on a particular field.

Adabas provides a special read command (L9) to determine the range of values present for a descriptor, and the number of records that contain each value. Such a retrieval is called a histogram.

The L9 command does not require any access to data records, only to the inverted lists stored in the Associator.



Random Access

Adabas uses the S1/S2/S4 commands to select records that satisfy a *search criterion*: the count of the records found and a list of their ISNs is returned. The S1/S4 commands return the ISNs in ascending sequence; the S2 command allows you to specify a sort sequence for the returned ISNs.

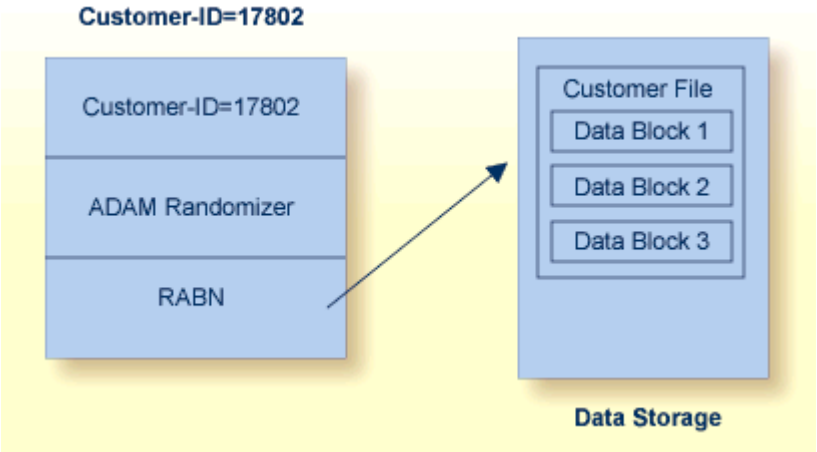
The search criterion may comprise

- one or more fields in a single file;
- fields contained in two or more physically coupled files; or
- search, read, and internal list matching based on the soft coupling feature.

A search criterion may contain one or more fields that are not defined as descriptors. If nondescriptors are used, Adabas performs read operations to determine which records to return to the user. If only descriptors are used within the search criterion, Adabas resolves the query by using the Associator inverted lists; no read operation is required.

Random Access Using the Adabas Direct Access Method (ADAM)

The Adabas direct access method (ADAM) improves random access performance on a particular descriptor field in a file. ADAM uses the field value to compute the relative block address (RABN) for record storage. The ADAM descriptor may also be used like any other descriptor within a selection criterion, and for logical sequential processing as well. This option may be selected for any given file when the file is loaded into the database.



Using Triggers and Stored Procedures

The Adabas triggers and stored procedures facility is an integral part of Adabas; however, Natural (read *Natural Application Development Environment* , elsewhere in this guide) is required to use the facility. The online Trigger Maintenance facility can be accessed using the **Adabas Online System** add-on product.

A procedure is a Natural subprogram that is written and tested using standard Natural facilities.

- A *trigger* is a procedure that is executed automatically by Adabas when a specified set of criteria is met. The set of criteria is determined for each command sent to Adabas and is based on the target file number and optionally the command type and/or field. The command type refers to the find, read, store, update, and delete commands. The field must be in the corresponding format buffer of the command.
- A *stored procedure* is executed by Adabas, but is invoked directly by a special user call from any of a number of applications that use it. Storing programs that are used by multiple clients in an Adabas file on the server reduces the amount of data traffic to and from the server.

The same types of parameters are passed to the subprogram whether it is a trigger or a stored procedure.

The Adabas facility for triggers and stored procedures allows you to implement and maintain both types of procedures. It resides within Adabas and provides an extension to an application. It can be used to

- implement various security and auditing features for an application; and
- provide a consistent, central environment where data can be verified or manipulated, either manually by the application or automatically by Adabas when triggers are defined.

Universal Encoding Support (UES)

Adabas provides facilities for converting data bidirectionally between different architectures. This makes it possible to support access to the mainframe database through the TCP/IP protocol from web-based applications or from PC-based applications such as Natural for Windows.

A database can use these UES facilities if activated with ADADEF.

Adabas performs character data conversion for fields with alphanumeric (A) and wide-character (W) formats. It supports a wide range of character sets or code pages, including Unicode, double-byte character sets (DBCS), and multiple-byte characters sets (MBCS).

- Alphanumeric fields are extended to support wide-character data by defining encoding keys on both the database and file levels: the file level encoding takes precedence over the database encoding. The encoding specifies the format in which the data is to be stored. It is also used as the default format in which data is exchanged with a local user. The file encoding must belong to the EBCDIC group.
- Wide-character fields are similar to alphanumeric fields in that encoding defaults are defined on both the database and file levels: the file encoding takes precedence over the database encoding. Unicode is the default encoding.

In addition, Adabas supports the conversion of numeric data, such as low-order-byte-first or IEEE floating point numbers.

At the client application, the Adalink code determines the default architecture (for example, ASCII, byte-swap, and IEEE floating point). The client application can override this default by specifying different encodings or architecture using the session open (OP command).

A number of utilities provide for special encoding and architecture settings.

To ensure round-trip compatibility between architectures and encodings, a file encoding should be chosen that has the same or a superset of characters versus the user encoding. For example, US-EBCDIC and ISO-8850-1 have the same character set.

Collation descriptors may be defined for alphanumeric or wide-character format fields. The descriptor values are obtained algorithmically by calling a collation exit, for example to produce a culturally correct, sorted key (that is, dictionary order).

Maintaining Database Integrity

Adabas provides facilities to ensure the logical consistency of data in a competitive updating environment and when a user or Adabas session is interrupted.

Facilities are available for both online and traditional batch update. For online, transaction-oriented processing, Adabas ensures that the database is free of incomplete transactions. For batch mode updating, Adabas ensures restart in the event of failure by writing checkpoints and backing out/regenerating updates.

- Transaction Logic
- Distributed Transaction Processing
- Competitive Updating
- Timeout Controls
- Backout, Recovery, and Restart
- Extended Error Handling and Message Buffering

Transaction Logic

Adabas data protection, recovery, and user restart are based on the concept of a *logical transaction*: the smallest unit of work (as defined by the user) that must be performed in its entirety to ensure that the information contained in the database is logically consistent.

A logical transaction may comprise one or more Adabas commands that together perform the database read/update required to complete a logical unit of work. A logical transaction begins with the first command that places a record in hold status and ends when an ET (end transaction), BT (back out transaction), CL (close), or OP (open) command is issued for the same user.

The OP (open) or RE (read ET data) commands can be used to retrieve user restart data stored by the C3, CL, or ET command. This data is also written to the Adabas data protection log with each checkpoint written by the transaction and can be read using the ADASEL utility.

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 are physically applied to the database, regardless of subsequent user or Adabas session interruption.

Updates performed during transactions for which ET commands are not successfully executed are backed out, either manually by issuing the BT command or automatically by the autobackout routine.

Distributed Transaction Processing

Adabas incorporates nucleus functions to support the execution of global database transactions in distributed environments; that is, across multiple local or remote databases and/or system images in parallel.

A two-phase commit protocol ensures that all database management systems (DBMSs) that participate in processing the global transaction (that is, resource managers or RMs) either commit or roll back their local parts of a transaction as a whole. In the first phase, the coordinating component (a transaction manager or TM) prepares all involved RMs for the commit. Only when the first phase is successful does the TM instruct the RMs to commit (second phase).

Adabas functions in this scenario as an RM. **Adabas Transaction Manager**, an Adabas add-on product, functions as the coordinator within operating system images and, with the help of Entire Net-Work, across system images.

The new protocol also integrates Adabas with other DBMSs. It is transparent to existing application systems and to Natural.

Included in Adabas is a CICS-controlled interface that conforms to the CICS Resource Manager Interface (RMI). It issues the appropriate Adabas commands to coordinate with the two-phase commit protocol.

Competitive Updating

Competitive updating is in effect when two or more users (in multiuser mode) are updating the same Adabas file(s). The Adabas facilities used to ensure data integrity in a competitive updating environment include the following features:

record hold/release, avoidance of resource deadlock, exclusive control updating, and shared hold updating.

- **Record Hold and Release:** The Adabas record hold facility ensures that a record will not be updated by more than one user at a time. A user can put the record in hold status (that is, the ISN of the record is put in the hold queue) using the commands S4 (find with hold), L4/L5/L6 (read with hold), A1/E1 (update/delete) with the hold option specified, N1/N2 (add record with hold), or HI (hold record).

  If a record requested for hold status is already being held by another user or utility, the user issuing the record hold command is put in wait status until the record becomes available, at which time Adabas reactivates the command. You may request that a response code be returned if you do not want to be put in wait status.

  Records in hold status can be accessed (found and read) by users who do not seek to hold the record.

Records in hold status are released by issuing the ET or CL commands. Options are available with ET and BT commands to release records selectively. The CL command releases all records in hold status for the issuing user.

- **Avoiding Resource Deadlock:** Resource deadlock occurs when two users are each waiting for a record held by the other user. Adabas protects against such a user deadlock situation by detecting the potential deadlock and returning a response code to the second user after putting the first user in wait status. This occurs if the two users are serviced by the same nucleus (a non-cluster nucleus or the same cluster nucleus). But even in a single nucleus, a deadlock might not be detected if an ISN involved in the deadlock is being held as a shared resource by more than one user. For more information, read about **shared hold status**.
- **Shared Hold Status:** Shared hold status can be used to lock data records in shared mode, rather than in exclusive mode. This allows your database users to read the same record in parallel transactions, but ensures that no one can update the record concurrently.

  Using shared hold status, your users can protect large object values from concurrent updates without locking out other users who may need to read the same LOB value or other LOB values in the same record. It also allows your users to protect the records they read against concurrent updates for specific periods of time:

  - For the duration of the read command;
  - When the next record in a sequence is read;
  - When the user's transaction ends;
  - Indefinitely.

  Options

  - Option C puts the record in shared hold status for the duration of the read operation. It ensures that the version of the record being read has been comitted by the last updater. This option is available for the L4, L5, L6 and S4 commands. For the S4 command, the shared hold remains in place for the duration of the read operation.
  - Option Q puts the record in shared hold status until the next record in the read sequence is read or the read sequence or transaction is terminated, whichever happens first. It ensures that the record being read cannot be updated concurrently until the next record in the sequence is read (or the transaction is terminated). This option is available for the L4 (when command option 2 is set to "N"), L5, L6 and S4 commands.
  - Option S puts the record in shared hold status until the end of the transaction. It ensures that the record being read cannot be updated concurrently until the transaction is terminated. This option is available for the HI, L4, L5, L6, RI and S4 commands.
  - Option H keeps a record in shared hold status indefinitely (until the next ET or BT command). This option is available for the BT and ET commands. Records in shared hold status at the time of the BT or ET command are kept in shared hold status beyond the end of the transaction until another ET or BT command is issued (without this H option or the prefetch or multifetch

options). Any records in exclusive control are also changed to shared hold status beyond the end of the transaction.

If the same record is placed in shared hold status more than once (using the C or S options or the Q option for different read sequences), it stays in shared hold status until all of the specified hold lifetimes have expired.

For more information about these command options and their detailed functioning in each command, read about the individual commands in *Commands*, in the *Adabas Command Reference Guide*.

For complete information about shared hold updating, read *Shared Hold Status*, in the *Adabas Command Reference Guide*.

- **Exclusive Control Updating:** Users who use logical transaction commands (ET/BT) are called ET logic users.

  Alternatively, a user can request exclusive control of one or more Adabas files for the duration of the user session. If the file or files for which exclusive control is requested are not already opened for update by another user or utility, exclusive control is granted and the user becomes an exclusive update (EXU) user. Adabas treats EXU users as non-ET logic users.

  Adabas does not place an ISN in hold status for EXU users. Adabas disables hold logic processing for files being updated under exclusive file control.

  Instead of using ET commands, EXU users can request checkpoints to act as reference points; for example, updates applied after a checkpoint can be removed.

For complete and detailed information about competitive updating, read *Competitive Updating*, in the *Adabas Command Reference Guide*

Timeout Controls

Adabas times out:

- transactions that exceed a specified limit; and
- users who are inactive for a specified amount of time.

Transaction Time Limit

Adabas provides a transaction duration time limit for ET logic users. The time limit is set with the ADARUN TT parameter; an override for a specific user can be set using the OP command.

If a transaction exceeds the prescribed limit, Adabas generates a BT (back out transaction) command to remove all the updates performed during the transaction and release all held records. The user can then either repeat the backed out transaction from the beginning or begin another transaction.

Non-Activity Time Limit

All users are subject to a non-activity time limit; different limits can be set for different user types and for specific users within each user type.

If a user exceeds the prescribed limit and the user is

- an ET logic user, Adabas backs out the current transaction, releases all held records and command IDs, and deletes the user's file list.
- an EXU user, Adabas deletes the user's file list and releases all command IDs. The user loses his EXU user status and becomes an access-only user.
- an access-only user, Adabas deletes the user's file list.

Backout, Recovery, and Restart

Backout, recovery, and restart may be required when a user or Adabas session is interrupted due to a timeout (read ***Timeout Controls***, elsewhere in this guide); a program error when Adabas determines that a transaction cannot be completed successfully; an Adabas, hardware, or operating system failure; or a power failure.

A *user session* is a sequence of Adabas calls optionally starting with an open (OP) command and ending with a close (CL) command. A *user* is either a batch mode program or a person using a terminal. The uniqueness of each user is assured by the user ID, a machine, an address space, and a terminal ID.

An *Adabas session* starts when Adabas is activated and continues until Adabas is terminated. During this time, the Adabas nucleus creates a sequence of protection entries in exact historical sequence reflecting all modifications made in the database. The sequence of protection entries is written to the Work data set (part 1) and to a protection log in blocks. Each block contains the nucleus session number, a unique block number, and a time stamp.

User Program Error

A user program that is processing a transaction can detect that the transaction cannot be completed successfully. In this case, a BT (back out transaction) command is used to remove or back out the incomplete transaction.

If a user program error causes logical damage to the database, it may be necessary to recover the affected files using the ADASAV and ADARES utilities.

Adabas, Hardware, or Operating System Failure

After any failure that causes the Adabas nucleus to terminate abnormally, an automatic procedure is executed when Adabas is reactivated to bring the database to a physically and logically valid status. All partially executed update commands are reset; all incomplete transactions are backed out.

The automatic procedure comprises three steps: repair the database, autorestart, and autobackout:

- Database repair modifies the database to the status it would have had if a buffer flush had just been completed at the time of failure. That is, all blocks in the database are at a status that enables the nucleus to perform normally.
- Autorestart backs out updates of single update commands that were partially executed when the system failed. It resolves internal inconsistencies in the database and ensures physical integrity.
- Autobackout, which is performed only for ET logic users, backs out updates of user transactions that were partially executed when the system failed. Adabas performs an internal BT (back out transaction) followed by autorestart, and then informs the user that the last transaction has been backed out.

The autobackout routine is executed at the end of an ET session that was terminated with HALT. It is also executed automatically at the beginning of the next Adabas session to remove any updates performed within transactions that did not complete successfully.

After autobackout execution, the database contains updates only from logically complete transactions.

Note: ET users can manually back out an incomplete transaction at any time by issuing the BT (back out transaction) command. Read *Maintaining Database Integrity*, elsewhere in this guide.

If an Adabas, hardware, or operating system failure results in physical damage to the database, it may be necessary to recreate the database using the ADASAV and ADARES utilities.

Power Failure

Depending on the hardware, a power failure during an I/O operation may damage the Adabas blocks that were being processed. This damage cannot be detected during autorestart and therefore can result in problems later such as unexpected response codes or lost database updates.

If the ADARUN IGNDIB=YES parameter is set, the autorestart routine checks whether a buffer flush was active when the session interruption occurred. If a buffer flush was in process, the autorestart shuts down and Adabas alerts the user to the potential problem and includes a list of the files being updated when the buffer flush was in process. The DBA must then determine whether a power failure occurred.

If the cause of a session interruption

- *is* a power failure, we strongly recommend recovering the affected files using the ADASAV and ADARES utilities.
- is *definitely not* a power failure and the integrity of the information on the output hardware can be guaranteed, the database can be reactivated immediately. Database recovery is not necessary.

Extended Error Handling and Message Buffering

The error handling and message buffering facility helps implement 24X7 operations by analyzing and recovering from certain types of errors automatically with little or no DBA intervention. It also generates additional information so that the error can be diagnosed by the user and by Software GmbH.

A wrap-around message buffer collects Adabas messages for later review by Adabas Online System in case online access to the console or to DDPRINT messages becomes unavailable. The buffer aids problem analysis and performance tuning.

The error handling functions of the facility can be invoked from the operator console or from Adabas Online System.

User exits and hyperdescriptor exits that are essential to the operation of the Adabas nucleus can be marked as critical (the default) or not:

- a *critical* user exit is not affected by the error handling and message buffering facility: an abnormal termination in it causes the Adabas nucleus to terminate abnormally as well.
- for a *notcritical* user exit, the facility maintains an active Adabas nucleus, optionally refrains from invoking that exit, takes a dump of the nucleus at the point when the exit failed, and issues messages to the system log to inform the DBA of the problem. The DBA can then examine the diagnostic information, use it to resolve the problem, then load and reactivate the corrected exit.

The extensions (plug-in routines or PINs) analyze and, in some cases, determine the cause of an abend while allowing the nucleus to continue processing. Each PIN service routine handles a predefined condition when encountered, allowing the Adabas nucleus to

- remain active when it otherwise would terminate abnormally; or
- print extended error diagnostics as an aid to error recovery.

While the PIN is executing, most Adabas functionality is available to it as the registers at the time of the abnormal event are available. The PIN determines whether it is safe to allow the nucleus to continue processing and prints appropriate messages to notify the DBA.

Based on its execution, a PIN can either transfer control to the Adabas nucleus so that it can resume normal processing-usually with a response code -- or it can return control to the error handling and message buffering facility, allowing the Adabas nucleus to terminate abnormally.

A PIN can also be used to format an intelligent dump in a number of circumstances to help debug a particular response or abend code.

A special PIN routine user exit can be used to obtain additional information about response codes and abends. The user exit allows you to specify particular response codes or response code/subcode combinations to be monitored. Once you have modified the user exit, you can reload it and make your changes effective without bringing the database down.

5 Adabas Utilities

- Initial Design and Load Operations 78
- Backup / Restore / Recovery Routines 81
- Database Modification Routines 84
- Audit / Control / Tuning/Reporting Procedures 92

Database services such as loading or deleting files are handled by an integrated set of online and batch-mode utilities. Most utilities can be run in parallel with normal database activity to preclude interruption of daily production. See the *Adabas Utilities Manual* documentation for more information.

Adabas utilities address initial design and load operations, backup/restore/recovery routines, database modification routines, and audit/control/tuning procedures.

Note: Read ***Adabas Online System***, elsewhere in this guide, for information about this menu-driven, interactive DBA tool.

Initial Design and Load Operations

- ADACMP: Compress / Decompress
- ADALOD: Loader
- ADAULD: Unload

ADACMP: Compress / Decompress

ADACMP COMPRESS is used to edit and compress data records to be loaded into the database using ADALOD; ADACMP DECOMPRESS is used to decompress individual files for data structure or field definition changes, or for use as input to non-Adabas programs.

COMPRESS

Input

ADACMP input data must be in a sequential data set/file. Indexed sequential and VSAM input cannot be used. The records may be fixed, variable, or of undefined length. The maximum input record length permitted depends on the operating system. The maximum compressed record length is restricted by the Data Storage block size in use and the maximum compressed record length set for the file (read about the MAXRECL parameter of the ADALOD utility). The input records can be in either blocked or unblocked format.

It is possible to omit the input data set if the parameter NUMREC=0 is supplied.

The logical structure and characteristics of the data for input to COMPRESS are described with field definitions statements (FNDEF to define fields or groups of fields; SUBFN and SUPFN to define sub- or superfields, respectively; COLDE, HYPDE, PHONDE, SUBDE, and SUPDE to define various types of descriptors). Field definitions are used to create the Adabas field definition table (FDT).

By default, input data records are processed in the order of the field definition statements. The FORMAT parameter allows you to change the order of field processing or skip fields.

To support universal encoding (UES), parameters allow you to specify the data architecture and user encodings of the input and the desired file and user encodings to use during compression.

Processing

ADACMP COMPRESS edits and compresses the data records.

Editing includes checking each field defined with a packed (P) or unpacked (U) format to ensure that the field value is numeric and in the correct format. Any record that contains invalid data is written to the ADACMP error data set and is not written to the compressed data set. Adabas user exit 6 can be used to specify additional editing to be performed during ADACMP COMPRESS processing. Read the *Adabas User, Hyperdescriptor, Collation Descriptor, and SMF Exits Manual* documentation for information about user exits.

Compression includes removing trailing blanks from alphanumeric fields; removing leading zeros from numeric fields; removing trailing zeros in floating-point format fields; and packing numeric unpacked fields. Fields with the fixed (FI) option are not compressed, and empty fields located at the end of the record are neither stored nor compressed. Null value fields are processed differently depending on options being used. SQL null value processing is supported.

If universal encoding support (UES) parameters have been specified, compression includes converting the input to the specified encoding for compressed files.

Output

Processed data records are written out together with the file definition information to a sequential data set with the variable blocked record format. This data set, or several such data sets from multiple ADACMP executions, can be used as input to the ADALOD utility. The data set can be used as input to ADALOD even if it contains no records, meaning that no records were provided on the input data set or all records were rejected during editing.

The ADACMP processing report indicates the approximate amount of space required in Data Storage for the compressed records by device type (specified with the DEVICE parameter) and for Data Storage padding factors between 5 and 30 percent. The compression rate is computed based on the real amount of data used as input to the compression routine.

DECOMPRESS

ADACMP DECOMPRESS accepts as input data records from existing Adabas files, either directly without separate file unloading, or already unloaded with the ADAULD utility. If a file is directly decompressed, it is unloaded without FDT information as part of the decompression process, which can save time when decompressing larger files.

Direct decompression of multiclient files can be limited to records for a specific user only when a valid owner ID (ETID parameter) is specified.

The FORMAT parameter may be used to decompress the record to a format other than that specified by the FDT. This is particularly useful when the FDT of an existing file is to be changed.

If universal encoding support (UES) is used, the encoding characteristics for the decompressed file are passed in the header of the compressed sequential input. Parameters allow you to overwrite these encoding characteristics.

Processed data records are written to a sequential data set with the variable blocked record format. Rejected data records are written to the error data set.

ADALOD: Loader

The ADALOD LOAD function loads a file into the database. Compressed records produced by the ADACMP or ADAULD utility may be used as input. A parameter specifies whether the file index is loaded in compressed or uncompressed form.

ADALOD loads each compressed record into Data Storage, builds the address converter for the file, and enters the field definitions for the file into the field definition table (FDT). ADALOD also extracts the values for all descriptors in the file together with the ISNs of all records in which the value is present, to an intermediate data set. This data set is then sorted into value/ISN sequence and entered into the Associator inverted lists.

The ADALOD UPDATE function is used to add or delete a large number of records to/from an Adabas file. The UPDATE function requires considerably less processing time than the repetitive execution of the Adabas adddelete record commands. Records to be added may be the compressed records produced by the ADACMP or ADAULD utility. The ISNs of records to be deleted can be provided either in an input data set or by using control statements.

Records may be added and other records deleted during a single execution of ADALOD.

ADAULD: Unload

The ADAULD utility unloads an Adabas file from the database or from a save tape.

Adabas files are unloaded from a database to

- permit the data to be processed by a non-Adabas program. In this case, the file must also be decompressed after unloading using the DECOMPRESS function of the ADACMP utility.
- create one or more test files, all of which contain the same data. This procedure requires that a file be unloaded, and then reloaded as a test file with a different file number.
- change the field definition table (FDT). This requires that the file be unloaded, decompressed, compressed using the modified field definitions, and reloaded. If the ADADBS utility is used to add field definitions to a file, the file does not need to be unloaded first.

The sequence in which the records are unloaded may be

physical	the order in which they are physically positioned within Data Storage.
logical	a sequence controlled by the values of a user-specified descriptor.
ISN	ascending ISN sequence.

The unloaded record output is in compressed format. The output records have the same format as the records produced by the ADACMP utility.

Adabas files may be unloaded from a qualified database or file save tape to

- include a file from a save tape in one or another test environment.
- move a file from a save tape with one block size to a database with another.

Backup / Restore / Recovery Routines

- ADAPLP: Protection Log / Work Print
- ADARAI: Recovery Aid
- ADARES: Restart
- ADASAV: Save / Restore Database or Files
- ADASEL: Select Protection Data

ADAPLP: Protection Log / Work Print

The ADAPLP utility prints data protection records contained on the Adabas Work data set or the Adabas data protection log.

Note: The ADAPLP utility is primarily a debugging tool for Software GmbH developers. Consequently, detailed information about its output is not documented.

You can specify whether to print:

ALL	all protection records-the default
ASSO	just Associator protection records
DATA	just Data Storage protection records
C1	records resulting from Adabas C1 commands
C5	records resulting from Adabas C5 commands
EEKZ	records written at completion of a nucleus buffer flush
ET	records resulting from Adabas ET commands
REPR	Work data set records used by autorestart to repair the index
SAVO	records resulting from online SAVE database/file operations
VEKZ	records written at completion of update commands

The number of protection records printed can be reduced even more by specifying a file, ISN, or RABN.

ADARAI: Recovery Aid

The Adabas Recovery Aid utility ADARAI can be used to automate and optimize database recovery. For more information, read *Adabas Restart and Recovery* in *Adabas Operations*.

ADARAI supports all Adabas-compatible tape management systems.

The ADARAI utility prepares the recovery log file (RLOG), which records the information about data sets, utility parameters, and protection logs needed to build the recovery job control statements. ADARAI lists the information contained in the RLOG, creates the job control statements to recover the database, and disables ADARAI logging.

Information is stored on the RLOG by generations. A generation includes all activity between consecutive ADASAV SAVE/RESTORE (database) or RESTORE GCB operations. The first generation includes the first ADASAV SAVE/RESTORE (database) or RESTORE GCB operation and extends to (but excludes) the second.

Minimally, the RLOG retains the number of generations specified by the MINGENS parameter during the ADARAI PREPARE step. However, a maximum of 32 generations will be stored on the RLOG if there is enough space available.

Systems using the Recovery Aid feature require a recovery log (RLOG) data set DDRLOGR1, which must first be formatted with the ADAFRM utility and then defined using the ADARAI utility.

ADARES: Restart

The ADARES utility performs functions related to database recovery:

- BACKOUT removes all the updates applied between two checkpoints. The checkpoints used are normally the result of a non-synchronized checkpoint command (C1) but may also be synchronized checkpoints. The complete database may be included in the back-out process, or backout may be limited to selected files.
- CLCOPY copies a command log data set from disk to a sequential data set. This function is necessary only if dual or multiple command logging is in effect for an Adabas session.
- COPY copies a sequential Adabas protection log data set. This function should be executed if the Adabas session in which the sequential protection log data set was created was terminated abnormally.
- MERGE CLOG manually merges command log data sets resulting from individual nucleus CLCOPY executions into a single command log for a cluster of nuclei.
- PLCOPY copies a protection log data set from disk to a sequential data set. This function is necessary only if dual or multiple protection logging is in effect for an Adabas session.

- REGENERATE reapplies all the updates performed between two user-specified checkpoints. The checkpoints specified may be the result of a non-synchronized checkpoint command (C1) but may also be synchronized checkpoints. The REGENERATE function may process all files or be limited to one or more files. It is most often used after the database (or one or more files) has been restored to a previous status with the RESTORE or RESTONL function of the ADASAV utility.
- REPAIR repairs one or more blocks in Data Storage that, for any reason, have become unusable. The most recent save tape of the database and any protection log tapes created thereafter are used as input to this function.

To minimize the time required to recover after a system failure, the BACKOUT, BACKOUT DPLOG or MPLOG, and REGENERATE functions of ADARES can be executed in multiple threads that simulate the original update environment with multiple commands active at one time.

ADASAV: Save / Restore Database or Files

The ADASAV utility saves and restores the contents of the database, or one or more files, to or from a sequential data set. ADASAV should be run as often as required for the number and size of the files contained in the database, and the amount and type of updating. For large databases, ADASAV functions may be run in parallel for the various disk packs on which the database is contained.

Special ADASAV functions are available for use with the Adabas Delta Save Facility. For more information, read the *Adabas Delta Save Facility Facility* documentation.

RESTONL functions restore from one or more SAVE data sets created while the Adabas nucleus was *active* (that is, online); RESTORE functions restore from one or more SAVE data sets created while the Adabas nucleus was *inactive* (that is, offline).

RESTONL and RESTORE have the subfunctions GCB, FILES, and FMOVE:

- Without a subfunction, RESTONL and RESTORE restore entire databases.
- With the GCB subfunction, they restore the general control blocks, Associator RABNs 3-30 of the database, and specified files.
- With the FILES subfunction, they restore one or more files into an existing database to their original RABNs.
- With the FMOVE subfunction, they restore one or more files into an existing database to any free space, allowing changes to extent sizes.

If changes occurred during an online SAVE, the RESTONL function is followed automatically by the RESTPLOG function. RESTPLOG applies the updates that occurred during, and therefore were not included in, the online SAVE.

RESTPLOG is also executed following a RESTONL or RESTONL FILES function that ended before the protection log (PLOG) updates were completely restored. RESTPLOG applies the database updates not applied by the unsuccessful RESTONL function.

The SAVE function to save a database or one or more files may be executed while the Adabas nucleus is active (online) or inactive (offline). If the Recovery Aid option is active, a SAVE database operation begins a new RLOG generation.

ADASEL: Select Protection Data

The ADASEL utility selects information in the Adabas sequential (SIBA), dual, or multiple (PLOG) protection log. ADASEL decompresses the information and writes it to a print data set (DDDRUCK) or to a user-specified output data set.

The protection log contains information on all updates applied to the database during a given Adabas session. Information selected by ADASEL can be used for auditing or as input to a Natural or non-Adabas program.

You can select before-images, after-images, or both for new, updated, and deleted records. You can also select data written to the protection log by an Adabas C5 command.

Database Modification Routines

- ADACDC: Changed-Data Capture
- ADACNV: Database Conversion
- ADADBS: Database Services
- ADADEF: Define a Database
- ADAFRM: Format Data Sets
- ADAINV: Invert
- ADAORD: Reorder
- ADAZAP: Modify Physical Database Blocks

ADACDC: Changed-Data Capture

ADACDC is an interval-driven, asynchronous mass update feature to generate a sequential file containing all database modifications. This feature is important for open systems and data warehousing solutions.

ADACDC then processes the raw data in the sequential file to isolate the latest status of the data. The ADACDC utility

- takes as input one or more sequential protection logs; and
- produces as output a *delta* of all changes made to the database over the period covered by the input protection logs.

A *delta of changes* means that the last change to each ISN in a file that was altered during this period appears on the primary output file.

This output may be used on a regular basis as input for data warehousing population procedures so that what is applied to the data warehouse database is the delta of changes to a database rather than a copy of the entire database. This affords more frequent and less time consuming updates to the data warehouse, ensuring greater accuracy of the information stored there.

ADACNV: Database Conversion

The utility ADACNV *must* be used to perform all necessary conversions of both operating system-dependent and -independent database system structures when moving in either direction between Adabas versions.

The ADACNV utility converts (CONVERT) an Adabas database from lower versions to higher versions; it also reverts (REVERT) an Adabas database from a higher version to a lower one. Some restrictions may apply these functions.

To ensure database integrity, ADACNV writes changed blocks first to intermediate storage; that is, to the sequential data set DDFILEA. After all changed blocks have been written out to DDFILEA, a point of no return is reached and the changed blocks are written to the database. If ADACNV terminates abnormally after the point of no return, the RESTART parameter can be used to begin the ADACNV run by reading the contents of DDFILEA and writing them out to the database.

The TEST parameter is provided to check the feasibility of a conversion or reversion without writing any changes to the database.

ADADBS: Database Services

All ADADBS functions can also be performed using Adabas Online System (AOS). When the Adabas Recovery Aid is active, using AOS is preferable for file change operations because it writes checkpoints that are necessary for recovery operation.

ADADBS offers a variety of functions, any number of which may be performed during a single execution of the utility.

Database Functions

The ADD function adds a new data set to the Associator or Data Storage to a maximum of 99 data sets for each. However, your actual real maximum will be less because the maximum derived from the block size of the first Associator data set (DDASSOR1).

The DECREASE function reduces the size of the last data set currently being used for Associator or Data Storage. The space to be released must be available in the free space table (FST).

The DECREASE function does *not* deallocate any of the specified physical extent space. To deallocate space, you must decrease the database with the DECREASE function; save it with ADASAV SAVE; reformat the data sets with ADAFRM; and restore the database with ADASAV.

The INCREASE function increases the size of the last data set currently being used for the Associator or Data Storage. This function may be executed any number of times for the Associator. When the maximum (99) number of Data Storage Space Tables (DSSTs) has been reached, all Data Storage extents must be combined into a single extent with either the REORASSO or REORDB function of the ADAORD utility.

The RENAME function changes the name assigned to a file or database. If a file is not specified or is specified with file number zero, the database is renamed.

The TRANSACTIONS function suspends and resumes update transaction processing; that is, it creates a quiesced state for the database that could be a recoverable starting point.

File Functions

The ALLOCATE/DEALLOCATE functions are used to allocate/deallocate, respectively, a logical extent (an address converter, Data Storage, normal or upper index) of a specific size. Only one extent may be allocated or deallocated per ADADBS execution.

The CHANGE function changes the standard length of an Adabas field but does not modify records in Data Storage. The user is, therefore, responsible for preventing references to the field that would cause invalid results because of an inconsistency between the new standard length as defined to Adabas and the actual number of bytes contained in the record.

The DELETE function deletes an Adabas file from the database. The file may not be coupled. If an Adabas expanded file is specified, the complete expanded file (the anchor and all component files) is deleted. The deletion process deallocates all logical extents assigned to the file, releasing space that may be used for a new file or for a new extent of an existing file.

The DSREUSE function determines, for a specified file, whether Data Storage blocks that become free as a result of record deletion are reused. Block reuse is originally determined when a file is loaded into the database with the ADALOD FILE function, or when the system file is defined with the ADADEF DEFINE function. In both cases, block reuse defaults to YES.

To support universal encoding (UES), the ENCODEF function can be used to define encodings for fields in a file that is already loaded:

- an EBCDIC file encoding for alphanumeric fields; or
- a user encoding for the wide-character fields. The file encoding of wide-character fields cannot be changed using this function.

The ISNREUSE function determines, for a specified file, whether Adabas reuses the ISN of a deleted record for a new record. If not, each new record is assigned the next higher unused ISN.

For a specified Adabas file that is not a system file, the MODFCB function modifies parameters such as file padding factors for the Associator or Data Storage; maximum size of secondary logical extent allocations for Data Storage, normal index, and upper index; maximum compressed record length permitted; and whether a user program is allowed to perform a file refresh operation by issuing a special E1 command.

The NEWFIELD function adds one or more fields to a specified Adabas file that is not a system file. The new field definition is added to the end of the field definition table (FDT). NEWFIELD cannot be used to specify actual Data Storage data for the new field; the data can be specified later using Adabas add or update commands, or Natural commands.

The ONLINVERT function allows you to invert files when online applications are active, ensuring continuous access to the files. You can add one descriptor per file per run.

The ONLREORFASSO (reorder Associator), ONLREORFDATA (reorder Data Storage), and ONLREORFILE (reorder both Associator and Data Storage) functions allow you to reorder a list of files when online applications are active, ensuring continuous access to the files. Files are reordered within their existing extents, thus increasing I/O performance as free space is recovered and the sort sequence of data records is changed according to processing needs.

The REFRESH function sets the file to "0" records loaded; sets the first extent for the address converter, Data Storage, normal index, and upper index to *empty* status; and deallocates other extents.

The RELEASE function releases a descriptor from descriptor status. All space currently occupied in the Associator inverted list for this descriptor is released. The space can then be reused for this file by reordering or by ADALOD UPDATE. No changes are made to Data Storage.

The RENAME function changes the name assigned to a file or database. If a file is not specified or is specified with file number zero, the database is renamed.

The RENUMBER function changes the number of an Adabas file that is not a system file. If the new number specified is already assigned to another file, the RENUMBER function will not execute.

The UNCOUPLE function eliminates the coupling relationship between two files.

Other Functions

The CVOLSER function prints the Adabas file extents that are contained on a disk volume specified by its volume serial number.

The DELCP function deletes checkpoint information recorded up to and including a specified date; checkpoint information recorded after the date specified is not deleted. After running ADADBS DELCP, the remaining records are reassigned ISNs to include those ISNs made available when the checkpoint records were deleted. The lower ISNs are assigned but the chronological order of checkpoints is maintained.

The OPERCOM function issues operator commands to the Adabas nucleus. Adabas issues a message to the operator, confirming command execution. In cluster environments, OPERCOM commands can often be directed to another nucleus in the cluster or to all nuclei in the cluster for execution.

The PRIORITY function sets or changes the Adabas priority of a user. A user's priority can range from 0 (the lowest) to 255 (the highest, and the default). The priority value is added to the operating system priority by the interregion communications mechanism. The user for which a priority is to be set or changed is identified by the same user ID provided in the Adabas control block (OP command, Additions 1 field).

The RECOVER function recovers space allocated by rebuilding the free space table (FST). RECOVER subtracts file and DSST extents from the total available space.

The REFRESHSTATS function resets statistical values maintained by the Adabas nucleus for its current session. Parameters may be used to restrict the function to particular groups of statistical values:

- ALL (the default) resets values for the combination of CMDUSAGE, COUNTERS, FILEUSAGE, POOLUSAGE, and THREADUSAGE.
- CMDUSAGE resets the counters for Adabas direct call commands such as L*x*, S*x*, or A1.
- COUNTERS resets the counter fields for local or remote, physical or logical calls, format translations, format overwrites, autorestarts, protection log switches, buffer flushes, and command throw-backs.
- FILEUSAGE resets the count of commands for each file.
- POOLUSAGE resets the high-water marks for the nucleus pools such as the Work pool, the command queue, or the user queue.
- THREADUSAGE resets the count of commands for each Adabas thread.

Adabas maintains a list of the files used by each Adabas utility in the data integrity block (DIB). The DDIB operator command (or Adabas Online System) displays this block to determine which jobs are using which files. A utility removes its entry from the DIB when it terminates normally. If a utility terminates abnormally (for example, the job is canceled by the operator), the files used by that utility remain in use. The RESETDIB function releases any such files and resets the DIB entries for a specified job and/or a particular utility execution.

ADADEF: Define a Database

The ADADEF utility is used to

- define a new database (DEFINE functions), including the checkpoint file,
- set database encoding defaults for a new database or modify them (MODIFY function) for an existing database
- define a new Work file (NEWWORK function) for an existing database.

Databases are defined with name, ID, components (Associator, Data Storage, and Work) with device type and size, and default encodings.

Adabas uses certain files to store system information. The checkpoint file is used to store checkpoint data as well as user data provided with the Adabas CL and ET commands. It is required and must be specified in the ADADEF DEFINE (database) function.

Before database components (Associator, Data Storage, and Work) can be defined with ADADEF, each must be formatted by the ADAFRM utility.

ADAFRM: Format Data Sets

The ADAFRM utility formats the Adabas direct access (DASD) data sets; that is, the Associator, Data Storage, and Work data sets as well as the intermediate storage (temp, sort, recovery log, and dual or multiple command/protection log) data sets.

Formatting with ADAFRM comprises two basic operations: creating blocks (that is, RABNs) on the specified tracks/cylinders; and filling the created blocks with binary zeros (nulls).

Any new data set must be formatted before it can be used by the Adabas nucleus or an Adabas utility. After increasing a data set with the ADADBS INCREASE or ADD function, new RABNs must also be formatted.

ADAFRM also provides functions to reset existing Associator, Data Storage, or Work blocks to binary zeros (nulls).

More than one ADAFRM function (ASSOFRM, DATAFRM, RLOGFRM, and so on) can be performed in the same job. However, each function must be specified on separate statements.

ADAINV: Invert

The ADAINV utility is used to

- create a descriptor (INVERT function); or
- couple two files (COUPLE function).

The INVERT function

- modifies the field definition table (FDT) to indicate that the specified field is a descriptor; and
- adds all values and corresponding ISN lists for the field to the inverted list.

The newly defined descriptor may then be used in the same manner as any other descriptor. This function may also be used to create a subdescriptor, superdescriptor, phonetic descriptor, hyper-descriptor, or collation descriptor.

The COUPLE function adds a common descriptor to two files (updates their inverted lists). Any two files may be coupled provided that a common descriptor with identical format and length definitions is present in both files. A single file may be coupled with up to 18 other files, but only one coupling relationship may exist between any two files at any one time. A file may not be coupled to itself.

Note: Only files with numbers 255 or lower can be coupled.

Changes affecting a coupled file's inverted lists are automatically made to the other file. The DBA should consider the additional overhead required to update the coupling lists when the descriptor used as the basis for coupling is updated, or when records are added to or deleted from either file. For example, if a field used as the basis for coupling contains a large number of null values and is not defined with the NU (null suppression) option, the result may be a significant increase in execution time and required disk space to store the coupling lists.

An interrupted ADAINV operation can be restarted without first having to restore the file.

ADAORD: Reorder

Three types of functions are available within the ADAORD utility; only one function may be executed during a given execution of ADAORD.

Reorder Functions

The REORASSO function physically reorders all Associator blocks for all files; REORFASSO reorders the Associator for a single file. This eliminates Associator space fragmentation, and combines multiple address converter, normal and upper index, and Data Storage Space Table (DSST) component extents into a single logical extent for each component.

The REORDATA function reorders Data Storage for all files in the database; REORFDATA reorders Data Storage for a single file. This condenses extents containing only empty blocks, and also eliminates any Data Storage fragmentation caused by file deletion.

The REORDB function performs both the REORASSO and REORDATA functions in a single ADAORD execution; the REORFILE function performs both the REORFASSO and REORFDATA functions in a single ADAORD execution. The records may be reordered in the logical sequence by a descriptor, by ISN, or in the current sequence.

Restructure Functions

The RESTRUCTURE functions are used to relocate a database or specified files to a different physical device.

The RESTRUCTUREDB function unloads an entire database to a sequential data set; RESTRUCTUREF unloads one or more files to a sequential data set. This data set can be used as input to the STORE function.

Store Function

The STORE function loads one or more files into an existing database using the output produced by the RESTRUCTURE functions or the REORDB function.

ADAZAP: Modify Physical Database Blocks

The ADAZAP utility is used to modify physical database blocks. It can be used to

- write a checkpoint for each VER and REP it processes providing an audit trail of database modifications. SYNP 3F checkpoints are printed by both Adabas Online System and ADAREP and are ignored by ADARES.
- handle errors according to standard Adabas utility conventions.

Because caution is necessary when running ADAZAP:

- we recommend that you have a current save tape available before running ADAZAP. If an error is encountered while running ADAZAP, it may be necessary to restore the affected file or database.
- a mastercode available only to authorized personnel controls its use. The mastercode is distributed by Software GmbH on written request.

Audit / Control / Tuning/Reporting Procedures

- ADACHK: Adabas Common Check Utility
- ADADRU: Database Resource Usage Utility
- ADAMER: ADAM Estimation
- ADAREP: Report
- ADAWRK Utility: Work Area Recovery Reports
- ADAZIN: Print Database Information

ADACHK: Adabas Common Check Utility

The ADACHK utility performs a variety of functions and replaces the now obsolete utilities ADAACK, ADADCK, ADAICK, ADAVAL, and ADAPRI. The ADACHK utility also brings additional features not previously provided by those other utilities. The ADACHK utility also includes the integrity checks that ADAREP currently performs and checks all persistent data structures on disk with minimal impact to the production environment.

You should only use ADACHK for diagnostic purposes. It:

- checks the address converter for a specified file(s) and ISN range;
- checks each address converter element to determine whether the Data Storage RABN is within the used portion of the Data Storage extents specified in the file control block;
- checks the ISN for each record in each Data Storage block within the specified ISN range to ensure that the address converter element for that ISN contains the correct Data Storage RABN;
- checks whether the block length is within the permitted range (4 block length physical block size);
- checks that the sum of the lengths of all records in the Data Storage block plus 4 equals the block length;
- checks that any record exists with a record length greater than the maximum compressed record length for the file or with a length 0;
- checks that any duplicate ISNs exist within one block;
- checks that the associated DSST element contains the correct value (if not the DSST must be repaired using the ADACHK DSCHECK REPAIR parameter);
- checks the physical structure of the Associator;
- validates the index based upon the descriptor value structures and the Associator extents defined by the general control blocks (GCBs) and file control blocks (FCBs);
- compares the actual descriptor values contained in the records in Data Storage with the corresponding values stored in the Associator to ensure that the Associator and Data Storage are synchronized, and that there are no values missing from the Associator; and

- checks the entire database for overlaps or gaps and the integrity of any expanded file chains.

ADADRU: Database Resource Usage Utility

ADADRU should only be used for reporting purposes. It may be used to:

- produce detailed reports about the space usage of files;
- produce reports to show the RABN allocation, the free space inside RABNs for ASSO (Normal Index) and DATA areas with indications of available padding space inside this free space, the presence of free blocks, and the average characteristics of objects (record lengths and number of records in the block); and
- estimate the efficiency of database space usage (for one or more specific files or for the database as a whole) and can be a helpful as source of information when deciding whether database reorganization is required to optimize block sizes, decrease fragmentation and space occupied, and increase overall performance etc

ADAMER: ADAM Estimation

The ADAMER utility produces statistics that indicate the number of Data Storage accesses required to find and read a record when using an ADAM descriptor. This information is used to determine

- whether the number of accesses required to retrieve a record using an ADAM descriptor would be less than the standard Adabas accessing method;
- the amount of Data Storage space required to produce an optimum distribution of records based on the randomization of the ADAM descriptor.

The input data for ADAMER is a data set containing the compressed records of a file produced by the ADACMP or ADAULD utility.

The field to be used as the ADAM descriptor is specified with the ADAMDE parameter. A multiple value field or a field contained within a periodic group may not be used. The ISN assigned to the record may be used instead of a descriptor as the basis for randomization (ADAMDE=ISN).

The ADAM descriptor must contain a different value in each record, since the file cannot be successfully loaded with the ADAM option of the ADALOD utility if duplicate values are present for the ADAM descriptor. The ADAMER utility requires a descriptor field defined as unique (UQ), but does not check for unique values; checking for unique descriptor values is done by the ADALOD utility when loading the file as an ADAM file.

The BITRANGE parameter may be used to specify that a given number of bits are to be truncated from each ADAM descriptor value before the value is used as input to the randomization algorithm. This permits records containing ADAM descriptor values beginning with the same value (for example, 40643210, 40643220, 40643344) to be loaded into the same physical block in Data Storage. This technique can be used to optimize sequential reading of the file when using the ADAM

descriptor to control the read sequence, or to remove insignificant information such as a check digit.

ADAREP: Report

The ADAREP utility produces a status report that provides information concerning the current physical layout and logical contents of the database or a qualified save tape.

The information provided in this report includes:

- a database overview: the database name, number, creation date/time, file status, and current log number;
- current space resources for Associator, Data Storage, and Work: amount and locations of currently used space, and allocated but unused space;
- summary and detailed file information: summary by file of ISN, extent, padding factor, used/unused Associator and Data Storage space, and file options; and detailed, optionally by file, that includes all summary information plus MINISN/MAXISN settings, detailed space information, creation and last use date/time, field definition table (FDT) contents, and general or extended checkpoint file information;
- checkpoint information: general and extended checkpoint file information; and
- physical structure: Associator/Data Storage RABN information including device type, VOLSER number, file number (if appropriate), and usage (AC, NI/UI, Data Storage, DSST, or unused).

The purpose of the save tape report is to determine what the save tape contains.

ADAWRK Utility: Work Area Recovery Reports

The ADAWRK utility can be used to produce reports from records in the autorestart area of Work part 1. This information can be used when the database autostart fails and the database will not come up. In such a situation, you need to know what can be done to get the database back up and running quickly, with a minimum amount of lost data and with enough information to retrieve any lost updates.

The data on the ADAWRK reports can help you determine whether:

- you should run a restore/regenerate (ADASAV RESTORE utility function followed by the ADARES REGENERATE utility function) of the database, which can be time-consuming;
- excluding specific files from the autorestart using AREXCLUDE and then restoring/regenerating only these single files would be beneficial; or
- the database can be quickly repaired so it can be started and functional more quickly.

ADAWRK can produce the following reports:

- The Environment report is always produced, regardless of the ADAWRK parameters specified. It identifies the ADAWRK parameters used to produce the report as well as the Work data sets used for the report.
- The Summary report is produced by default and provides an overview of the data in the autorestart area of Work part 1.
- The File report is an optional report that provides a breakdown of the data in the autorestart area of Work part 1 by file.
- The Transaction report is an optional report that provides a breakdown of the data in the autorestart area of Work part 1 by transaction.
- The Checkpoint record report is an optional report that lists the checkpoint records and associated data within the autostart area of Work part 1.
- The Replication-related reports are optional reports that report on data that may need to be replicated.

The ADAWRK utility will only report on transactions that may need to be corrected as part of the autorestart processing logic. You can filter all of the Work part 1 autorestart area records processed and reports produced in an ADAWRK run by communication ID, ETID, user ID, and file number.

ADAZIN: Print Database Information

The ADAZIN utility can be used to print maintenance information about Adabas load modules and status information about the Adabas SVC on the system in which ADAZIN is run. Names of target load modules and SVC numbers can be specified to the utility to limit the range of the printed report. Module information in the ADAZIN report includes:

- The load module name;
- CSECT names (if appropriate);
- The date the module was last compiled;
- The Adabas version and release of the module;
- The number of the library from which the module was loaded within the concatenation list;
- A list of zap numbers applied to the module for the zap base level.
- The load module name CSECT names (if appropriate), the date the module was last compiled, the Adabas version and release of the module, the number of the library from which the module was loaded within the concatenation list, a list of zap numbers applied to the module for the zap base level.

The Adabas modules that ADAZIN reviews for the report reside in a load library (or concatenation of load libraries) defined through either the DDZIN or STEPLIB job control statements in the ADAZIN batch job.

6 Licensing Adabas

You must install a valid license file on all mainframe platforms in which your Software GmbH mainframe product is installed.

The license file is provided as an XML document (encoded in US-ASCII). This document can be viewed using a browsing tool or text editor on a PC. It can also be viewed on the mainframe using the DISPLAY function of the license utility, LICUTIL, described in *Mainframe Product Licensing*. The license file contains text that represents the licensing information for your product and an associated digital signature, the license key. Among other things, it also displays Software GmbH legal notices and environmental information.

Your license file is obtained from your Software GmbH sales representative. If you should have problems with your license file, please contact your sales representative. Please do not edit the file yourself, as you may invalidate it during your attempt. If the product license is incorrect, insufficient, or not installed, Adabas terminates. Contact your Software GmbH sales representative for assistance.

Adabas requires MLC Version 1.3.8 or higher. For complete information about the licensing process for Software GmbH mainframe products, read *Mainframe Product Licensing*.

7 Adabas Security

- Security System User IDs 100
- Data Encryption 103
- Multiclient Files 103
- Adabas Security and ADASCR 104
- Adabas SAF Security 105
- Related Security Options 109

Adabas provides the following facilities to prevent unauthorized access and updates to Adabas database files:

Security System User IDs

Security system user IDs are the sign-on user IDs provided by security software such as RACF, ACF2, or Top Secret. Adabas support for security system user IDs allows you to track user activity in Adabas databases.

Security system user IDs are supported in Adabas system fields. command logs, some operator commands, operator command output displays, PRILOG displays, Adabas utilities, and Adabas session statistics. A SECUID ADARUN parameter is provided to allow you to specify the requirement level of security system user IDs for a database. This section describes security system user ID support in Adabas in more detail.

- CICS System Requirements
- SECUID ADARUN Parameter
- SECUID Operator Command
- STOPSU and STOPSUR Operator Commands
- System Field Support for Security System User IDs
- SECUID Field Included in Command and Protection Logs
- Operator Command Output Displays
- PRILOG Displays
- Adabas Utility Support
- Adabas Session Statistics

CICS System Requirements

In CICS environments only, if you want your security system user IDs to be stored in Adabas user queue elements (making them available for display and review as well as preventing response code 200, ADARSP200, subcode 21 when ADARUN SECUID=REQUIRE is in effect for Adabas), you must code the SAF parameter as YES. This is only required in CICS environments; in other environments, the security system user IDs are automatically stored.

For more information, read *Step 6. Prepare the CICS Link Globals Table -- CICSGBL* and *SAF: Adabas Security Interface Flag*, in the *Adabas z/OS Installation Guide*.

SECUID ADARUN Parameter

Use the SECUID ADARUN parameter to specify the requirement level of security system user IDs for a database. You can indicate how Adabas handles calls from users without a security system user ID or with a security system user ID that changed during the Adabas session. For complete information about the SECUID ADARUN parameter, read *SECUID Parameter: Security System User ID Requirement Level*, in the *Adabas Operations Manual*.

SECUID Operator Command

The SECUID operator command can be used to alter the setting of the ADARUN SECUID parameter while the nucleus is active. This operator command is valid for use both on the console and in the ADADBS OPERCOM utility function. For more information about the SECUID operator command, read *SECUID Command*, in the *Adabas Operations Manual*; for more information about use of the SECUID operator command in the ADADBS OPERCOM utility function, read about the SECUID operator command, in the *Adabas Utilities Manual*.

STOPSU and STOPSUR Operator Commands

You can use the STOPSU operator command to stop and delete a user with a specified security user ID. Any open transactions of the stopped user are backed out. No response code is issued; the next time the stopped user issues a command, a new user queue element (UQE) is created. For more information about the STOPSU operator command, read *STOPSU Command*, in the *Adabas Operations Manual*; for more information about use of the STOPSU operator command in the ADADBS OPERCOM utility function, read about the STOPSU operator command, in the *Adabas Utilities Manual*.

You can use the STOPSUR operator command to stop a user with a specified security user ID, but the stopped user is not immediately deleted. Any open transactions of the stopped user are backed out. The stopped user is only deleted after he or she has issued a subsequent command and response code 22 (ADARSP22), subcode 54 has been issued in response to that command. This response code-subcode combination is used to notify users that their Adabas activity has been halted and their user session resources have been freed. Only after the response code-subcode combination has been issued is the user queue element (UQE) of the stopped user deleted. For more information about the STOPSUR operator command, read *STOPSUR Command*, in the *Adabas Operations Manual*; for more information about use of the STOPSUR operator command in the ADADBS OPERCOM utility function, read about the STOPSUR operator command, in the *Adabas Utilities Manual*.

System Field Support for Security System User IDs

Security system user IDs (SSIDs) can be stored in system fields in an Adabas file. The value "SECUID" can be used in field definitions that include the SY field option. This SY=SECUID value indicates that the system field contains an SSID. For complete information read ***System Fields***elsewhere in this guide and *SY: System Field* in the *Adabas Utilities Manual*.

SECUID Field Included in Command and Protection Logs

The SECUID field is included in Adabas command logs (CLOGs) and protection logs (PLOGs).

Operator Command Output Displays

Output reports from various operator commands (such as the DPARM, DCQ, DFILES, DUQ, or DHQA operator commands) include the current SECUID setting of the user's actual security system user ID.

PRILOG Displays

The PRILOG print program displays the SECUID when CLOGLAYOUT=8 is used. All of the reports produced by the PRILOG print program include this field.

Adabas Utility Support

The following Adabas utilities support security system user IDs:

- The security system user ID will appear in the extract file and the primary output file produced by the ADACDC utility. If the security system user ID is not known, blanks are stored in the extract file. For more information, read *ADACDC Utility: Changed-Data Capture*, in the *Adabas Utilities Manual*.
- The ADASEL utility supports security system user IDs in three places in the SELECT parameter of the ADASEL utility:
  1. The SECUID parameter can be used to specify a security system user ID or range of security system user IDs in the value-criterion of the WITH clause or the IF statement.
  2. The SECUID [HEX] parameter can be used in the DISPLAY instruction to indicate that the security system user ID of the user who added, updated, or deleted the record should be displayed.
  3. The security system user ID appears in the output log data set if the EXPANDED keyword is specified in the output instruction of the ADASEL utility. If the security system user ID is not known, "UNKNOWN" displays.

  For more information, read *ADASEL Utility: Select Protection Data*, in the *Adabas Utilities Manual*.

- The ADAWRK utility includes a SECUID parameter that you can use to specify up to 24 security system user IDs (SECUIDs) in character format. Each SECUID must be one to eight bytes long. When SECUIDs are specified, only Work part 1 autorestart area records for those SECUIDs are processed by the ADAWRK utility and printed on its reports. For more information, read *ADAWRK Utility: Work Area Recovery Reports*, in the *Adabas Utilities Manual*.

Adabas Session Statistics

The high usage session statistics produced for a nucleus session include the user's security system user ID, if it is known. For more information, read *High Usage Session Statistics*, in the *Adabas DBA Tasks Manual*.

Data Encryption

Data encryption is an integral feature of Adabas and requires no options or extra modules. Data may be enciphered before being placed in the database.

The user must provide the cipher key at the time records are stored. This key is not stored and must be available to request or decipher the data. This minimizes the chances of data being compromised by unauthorized access to the system.

To retain maximum control over cipher codes, an Adabas user exit program can be created to insert the currently valid cipher code into user applications; this removes the need to make the codes known to users, and protects the file from corruption that can occur by adding data that is encrypted with the wrong cipher code.

Multiclient Files

Also available as an integral feature of Adabas that requires no options or special modules is the multiclient file.

A single Adabas physical file defined as multiclient can store records for multiple users or groups of users. The multiclient feature divides the physical file into multiple logical files by attaching an internal owner ID to each record.

The owner ID is assigned to a user ID. A user ID can have only one owner ID, but an owner ID can belong to more than one user. Each user can access only the subset of records that is associated with the user's owner ID.

Note: For any installed external security package such as RACF or CA-Top Secret, a user is still identified by either Natural ETID or LOGON ID.

All database requests to multiclient files are handled by the Adabas nucleus.

Adabas Security and ADASCR

Access/update control is available only with Adabas Security and the related security utility ADASCR that defines and controls Adabas Security functions.

Adabas Security provides two levels of protection: access/update and value.

- Access/Update Level Protection
- Value Level Protection

Note: Adabas Security documentation is available upon request. For complete information about Adabas Security, contact your Software GmbH representative.

Access/Update Level Protection

Access-/update-level protection applies a basic level of security on a file-by-file basis. Access/update protection can be defined for some files and not for others. It restricts use of a file or field within the file to those having an appropriate access/update profile definition and a password specified by the user of the file.

Access/update permission values ranging from 0 to 14 are defined for each user and attached to that user's password, and each protected file (and selected field or fields, if desired) has equivalent access/update threshold protection values of the same range. Only a user whose permission value equals or is greater than the protection level of the specified file (and, when applicable, field) is permitted to perform that operation type (access or update) on the file or field. An access/update permission level of 0 only allows access/update of unprotected files or fields with protection level 0 or no defined protection password.

Value Level Protection

Value-level protection applies restrictions on the type and range of values that can be accessed or updated in specific fields. The restrictions are applied according to user password (files with fields using value-level protection must be password-protected). They can be for specific values or for value ranges, and can be either accept or reject criteria.

Adabas SAF Security

- Adabas Interface to SAF-based Packages
- Adabas SAF Security Overview

Adabas Interface to SAF-based Packages

The System Authorization Facility (SAF) is used by z/OS to provide rigorous control of the resources available to a user or group of users. Compatible security packages such as IBM's RACF and Computer Associates' ACF2 or Top Secret allow the system administrator

- to maintain user identification credentials such as user ID and password; and
- to establish profiles determining the data sets, storage volumes, transactions, and reports available to a user.

Generally, a security package allows the system administrator to authorize a user's access to system resources. The security package then monitors all users and their resource usage to ensure that no unauthorized access or change occurs. Attempts by unauthorized users to use either the system or specific system resources are recorded and reported.

A user profile, which can be for a single user or a group of users, defines which system hardware and software resources a user is allowed to use. A resource profile defines access/update privileges for one or more devices, volumes, and/or programs (resources that must be used together to perform certain functions can be defined together in the same profile).

When a user logs on to the system, the security package uses the user's logon ID to identify that user's profile. Each time the user attempts to perform a task or access information, the security package uses information in its resource profiles to allow or deny access. Using the profile concept, the security package expands the single point of authorization-the logon ID-to provide extensive control over all system resources.

The resulting security repository and the infrastructure to administer it represent a significant investment. At the same time, the volume of critical information held by a business is constantly growing, as is the number of users referencing the data. The challenge of controlling these ever-increasing accesses requires a solution that is flexible, easy to implement and, above all, one that safeguards the company's investment.

Adabas SAF Security Overview

Adabas SAF Security enhances the scope of SAF-based security packages by integrating Adabas resources into the central security repository. Adabas SAF Security enables:

- a single control and audit system for all resources;
- industry-standard protection of Adabas data; and
- maximized return on investment in the security repository.

Adabas SAF Security operation can be tailored on a nucleus-by-nucleus basis, allowing great flexibility in its implementation. It comprises:

- a server operating in each secured Adabas address space;
- an online administration and monitoring system, an application written in Natural and accessed from either the demo or full version of Adabas Online System (AOS); and
- a plug-in routine PINSAF that interfaces with the Adabas error handling facility. It is activated automatically at initialization to aid problem diagnosis.

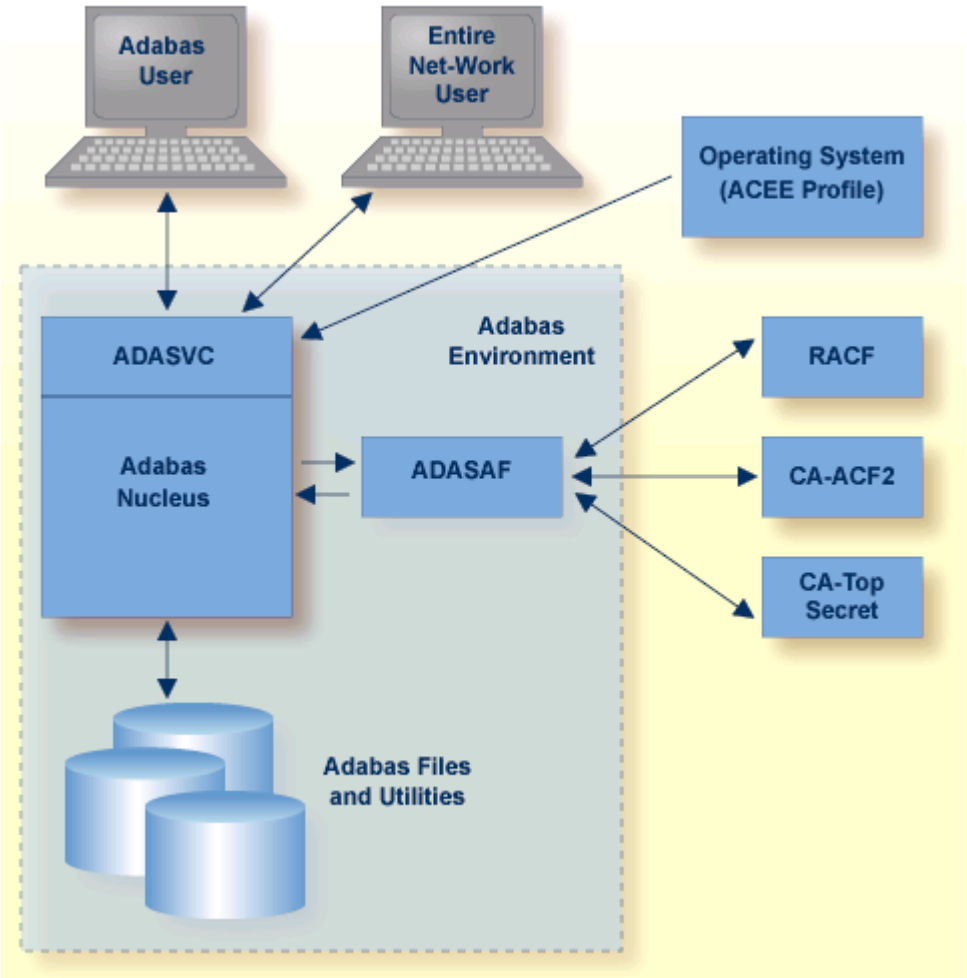
Adabas SAF Security allows you to protect the following Adabas resources:

Resource	Protection
Adabas Nucleus	■ Controls the users allowed to start an Adabas nucleus. ■ Controls the users allowed to perform Adabas nucleus administration functions; for example, a user or group might be allowed to display nucleus session parameters but not modify them. ■ Controls the Adabas nucleus operator commands that can be issued from the system console.
Adabas Utilities	■ Controls the users allowed to execute utilities by utility name or database ID; for example, a user or group might be allowed to run ADAREP but not ADASAV against a particular database. ■ Controls the users allowed to execute utilities by utility function; for example, a user or group might be allowed to run ADASAV SAVE but not ADASAV RESTORE against a particular database. ■ Controls the users allowed to execute utilities by function and file number; for example, a user or group might be allowed to run ADASAV SAVE FILE=10 but not ADASAV SAVE FILE=20 against a particular database.
Adabas Files	■ Controls the users allowed to access database files. ■ Controls the users allowed to use access (READ/FIND) and update (STORE/UPDATE/DELETE) commands. To optimize performance, commands that are not file-specific (such as RC) are disregarded.

Resource	Protection
Production Environment Data	Controls the users allowed to operate in a production or test environment. Such cross-level checking could be used, for example, to prevent damage by an application program inadvertently cataloged against the wrong database ID.
Transaction Data	Controls the users allowed to store or retrieve ET data.
File Passwords and Cipher Codes	Dynamically applies passwords and codes held in the security repository or supplied by a user exit. This eliminates the need for the application to manage security data and removes the requirement to transmit sensitive information from the client to the database.
Administration Services	Controls the users allowed to perform online administration of the Adabas environment. The following administration services can be protected in this way: ■ Adabas Basic Services ■ Adabas System Coordinator Administration Services ■ Adabas Fastpath Administration Services ■ Adabas Vista Administration Services ■ Adabas Transaction ManagerAdministration Services ■ Adabas SAF Security Administration Services

Refer to the *Adabas SAF Security* documentation for more information regarding the protection of the above resources.

In the following figure, all traffic between clients and the Adabas nucleus is controlled by the Adabas router (ADASVC). The Adabas router extracts the user's logon ID from the user's ACEE (a system representation of the user's identification) and makes it available for Adabas SAF Security (ADASAF) to perform security checks using the installed external security package.



The central security logon ID is used to log on to the system. Through the operating system or TP monitor, the installed external security package package performs authentication of the logon ID. For calls from a remote workstation or non-IBM platform, a remote logon procedure is used to give the logon ID to ADASAF.

Full, flexible control is maintained with a *one user : one definition* approach while previous investments in host-based security systems and infrastructures are enhanced, not discarded.

Related Security Options

- Adabas Online System Security
- Natural Security
- Using the SAF Repository to Secure Software GmbH Products
- Entire Net-Work SAF Security (NETSAF)

Adabas Online System Security

The demo version of Adabas Online System (AOS) distributed with Adabas includes a security facility for restricting access to the Adabas online facilities. AOS Security requires Natural Security as a prerequisite. See the *Adabas Security* documentation for more information.

Natural Security

The Natural Security system provides extensive security for Adabas/Natural users. It is required for AOS Security and recommended for other features of Adabas. See the *Natural Security* documentation for more information.

Using the SAF Repository to Secure Software GmbH Products

Adabas SAF Security or ADASAF is one of several Software GmbH security products that enhance the effectiveness of the SAF central security repository:

Product	Protects
Adabas SAF Security	Adabas
Entire Net-Work SAF Security	Entire Net-Work
EntireX Security	EntireX, Entire Broker, Broker Services
Natural SAF Security	Natural

Entire Net-Work SAF Security (NETSAF)

The Entire Net-Work SAF Security (NETSAF) is a separate, optional product for z/OS environments running Entire Net-Work. It allows Entire Net-Work clients to access SAF-secured data sources (targets); for example, Adabas, EntireX Communicator, and Entire System Server.

NETSAF can be activated on a link-by-link basis. If only one node of several communicates externally, security can be activated for that node alone and only for external links.

To secure Entire Net-Work, it is necessary to define resource profiles in the SAF repository. Resource profiles are defined for each host target. Adabas resource profiles can be defined at the file level.

The command type determines the access level required for successful authorization: valid access levels are READ, UPDATE, and CONTROL. CONTROL applies to AOS commands, for example.

Point-of-access verification of incoming requests is made against the SAF-based central security repository: all access from mainframe clients can be verified against the same security profile.

Security checks are based on a trusted user ID, which must exist in the central security repository. In some cases, the user ID is authenticated in the caller's home environment or is fixed by, for example, the Entire Net-Work configuration. A user ID can be lost if calls are routed through an intermediate gateway node.

8 Optional Product Extensions

- Adabas Bridges 112
- Adabas Caching Facility 116
- Adabas Cluster Services 117
- Adabas Delta Save Facility 120
- Adabas Encryption 121
- Adabas Fastpath 121
- Adabas Native SQL 122
- Adabas Online System 123
- Adabas Parallel Services 125
- Adabas Review 125
- Adabas SAF Security 128
- Adabas SQL Gateway 129
- Adabas Statistics Facility 130
- Adabas Text Retrieval 132
- Adabas Transaction Manager 133
- Adabas Vista 134
- Event Replicator for Adabas 135
- Entire Net-Work Multisystem Processing Tool 139
- Natural Application Development Environment 141
- Predict Data Dictionary System 142

The add-on products discussed in this chapter are available to Adabas customers who have exercised a separate purchase agreement for the feature or product.

Adabas Bridges

Adabas Bridge technology allows you to access the DL/I (and IMS/DB) and VSAM application development environments efficiently. Emulation requires no modifications to application programs and the delay and expense of traditional conversion are avoided.

Adabas Bridge technology provides:

- user application transparency (it is not necessary to change any existing application programs or third party application software using native VSAM or DL/I calls);
- support for batch and online processing environments and the RPG, COBOL, PL/I, FORTRAN, and Assembler programming languages;
- data and application integrity.

Adabas Bridge for VSAM

Adabas Bridge for VSAM (AVB) allows application software written to access data in the VSAM environment to access data in an Adabas environment. It operates either in batch mode or online (under CICS).

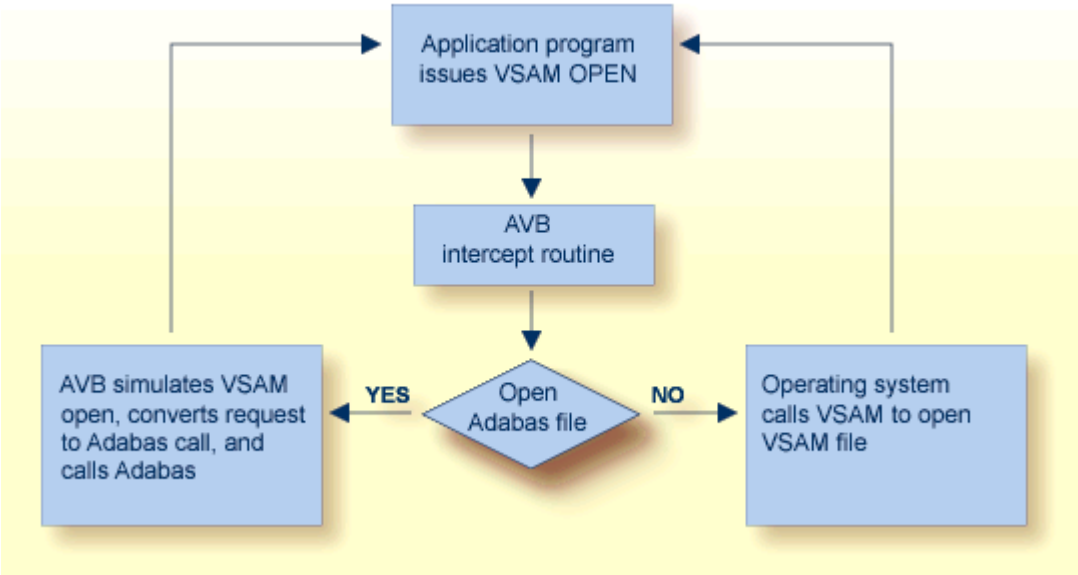
AVB may be executed in an environment with Adabas files only; VSAM files only; or both Adabas and VSAM files (mixed environments). The ability to operate in a mixed environment means that your migration schedule can be tailored to your needs and resources. You can migrate files from VSAM to Adabas as needed, one application or even one file at a time.

AVB uses a transparency table to map the names and structures of VSAM files to the numbers and structures of corresponding Adabas files. Once a VSAM file has been migrated to Adabas and defined in the AVB transparency table, it can be bridged to Adabas. When a VSAM file is bridged, AVB converts each request for the VSAM file to an Adabas call; the Adabas file is accessed instead of the VSAM file.

When AVB is active, it intercepts each file OPEN and CLOSE request and performs a series of checks to determine whether to process the request against an Adabas file. If not, AVB passes the request to the operating system to open the referenced VSAM file.

When AVB detects an OPEN or CLOSE for a bridged file, it converts it to an Adabas command and calls Adabas to open or close the corresponding Adabas file. After the OPEN, all requests to read or update the VSAM file are passed directly to AVB.

AVB allocates VSAM control blocks and inserts information the application needs to process results as if they were returned from a standard VSAM file request. After the Adabas call, AVB returns the results to the application using standard VSAM control blocks and work areas.



Advantages of Adabas over VSAM

The availability of Adabas in an environment in which only VSAM file structures were previously used results in the following benefits:

- Applications can be extended with the powerful indexing facilities of Adabas. You can query, retrieve, and manipulate data using efficient views and paths.
- Applications can be extended with programming languages such as Natural and SQL.
- Application programs are independent of the data structure, which reduces maintenance costs and increases programmer productivity.
- Automatic restart/recovery ensures the physical integrity of the database in the event of a hardware or software failure.
- Data compression significantly reduces the amount of online storage required and allows you to transmit more information per physical I/O.
- Security is improved by password protection at both the file and field levels and, on the basis of data values, at the record level as well.
- Adabas provides encryption options, including a user-provided key that drives the encryption process.

After migration, your application programs have the same view of data as before, but you can structure the new Adabas files to optimize the benefits outlined above.

The tables that identify files to Adabas are external to the applications and may be changed without relinking the application programs. This feature is especially useful when you want to change file or security information, or move applications from test to production status.

Adabas Bridge for DL/I (and IMS/DB)

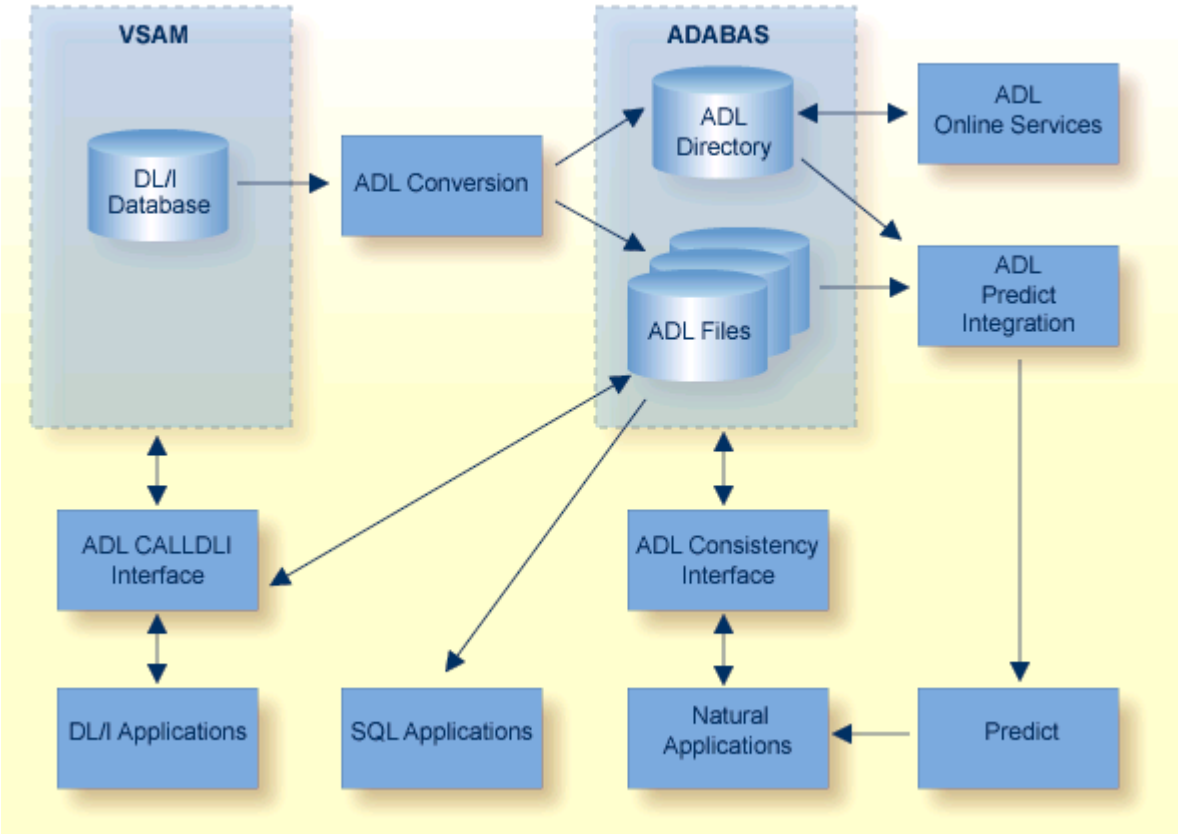
Adabas Bridge for DL/I (ADL) is a tool for migrating DL/I or IMS/DB databases into Adabas. The term DL/I is used as a generic term for IMS/VS and DL/I DOS/VS. ADL operates either in batch mode or online (under CICS or IMS/DC).

DL/I applications can continue to run without change: the migrated data can be accessed by Natural and by SQL applications if the Adabas SQL Gateway is available. ADL can also be used to run standard DL/I applications against Adabas databases.

Functional Units

ADL comprises six major functional units:

- A collection of conversion utilities automatically converts DL/I databases into Adabas files called ADL files to emphasize the special properties of these files as opposed to native Adabas files.
- As a result of the conversion process, the DL/I database definitions (DBDs) and the related Adabas file layouts are stored on an Adabas file called the ADL directory, which also contains the ADL error messages and other information.
- A menu-driven application written in Natural provides a number of online services, including reports on the contents of the ADL directory.
- A special set of integration programs uses data from the ADL directory to generate Predict definitions for ADL files, which can then be used to generate Natural views.
- A call interface allows DL/I applications to access ADL files in the same way as original DL/I databases (and both concurrently in mixed mode). It supports Assembler, COBOL, PL/I, RPG, FORTRAN, and Natural for DL/I. A special precompiler is provided for programs using the EXEC DL/1 interface.
- A consistency interface provides access to ADL files for Natural applications or programs using Adabas direct calls. This interface preserves the hierarchical structure of the data, which is important for ongoing DL/I applications.



Advantages of Adabas over DL/I

The availability of Adabas in an environment in which only DL/I file structures were previously used results in the following benefits:

- Data can be manipulated by Natural, Software GmbH's fourth generation language;
- Data is compressed at field level automatically by Adabas;
- Deleted data records are released from storage immediately. This is in contrast to DL/I, which simply sets a flag in such records but does not release the storage used by them. With Adabas, the released space can be reused immediately for new records: there is no requirement to maintain records that are marked as deleted, and less reorganization of the database is required;
- All converted DBDs have full HIDAM functionality, regardless of the original access method;
- The length of a field can be increased without unloading and reloading the data;
- Segments can be added to the end of a DBD without unloading and reloading the data;
- Trace facilities are available for online and batch;
- The batch CALLDLI test program is available.
- Since Adabas (unlike DL/I) does not run in the CICS region, online system resources are reduced.
- HD databases are available under z/OS.

Adabas Caching Facility

Adabas Caching Facility helps improve system performance and make full use of ESA functions by augmenting the Adabas buffer pool in extended memory, data space, hiperspace, and 64-bit virtual storage.

Adabas Caching Facility augments the Adabas buffer manager by reducing the number of read Execute Channel Programs (EXCPs) to the database. This allows you to use the available operating system facilities without monopolizing valuable virtual memory resources.

Note: Write EXCPs are always issued to maintain the integrity of the database.

Adabas Caching Facility is functionally similar to the Adabas buffer manager, but offers the following additional capabilities:

- User-specified RABNs (blocks) can be cached or fenced to make them readily accessible when demand arises, even though the activity against them is not sufficient to keep them in the active buffer pool. RABN-fencing reduces the required I/O response time if the Adabas nucleus needs to reread those RABNs.
- The Adabas Work data set parts 2 and 3 can be cached to improve performance in environments that service large numbers of complex queries. Adabas Work parts 2 and 3 serve as temporary work areas used to resolve and maintain the ISN lists of complex queries. Reducing the number of read and write EXCPs to Work parts 2 and 3 for these complex queries can decrease processing time dramatically and improve performance substantially.
- A file or range of files can be specified to cache all associated RABNs. It is also possible to cache only Associator or Data Storage blocks if required. Files can be prioritized by assigning a class of service which determines the percentage of the maximum available cache space that a given file can use and when the file's RABN blocks will be purged from the cache.
- Operator commands are available to dynamically respond to a changing database environment by modifying
  - the RABNs to be cached by RABN range, file, or file range;
  - the RABNs to be enabled or disabled by RABN range, file, or file range;
  - when to acquire and release the system resources used by the Adabas Caching Facility.
- When processing serial Adabas commands (e.g., read logical, read physical, histogram, and searches using non-descriptors) with the read-ahead caching option, a single EXCP is issued to read all the consecutive ASSO and/or DATA blocks that reside on a single track of the disk device. The blocks are kept in cache and are immediately available when the nucleus requests the next block in a sequence. This feature may enhance performance by reducing the number of physical read I/Os for a 3390 ASSO by as much as 17:1.

The integrity of the database is preserved because Adabas RABNs are not kept redundantly in both the Adabas buffer pool and the cache. An Adabas RABN may reside either in the Adabas buffer pool or in the cache area, but never in both. All database updates and consequently all buffer flushes occur only from the Adabas buffer pool. Unlike other caching systems, this mechanism of non-redundant caching conserves valuable system resources.

The demo or full version of Adabas Online System is required to use the online cache-maintenance application Cache Services. For more information, see the *Adabas Caching Facility* documentation.

Adabas Cluster Services

Adabas Cluster Services implements multinucleus, multithread parallel processing and optimizes Adabas in an IBM Parallel Sysplex (systems complex) environment. The Adabas nuclei in a sysplex cluster can be distributed to multiple z/OS images that are synchronized by a Sysplex Timer® (IBM). One or more Adabas nuclei may be run under a z/OS image.

Adabas Cluster Services comprises software components that ensure intercommunicability and data integrity among the z/OS images and the associated Adabas nuclei in each sysplex nucleus cluster. An unlimited number of sysplex clusters each comprising up to 32 clustered nuclei can reside on multiple z/OS images in the sysplex.

In addition to the increased throughput that results from parallel processing, Adabas Cluster Services increases database availability during planned or unplanned outages; the database can remain available when a particular operating system image or cluster nucleus requires maintenance or goes down unexpectedly.

To support a cluster environment that includes more than one operating system image, a limited Software AG Entire Net-Work library is included as part of Adabas Cluster Services (see also ***Entire Net-Work Multisystem Processing Tool***, elsewhere in this guide). Entire Net-Work is used to send Adabas and Adabas Cluster Services commands back and forth between z/OS images. It provides the communication mechanism among the nuclei in the sysplex cluster. No changes have been made to Entire Net-Work to accommodate Adabas Cluster Services.

The ADACOM module is used to monitor and control the clustered nuclei. For each cluster, the ADACOM module must be executed in each z/OS image that either has a nucleus that participates in the cluster or has users who access the cluster database.

The Adabas Cluster Services SVC component SVCCLU is prelinked to the Adabas SVC and is used to route commands to local and remote nuclei. CSA space is used to maintain information about local and remote active nuclei, and currently active users.

The sysplex cache structure is used to hold ASSO/DATA blocks that have been updated during the session. It synchronizes the nuclei, users, and the z/OS images; ensures data integrity, and handles restart and recovery among the nuclei.

Cluster Services With Other Adabas Products

Adabas Online System communicates with all nuclei within the sysplex cluster.

Adabas Caching Facility supports clustered nuclei and can provide a performance boost to the cluster.

Advantages of Using Entire Net-Work

In an Adabas Cluster Services environment, Entire Net-Work allows users on various network nodes to query a logical database across multiple z/OS images. Users access a cluster database as they would a conventional, single-node database.

A request to Adabas can be made from within an existing application, without change. The request is processed automatically by the system; the logistics of the process are transparent to the application.

Entire Net-Work ensures compatibility by using Adabas-dependent service routines for the operating system interface as well as for interregion communication. Job control statements for running Entire Net-Work are much like those needed to run Adabas. For example, the EXEC statement invokes the ADARUN program for Entire Net-Work just as it does for Adabas, and the ADARUN parameters for Entire Net-Work are a subset of Adabas parameters.

Because status information is broadcast to all nodes whenever a target or service establishes or terminates communication with the network, there is no need to maintain or refer to database or target parameter files at a central location.

Allowing only one Entire Net-Work task on each node enforces control over the network topology by maintaining all required information in one place. This avoids confusion in network operation and maintenance. If, however, more than one Entire Net-Work task is required, this can be accomplished by installing additional routers.

Each Entire Net-Work node maintains only one request queue and one attached buffer pool for economical use of buffer storage. All buffers that are not required for a particular command are eliminated from transmission. In addition, only those portions of the record buffer and ISN Buffer that have actually been filled are returned to the user on a database reply.

Buffer size support in Entire Net-Work is comparable to that in Adabas, ensuring that all buffer sizes that are valid for Adabas can also be transmitted to remote nodes.

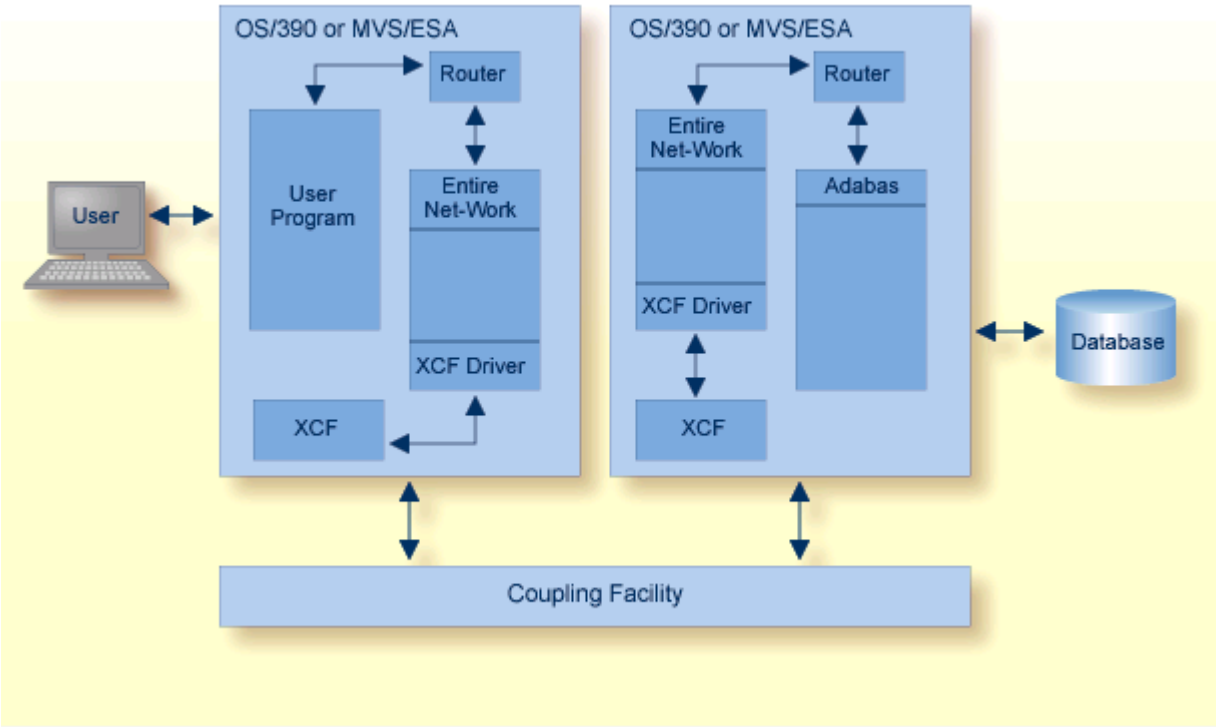
- Entire Net-Work XCF Option

Entire Net-Work XCF Option

Actual network data traffic is controlled by the Entire Net-Work XCF Option, an interface to IBM's cross-system coupling facility (XCF) which allows authorized applications on one system to communicate with applications on the same system or on other systems. XCF transfers data and status information between members of a group that reside on one or more z/OS images in a sysplex.

The Entire Net-Work XCF Option, which is installed on each Entire Net-Work node, provides high performance, transparent communications between z/OS images that reside on different central processors in the sysplex. Multiple connections to other nodes are supported, and the line driver's modular design permits easy addition of new access method support to the system.

A *member* is a specific function (one or more modules/routines) of a multisystem application that is defined to XCF and assigned to a group by the multisystem application. A member resides on one z/OS image in the sysplex and can use XCF services to communicate (send and receive data) with other members of the same group. Each Entire Net-Work node running the XCF line driver is identified as a different member in a group specifically set up for Entire Net-Work connectivity.



Adabas Delta Save Facility

Adabas Delta Save Facility (DSF) offers significant enhancements to ADASAV utility processing by backing up only the changed (delta) portions of Adabas databases. It reduces the volume of save output produced and shortens the duration of save operations; this increases database availability. By allowing more frequent save operations to be performed, it also reduces database recovery time.

Adabas Delta Save Facility provides for:

- more frequent saves without interrupting database availability;
- enhanced 24-by-7 operation;
- full offline saving parallel with the active database; and
- short REGENERATE duration during recovery.

Adabas Delta Save Facility achieves these objectives by saving only those Associator and Data Storage blocks that have changed (delta portion) since the last save operation. The result of this operation is called a *delta save tape*. Because a much smaller volume of output is written to delta save tapes, contention for secondary (tape, cassette etc.) storage is reduced.

Adabas Delta Save Facility can:

- maintain a log of changed database blocks (RABNs);
- create and merge interim delta save tapes while the database remains online, if required;
- consolidate delta save tapes with the most recent database save tape to create an up-to-date full save tape;
- restore the database from the most recent full save tape and all subsequent delta save tapes.

Adabas Delta Save Facility is intended for Adabas sites with one or more large, heavily updated databases that need to be available most of the time. It is particularly beneficial when the volume of data changed on a day-to-day basis is considerably smaller than the total database volume.

The demo or full version of Adabas Online System is required to use DSF. For more information, see the *Adabas Delta Save Facility Facility* documentation.

Adabas Encryption

Adabas Encryption is an out-of-the-box, selectable unit of Adabas that encrypts database container datasets (ASSO, DATA, WORK, and so on) at the dataset level. It leverages the security and performance benefits of hardware-based encryption and the key management facilities provided by the IBM Z platform.

Encryption of datasets on DASD is an effective way to strengthen their security:

- Separate one user's authorization to access sensitive data in datasets from another user's authorization to manage those datasets. Dataset management includes tasks like backing up, migrating, or replicating datasets.
- Protect datasets on shared DASD against accesses from other systems that can have different access control rules (in RACF, for instance). This is important, for example, when the datasets reside in a storage area network (SAN).
- Ensure dataset security across all storage tiers, from a single DASD device up to cloud storage.

The use of Adabas Encryption is transparent to existing Adabas application programs. No application changes are required for installing and using Adabas Encryption.

Adabas Encryption encrypts datasets on DASD (namely, the database container datasets). It does not encrypt datasets on tape (e.g. save tapes) or data in memory (e.g. the Adabas buffer pool).

Adabas Fastpath

Adabas Fastpath optimizes response time and resources by bringing reference data closer to the user, reducing overhead, and reducing response times by servicing requests locally (that is, within the same region).

Fastpath satisfies an Adabas query from within the application process, thus avoiding the operating system overheads needed to send a query to and from the database. Database activity such as command queue processing, format pool processing, buffer pool scanning, and decompression are also avoided.

Fastpath uses a query sampler to efficiently identify:

- the most commonly issued direct access queries; that is, queries where the client identifies the data being sought (e.g., ISN, search value).
- sequential access queries; that is, queries where the client identifies a series of data items related by sequence or search criteria.

The query sampler reports interactively and at shutdown the exact types of queries that can be optimized and their relative popularity.

For each type of query, Fastpath uses algorithms to recognize and retain the most popular data and discard or overwrite the least popular data. Given a particular amount of memory to use that is available to all clients within an operating system, Fastpath retains the results of popular data queries so that they can be resolved in the client process when repeated. The retained results comprise a common knowledge base that reflects the experience gained from past queries. The knowledge base is dynamic in that it is continually updated; the least popular data held there is discarded or overwritten.

A Fastpath component attached to the DBMS ensures that any changes to popular data are reflected in the results returned to the knowledge base. Fastpath data is always consistent with the DBMS data.

Before a query is passed to the DBMS, the Fastpath optimizer attempts to resolve the query from the knowledge base. If successful, the query is satisfied faster, without interprocess communication or DBMS activity. Fastpath optimizes sequences by dynamically applying Adabas prefetch read-ahead logic to reduce DBMS activity. As many as 256 data items can be retrieved in a single visit to the DBMS.

Fastpath optimization occurs in the client process, but requires no change to application systems. Different optimization profiles can be applied automatically at different times of the day. Once started, the Fastpath buffer can be left active without intervention; Fastpath reacts automatically to DBMS startup and shutdown.

For more information, refer to the *Adabas Fastpath* documentation.

Adabas Native SQL

Adabas Native SQL is Software GmbH's high-level, descriptive data manipulation language (DML) for accessing Adabas files from applications written in Ada, COBOL, FORTRAN, and PL/I.

Database access is specified in an SQL-like syntax embedded within the application program. The Adabas Native SQL precompiler then translates the SQL statement into a transparent Adabas native call.

Software GmbH's Adabas, Natural, Predict, and the Software AG Editor are prerequisites for Adabas Native SQL, which makes full use of the Natural user view concept and the Predict data dictionary system to access all the facilities of Adabas.

Using your Natural field specifications, Adabas Native SQL automatically creates Ada, COBOL, FORTRAN, or PL/I data declarations with the correct set of fields, field names, field sequence, record structure, field format and length.

Adabas Native SQL uses information about file and record layouts contained in Predict to generate the data structures that the generated Ada, COBOL, FORTRAN, or PL/I program needs to access the database. Then, as Adabas Native SQL processes the program, it records in Predict active cross-reference (Xref) information including the names of the files and fields that the program accesses.

These features help to eliminate the risk of writing incorrect data declarations in programs that access the database. In addition, they create comprehensive records in the data dictionary that show which programs read from the database and which programs update it, providing the DBA with an effective management tool.

For more information, refer to the *Adabas Native SQL* documentation.

Adabas Online System

Note: Adabas includes a demo version of the Adabas Online System to illustrate its capabilities and to provide access to selected other services.

Adabas Online System (AOS) provides an online database administration tool for Adabas. The same functionality is available using a batch-style set of utilities.

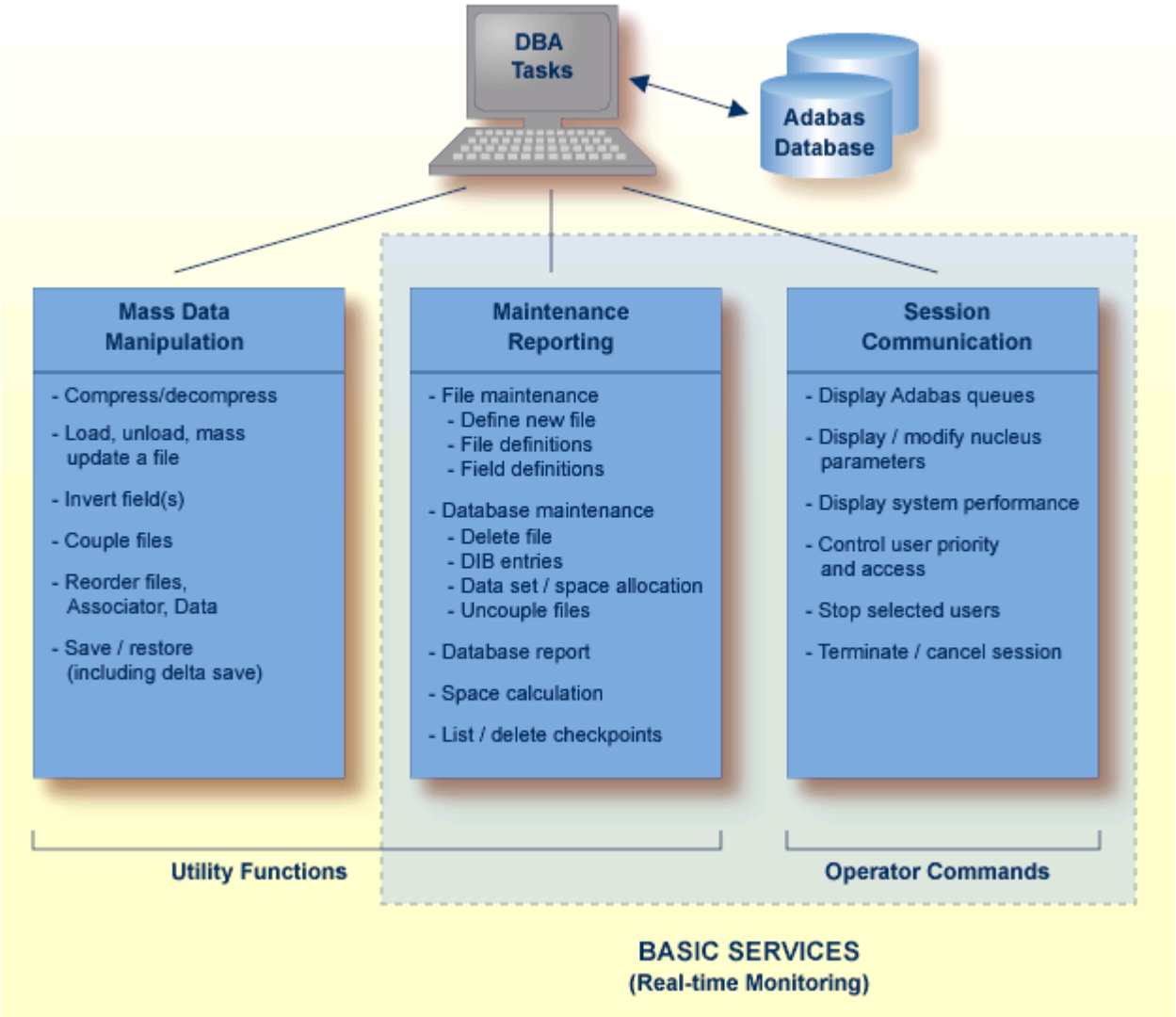
AOS is an interactive menu-driven system providing a series of services used for online Adabas database analysis and control. These services allow a database administrator (DBA) to:

- display Adabas user statistics, monitor and control access and operation of one or all users;
- display and modify Adabas fields and files: add fields, allocate and remove file space, change file and database layout, view and remove field descriptors;
- restrict file use to utility users only, or lock/unlock file access completely.

Support is also provided for nucleus cluster environments as well as SAF security. In addition, AOS has the ability to dynamically modify ADARUN parameters.

AOS is written in Natural, Software GmbH's fourth generation application development facility. AOS security functions are available only if Software GmbH's Natural Security is installed and operating.

AOS includes functions that are comparable to the Adabas operator commands and utilities.



Basic Services makes it possible for the DBA to interactively monitor and change aspects of an Adabas database while an Adabas session is active. Using menu options or direct commands, the DBA can view resource status and user queues, display and revise space allocation, change file and database parameters, define a new file online, and stop a selected user or current Adabas session.

AOS is delivered in a separate data set/library from Adabas. In the initial Adabas delivery, AOS functions as a demonstration system. For full functionality, the contents of the AOS load library must be copied to the Adabas load library, or the AOS load library must be concatenated in the STEPLIB of the Adabas nucleus JCL. In addition, all AOS users (demo or full version) and Predict users must include the load module AOSASM from the Adabas load library in the link of the Natural nucleus.

For more information, refer to the *Adabas Online System* documentation.

Adabas Parallel Services

Adabas Parallel Services implements multinucleus, multithread parallel processing and optimizes Adabas in a multiple-engine processor environment on a single operating system image.

Up to 31 Adabas nuclei in an Adabas Parallel Services *cluster* are distributed over the multiple engines that are synchronized by the operating system.

All nuclei in the cluster access a single physical database simultaneously. A single physical database is one set of Associator and Data Storage data sets identified by a single database ID number (DBID).

The nuclei communicate and cooperate with each other to process a user's work. Compression, decompression, format buffer translation, sorting, retrieving, searching, and updating operations can all occur in parallel.

In addition to the increased throughput that results from parallel processing, Adabas Parallel Services increases database availability during planned or unplanned outages: the database can remain available when a particular cluster nucleus requires maintenance or goes down unexpectedly.

An unlimited number of Adabas Parallel Services clusters can operate in the same operating system image under the same or different Routers or SVCs; that is, an unlimited number of separate databases can be processed, each with its own Adabas Parallel Services cluster of up to 31 nuclei.

Applications see only one database target; no interface changes are required. Applications still communicate with their intended databases and communicate with an Adabas Parallel Services cluster of nuclei without modification.

For more information, refer to the *Adabas Parallel Services* documentation.

Adabas Review

Adabas Review (formerly Review Database) provides a set of monitoring, accounting, and reporting tools that enable you to monitor the performance of the Adabas environment and the applications executing within them.

Information retrieved about Adabas usage helps you tune application programs to achieve maximum performance with minimal resources.

In addition to the local mode with Adabas Review running in the Adabas address space, Adabas Review offers the hub mode, a client/server approach to the collection of performance data for Adabas:

- the Adabas Review interface (the client) resides on each Adabas nucleus.
- the Adabas Review hub (the server) resides in its own address space or region.

Cluster Services statistics gathering is supported. Both file-level (CF) caching statistics and Cluster Services locks can be monitored over a period of time and at user-defined intervals. Statistics are written to the Review History file and can be retrieved or viewed using the power of Review reporting facilities.

For additional, information, refer to the *Adabas Review* documentation.

- The Hub Server
- The Interface Client
- Interface Calls
- Example Client/Server Environment

The Hub Server

The Adabas Review hub is the data collector and the reporting interface for the user. The hub handles the data consolidation and reporting functions for monitoring an Adabas database, including usage information related to applications, commands, minimum command response time (CMDRESP), I/O activity, and buffer efficiency.

An interactive reporting facility allows you to pinpoint problems quickly, providing detailed and summary data about Adabas activities. Specific information about each database is also available.

Proven Adabas and Review components are combined in the centralized collection server (hub) with the following advantages:

- A single hub can collect information from multiple Adabas nuclei and from Adabas nucleus clusters managed by either Adabas Parallel Services or Adabas Cluster Services. This means that the number of Adabas Review nuclei required to support an enterprise-wide distribution of Adabas nuclei is minimized, resource requirements are minimized, and performance increases.
- Removing the Review subtask from the address space or region of each Adabas nucleus improves the performance of the Adabas main task. At the same time, the isolation minimizes the impact of future Adabas releases on the functioning of Adabas Review.

The hub comprises

- ADAREV, a logic module that manages and supervises the incoming Review data calls and requests;
- REVHUB, a module to establish and maintain the environment for Adabas Review; and
- the Review nucleus and subsystems including RAOSAUTO, the autostarted report parameter generation routine, and RAOSHIST, the historical data population routine.

The Interface Client

The Adabas Review interface constructs and then transmits the Review data from the Adabas nucleus to the Adabas Review hub. An Adabas Review interface is integrated with each Adabas nucleus that is monitored.

The interface utilizes the existing Adabas interregion communication process; that is, ADALNK, ADASVC, and ADAMPM. This communication process is consistent across supported platforms.

When all supported platforms and systems are networked correctly, Adabas Review supports a multiple platform, multiple operating system, Adabas database environment.

The interface comprises the following:

- ADALOG, the Adabas command logging module;
- RAOSDAEX, the Adabas Review command log extension module that is responsible for acquiring additional information not present in the Adabas command log record; and
- ADARVU, which handles the environment conditions for RAOSDAEX and the Adabas API requirements for transmitting the Review data to the Adabas Review hub.

Interface Calls

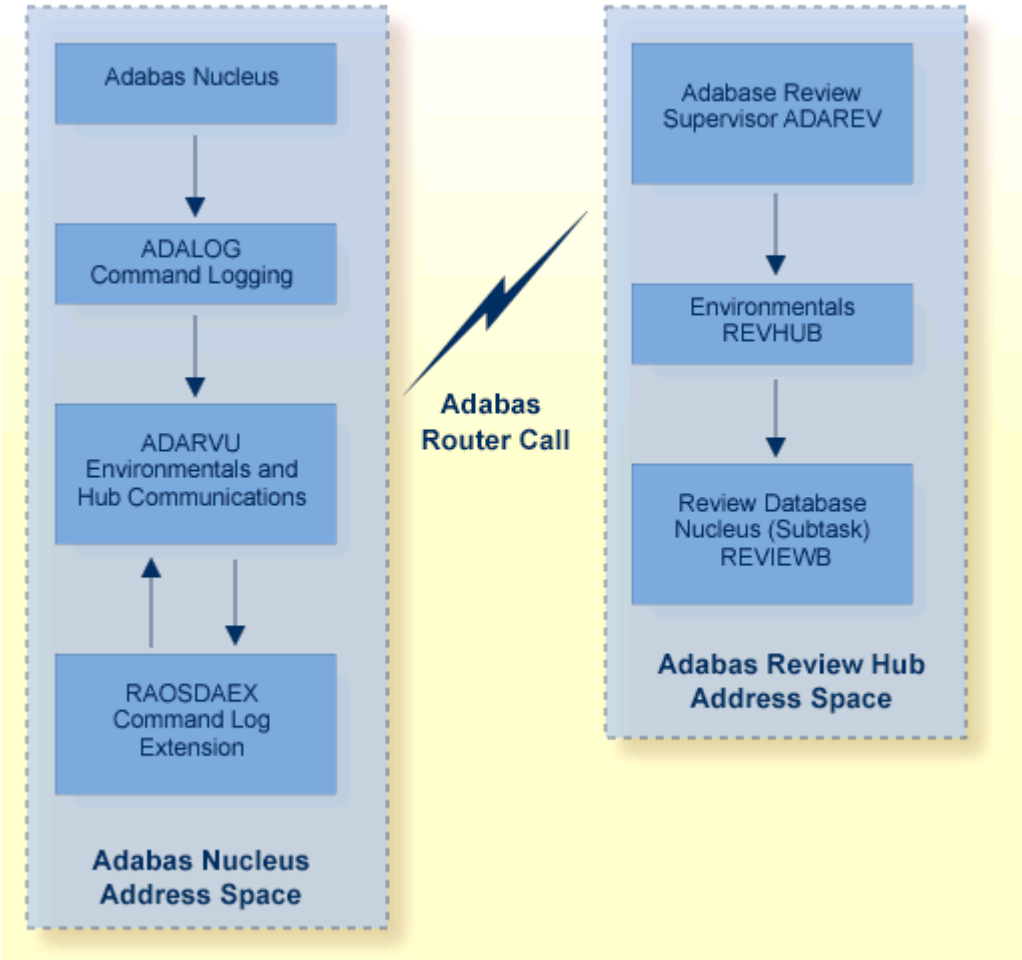
To maximize performance, the ADARVU module issues an optimistic call from an Adabas nucleus to the Adabas Review hub without waiting for a completion or post from the hub; ADARVU assumes that the Review data was successfully passed to the hub.

However, ADARVU does perform an initialization step to ensure that the hub is active prior to any command processing by the Adabas nucleus. If the hub is not active, ADARVU informs you using WTOs or a user exit. If a user exit is used, you are given the option to wait for the hub to be activated, or continue initialization and call the hub only when it is active.

On the hub side of the call, the elimination of the cross-memory post call enhances performance by reducing the overhead of active communication with the Adabas clients. This allows the hub to remain a passive data collector.

Example Client/Server Environment

The following figure shows the major components of the Adabas Review interface (Adabas nucleus address space) and the Adabas Review hub (Adabas Review hub address space) in a client/server architecture.



Adabas SAF Security

The System Authorization Facility (SAF) is a z/OS interface which provides rigorous control of the resources available to a user or group of users. SAF-based security packages such as IBM RACF, CA-ACF2, and CA-Top Secret allow the system administrator:

- To maintain user identification credentials such as User ID and password.
- To establish profiles determining the datasets, storage volumes, transactions, and reports available to a user.

The resulting security repository and the infrastructure to administer it represent a significant investment. At the same time, the volume of critical information held by a business is constantly growing, as is the number of users referencing the data. The challenge of controlling these ever-increasing accesses requires a solution that is flexible, easy to implement and, above all, one that safeguards the company's investment.

Adabas SAF Security enhances the scope of SAF-based security packages by integrating Adabas resources into the central security repository. Adabas SAF Security enables:

- A single control and audit system for all resources.
- Industry-standard protection of Adabas data.
- Maximized return on investment in the security repository.

Adabas SQL Gateway

The Adabas SQL Gateway is a subset of CONNX, a unique client/server connectivity programming tool set that makes it possible to use computers in real-time interactive operation with many databases. The CONNX data access engine is unique in that it not only provides access to the databases, but it presents them as one enterprise-spanning relational data source. CONNX also offers ODBC SQL Level 2 compatibility, additional security, metadata management, enhanced SQL capability, views, heterogeneous joins, bidirectional data conversion, and enables read/write access to the data. Such technology can be used in data warehousing, data integration, application integration, e-commerce, data migration, and for reporting purposes. The technology also has a place within companies seeking to make use of disparate data sources, that need to Web-enable their data, or that have older applications storing mission-critical information.

CONNX includes the following components:

- CONNX Data Dictionary (CDD)
- CONNX ODBC Driver

  CONNX OLE RPC Server (Not implemented for CONNX and VSAM)
- CONNX Host Data Server (RMS, VSAM [Implemented as CICS/C++ TCP/IP Listener/Server], C-ISAM, Rdb, and DBMS)
- CONNX JDBC Driver (Thin Client)
- CONNX JDBC Server
- CONNX JDBC Router

In addition to other databases, CONNX supports Adabas on IBM z/OS, Windows 2000/XP/2003, AIX, Solaris, Linux, HP-UX, and other platforms.

For complete information about the Adabas SQL Gateway, read the *Adabas SQL Gateway* documentation.

Adabas Statistics Facility

A database administrator (DBA) regularly checks the status of the database (such as disk and memory utilization) and plans for the long term, such as ensuring that the future disk space requirements can be met, based on current trends.

For Adabas, the DBA can check the status of individual databases and files using the ADAREP (Database Report) utility, the nucleus end session protocol, and ad hoc inquiries made using the Adabas Online System. This is often a time consuming process.

The Adabas Statistics Facility (ASF) provides an automated environment comprising:

- A *store program* that collects database status information during an active nucleus session. The store program is normally scheduled to run at regular intervals (for example once per day) over a period of many weeks or months to collect data that can be statistically evaluated. The store program can also be started by the DBA on an ad hoc basis, using commands in the ASF online menu system.
- A set of *evaluation programs* to interpret the statistics gathered by the store program and publish summary evaluation reports to either the screen or a hardcopy printer. Reports may also be downloaded to a PC using Entire Connection.

Database information can be collected at the start, at the end, and during a nucleus session. The start and end nucleus data, when accumulated over periods of weeks or months, gives an indication of long term database growth and permits projections of future database requirements. The nucleus performance data, such as main memory and pool usage, permits the DBA to analyze and tune the Adabas nucleus parameters.

For more information, refer to the *Adabas Statistics Facility* documentation.

Data Collection Program

ASF uses a data collection program called the *store program* to collect database status information at the start of, at the end of, or during an active nucleus session. This program is normally scheduled to run as a batch program at regular intervals (perhaps once per day) over a period of weeks or months to collect data that can be statistically evaluated. The store program can also be started by the DBA on an ad hoc basis, using commands in the ASF online menu system.

The DBA defines store profiles, each specifying a different set of databases and files to be monitored, that are specified as input to the store program when it runs. Several store programs, each with a different store profile, can run concurrently.

Approximately 170 criteria called data fields are used for monitoring Adabas databases and files. The data fields represent aspects of an Adabas database such as disk and buffer usage, thread

usage, database load, ADARUN parameters, pool usage, and frequency of use of particular Adabas commands. All data fields are stored for each database and file specified in the store profile.

Start and end nucleus data, when accumulated over periods of weeks or months, give an indication of the long term database growth, and permit projections of future database requirements. Nucleus performance data, such as main memory usage and pool usage, provide information for tuning Adabas nucleus parameters.

Data Evaluation Programs

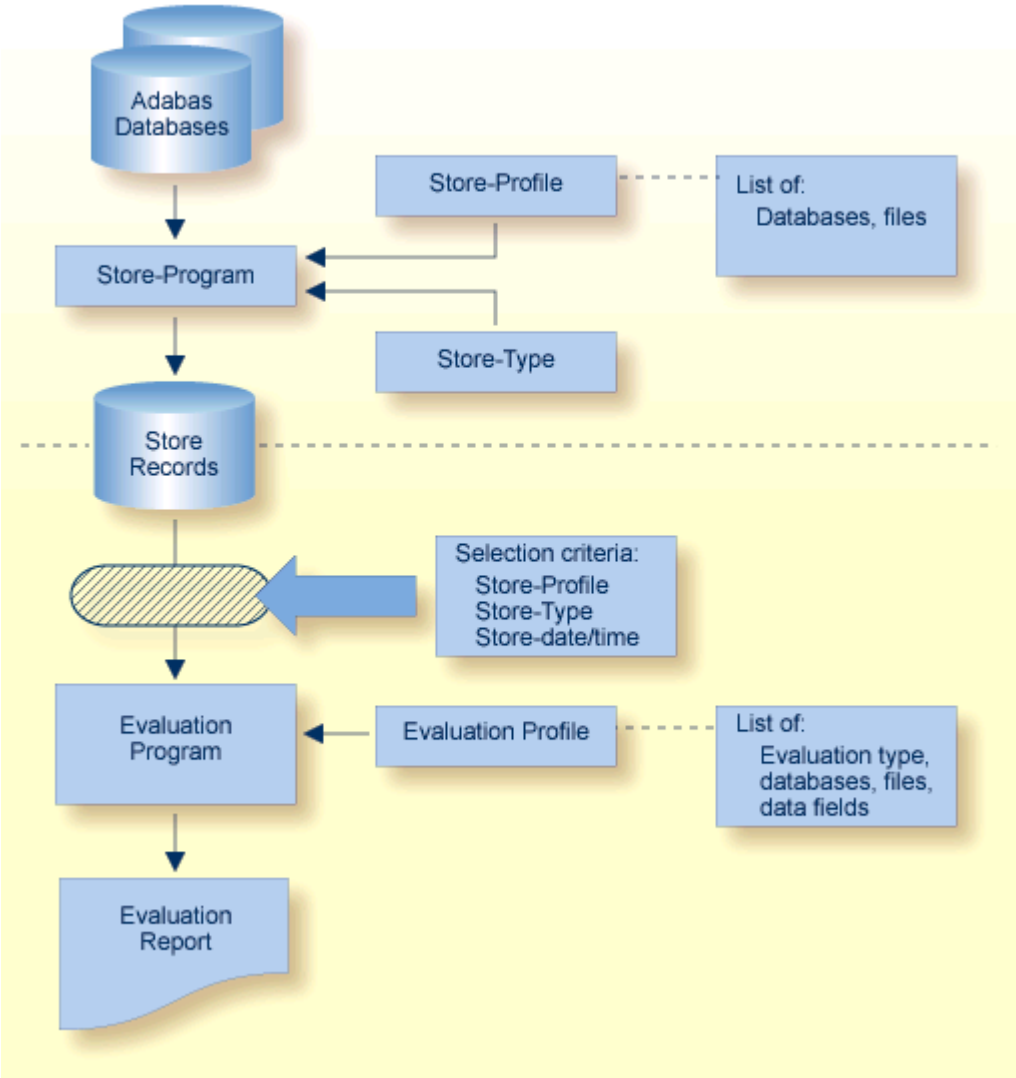
ASF uses a set of programs called *evaluation programs* to evaluate the statistics gathered by the Store Program and produce summary reports called evaluation reports that can be viewed online, printed, or downloaded to a PC.

For each evaluation report, the DBA uses the online menu system to define an evaluation profile specifying

- the databases and files for which data is to be evaluated (data for these must have been collected by a store program);
- for each database and file specified, one or more data fields to be analyzed in the report;
- the units of measurement of the data fields specified;
- upper and lower values representing critical levels for the data fields specified; and
- one of ten report types (01-10) which determines the format of the report heading.

The main types of reports are

- General evaluation: an analysis of the past and present database status. The statistical tables generated provide an overview of the status of various databases and files. The maximum and minimum values in rows of the output tables can be displayed, as well as statistical quantities such as the sum and average of the values.
- Trend evaluation: tables of projected statistics, in time steps of days, weeks, or months, until a specified end date.
- Critical Report: a report of databases and files for which the specified data fields have reached or exceeded the specified critical limits.
- Critical Trend Report: a report of databases and files for which the specified data fields will reach or exceed their critical limits within a time frame, based on an extrapolation of current trends.



Adabas Text Retrieval

Adabas Text Retrieval is an extension of Adabas that allows you to develop applications that access both formatted and unformatted (that is, text) data simultaneously. Adabas Text Retrieval manages the index information and not the content of the data, making it possible to store the document contents at any location: Adabas, sequential files, CD-ROM, PC, etc. The call interface for Adabas Text Retrieval can be embedded in Natural, Software GmbH's fourth generation application development environment, or in any third generation language such as COBOL or PL/I.

Documents comprise chapters designated as either free text to be managed by Adabas Text Retrieval or formatted fields to be managed by Adabas itself. Free-text chapters comprise paragraphs and sentences, which can be individually searched.

Text substrings of an entered text are identified or tokenized by identifying a character defined in the Adabas Text Retrieval character table, by using algorithms to identify characters based on their contexts, or by using translation tables to sort or limit previously identified characters.

Document index entries are created when unformatted data is inverted. The full text can be inverted, or the inversion can be limited by a controlled thesaurus or by ignoring words in a stopword list.

Searches can be based on words, parts of words (right/left/middle truncations are possible), phonetics, synonyms, integrated thesaurus relations (broader/narrower terms), proximity operators (adjacent, near, in sentence, in paragraph), relational operators, Boolean operators, references to previous queries (refinement), or sorts (ascending or descending). Searches can be independent of structure, meaning that they may encompass any combination of free text and formatted fields. Search returns can be highlighted.

Natural users can expand the functions of Adabas Text Retrieval using Natural Document Management, which provides complete document management services.

Adabas Transaction Manager

Adabas Transaction Manager introduces distributed transaction support to Adabas. Transactions may be distributed over multiple Adabas databases in one or more operating systems (connected by Entire Net-Work). Transactions may also be distributed over non-Adabas DBMSs such as DB2, IMS, and so on. Where transactions are distributed over a non-Adabas DBMS, Adabas Transaction Manager must be configured to inter-operate with other transaction coordinators such as IBM's CICS Syncpoint Manager and/or IBM's Recoverable Resource Management Services. At any time, Adabas Transaction Manager can account for in-flight transactions, suspect transactions, participating databases, and more.

Adabas Transaction Manager (ATM) is an Adabas add-on product that:

- coordinates changes to Adabas databases participating in a global transaction;
- (function not available in first release) processes two-phase commit directions from transaction managers that take a higher-level, controlling role in coordinating global transactions such as IBM's RRMS or the CICS Syncpoint Manager allowing transactions to encompass both Adabas and non-Adabas databases operating within a single operating system image; and
- plays a key role in coordinating global transactions that change Adabas databases on more than one system image. In this case, the communication mechanism between the components across the system images in Entire Net-Work.

Each Adabas Transaction Manager instance (one per operating system image) executes in its own address space as a special kind of Adabas nucleus. Each Adabas Transaction Manager is aware of and partners with the other Adabas Transaction Managers in the distributed system and the databases they coordinate. At any time, each Adabas Transaction Manager can account for the status of the global transactions it is coordinating.

Adabas Transaction Manager addresses two basic needs of the enterprise object revolution:

- the need to deliver industrial strength enterprise objects for widespread commercial use in mainstream, critical business systems; and
- the need to spread the masses of data that Adabas customers manage more evenly across the computer(s) and organization.

Adabas Transaction Manager includes an online administration system based on Natural and available through Adabas Online System.

For more information, refer to the *Adabas Transaction Manager* documentation.

Adabas Vista

Adabas Vista allows you to partition data into separately managed files without reconstructing your business applications, which continue to refer to one (simple) Adabas file entity even though the physical data model is partitioned and possibly distributed across a wide-ranging computer complex.

Data can be partitioned across multiple Adabas database services. When a large file is partitioned across two or more databases, the processing load is actually being spread across the computer service. With more than one CPU engine in your computer, greater use is made of the parallel availability of the CPU engines.

Adabas Vista partitions are truly independent Adabas files:

- partitions need not be identical. Provided all the partitions support all of the views to be used by Adabas Vista, the files can operate with different physical layouts (FDTs). Of course, the Adabas source fields that are common to all partitions must be defined identically in each FDT.
- partitions can be maintained individually. You can size, order, and restore according to the needs of the individual partition. You do not have to make all partitions operate from the same physical constraints. You may choose some to be large, others medium, etc. You can also tune the ASSO space according to the partition.

You can select the applications for which Adabas Vista provides a single file image for all the partitions. You can also set an application for mixed access mode so that a program can access a partition directly by its real file number, even while using the single file image.

A file is usually partitioned based upon the overall dominance of a key field such as location or date: the partition criteria. However, it is possible to partition a file without a partition criteria.

Applications generally access the file with search data based to some extent on the key field. Adabas Vista minimizes processing overhead by detecting access explicitly or implicitly based

upon the partition criteria, interrogating the search argument, and directing the access to the specific partition(s) needed. This is referred to as focused access.

The partition outage facility of Adabas Vista allows you to control what happens when a partition becomes unavailable. You can set sensitivity to partition outage unilaterally and allow business application to override it on a user basis. For example, if your partition criteria is location and only data in a particular location is critical to users in that location, you can set partition outage so that users are interrupted by outages in their own location but unaffected by outages in other locations. This can greatly increase the overall availability of your data, which can significantly enhance the effectiveness of your business.

The restricted partition facility of Adabas Vista allows you to hide partitions even though the data is available. You can use this facility to limit data to particular users based on role, location, or other business definition for security or performance reasons.

The consolidation facility of Adabas Vista allows you to impose a single file image upon multiple, previously unrelated files. The files may well be different but they support the same consolidated view.

Adabas Vista can be used on z/OS with all supported versions of Adabas.

Adabas Vista supports Adabas calls from 3GL programs as well as from Natural. An online services option is available in a Natural environment.

Adabas Vista comprises a stub (client) part and a server part. By design, most processing occurs in the client process rather than the server to

- minimize CPU usage;
- minimize the impact of overhead associated with partitioning on the database service; and
- spread the load among as many CPU engines in parallel or even computers as possible.

For more information, refer to the *Adabas Vista* documentation.

Event Replicator for Adabas

The Event Replicator for Adabas is composed of a family of Software GmbH products. The basic product can be used to monitor data modifications in an Adabas database and replicate the modified data to another application. For more information, read the following topics:

- Event Replication Overview
- Event Replicator Target Adapter

For complete information about the Event Replicator for Adabas, read the *Event Replicator for Adabas* documentation.

Event Replication Overview

Software GmbH's Event Replicator for Adabas allows specific Adabas files to be monitored for data modifications. Whenever any record modification (delete, store, or update) occurs in one of the monitored files, the Event Replicator extracts each modified record and delivers it to one or more target applications through a messaging system (such as IBM EntireX or IBM MQSeries). The set of replicated files are defined in one or more subscriptions.

Note: The term *MQSeries* is used in this documentation when referring to the product now known as *WebSphere MQ*.

The Event Replicator is an essential tool for organizations that need Adabas data modifications delivered to a target application while minimally impacting the normal processing of Adabas. The principle features of the Event Replicator include:

- Near real-time replication
- Asynchronous replication
- Guaranteed consistency and sequence of the delivered replicated data
- Replication of committed updates only

With the Event Replicator, whole Adabas files or a specific set of records can be replicated to the target location, as defined in one or more subscriptions. Data replication is asynchronous, which allows the Adabas database to operate normally while replication takes place. Only committed Adabas modifications are replicated for the predefined set of replicated files, at the transaction level.

For complete information about the Event Replicator for Adabas, read the *Event Replicator for Adabas* documentation.

Event Replicator Target Adapter

The Event Replicator Target Adapter can be used to transform and apply replicated data to a relational database, such as Oracle, DB2, Microsoft SQL Server, MySQL, or Sybase.

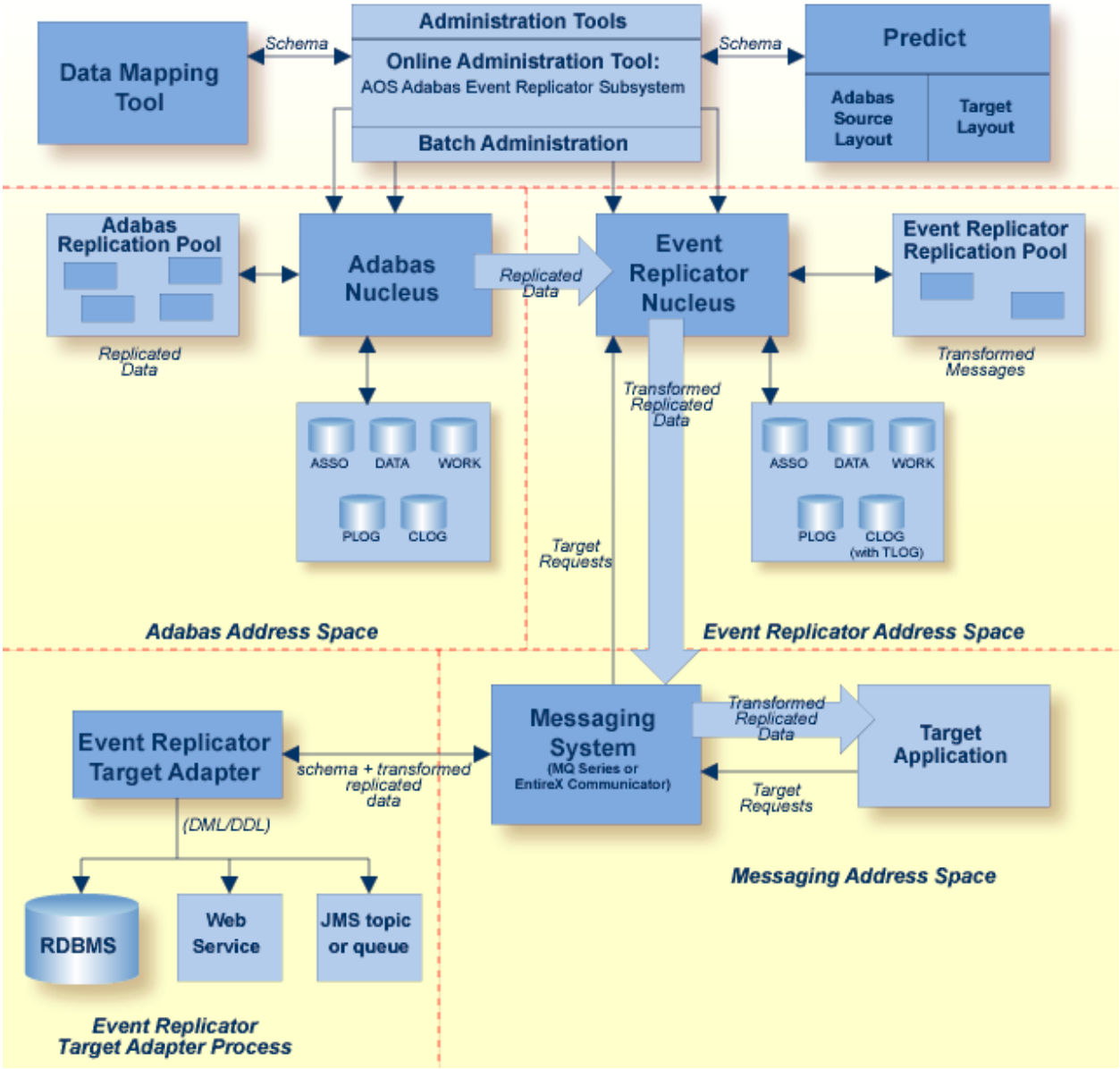
The Event Replicator Target Adapter requires the use of:

- An Event Replicator for Adabas subscription for which a global format buffer field table has been generated. If no field table has been generated, the Event Replicator Target Adapter will not work.
- At least one Event Replicator for Adabas destination definition with a class type of "SAGTARG". This destination must be used by the subscription.

When a subscription definition and one or more of its associated destination definitions have been defined in this manner and if they are activated, the Event Replicator for Adabas automatically creates a schema that maps the replicated data. The Event Replicator Target Adapter uses the

schema to transform and apply the replicated data to your relational database. The Event Replicator Target Adapter will dynamically create tables if they don't exist and populate the tables with Adabas data using insert, update, and delete processing as these processes occur in near real-time in the replicated Adabas file.

Event Replicator for Adabas and Event Replicator Target Adapter high-level processing are depicted in the following diagram.



Once appropriate Event Replicator subscription and destination definitions are activated and the Event Replicator Target Adapter is started, normal processing for the Event Replicator Target Adapter will transform and apply replicated data to a relational database as the data is processed

by the subscription. In addition, you can manually submit requests to the Event Replicator Target Adapter that:

- initiate an initial-state request to populate the relational database tables
- clear data in the relational database tables
- delete relational database tables and their data.

The Event Replicator Target Adapter includes the following component products, in addition to its own code that allows you to transform and apply replicated data to a relational database.

- The Administration tool.

  You can use the Administration tool to configure your Event Replicator Target Adapter installation.

- The Data Mapping Tool.

  You can use the Data Mapping Tool to generate a global format buffer (GFB) and field table (GFFT) for use with the Event Replicator and the Event Replicator Target Adapter.

- The Adabas Administration Services.

  Adabas Administration Services (AAS) provides remote administration and system service support for Software GmbH products. Specifically, Adabas Administration Services allows you to perform the following remote administration tasks for Software GmbH products with which it is supported:

  - You can start and stop the product remotely.
  - You can obtain product log files remotely.
  - You can read and write product files remotely.
  - You can read directories remotely.

  In addition, Adabas Administration Services allows you to operate the Software GmbH product as a system service.

  Ordinarily, Adabas Administration Services starts up automatically after it is installed or whenever the machine on which it is installed is started. In UNIX environments, however, you may occasionally need to start it manually.

  For complete information on the use and maintenance of Adabas Administration Services, read the *Adabas Administration Services Installation and Administration Guide*.

For complete information about the Event Replicator Target Adapter, read the *Event Replicator for Adabas* documentation.

Entire Net-Work Multisystem Processing Tool

Entire Net-Work, Software GmbH's multisystem processing tool, provides the benefits of distributed processing by allowing you to communicate with Adabas and other service tasks on a network-wide scope. This flexibility allows you to

- run Adabas database applications on networked systems without regard to the database location;
- operate a distributed Adabas database with components located on various network nodes;
- perform specific types of tasks on the network nodes most suitable for performing those services without limiting access to those services from other network systems;
- access Entire System Server (formerly Natural Process) to perform operating system-oriented functions on remote systems;
- access Entire Broker in order to implement your client/server applications.

Mainframe Entire Net-Work supports z/OS and provides transparent connectivity between client and server programs running on different physical or virtual machines, with potentially different operating systems and hardware architectures.

Entire Net-Work is additionally available on the midrange platforms such as UNIX platforms, as well as workstation platforms such as Windows and Linux. See Software GmbH's **Empower** web site for more information about platform support.

At its lowest level, Entire Net-Work accepts messages destined for targets or servers on remote systems, and delivers them to the appropriate destination. Replies to these requests are then returned to the originating client application, without any change to the application.

The method of operation and the location and operating characteristics of the servers are fully transparent to the user and the client applications. The servers and applications can be located on any node within the system where Entire Net-Work is installed and communicating. The user's view of the network targets and servers is the same as if they were located on the user's local node. Note that due to possible teleprocessing delays, timing of some transactions may vary.

Entire Net-Work insulates applications from platform-specific syntax requirements and shields the user from underlying network properties. It also provides dynamic reconfiguration and rerouting (in the event of a down line) to effect network path optimization and generate network-level statistics.

Entire Net-Work is installed on each participating host or workstation system requiring client/server capability. The configuration for a given system comprises an Entire Net-Work control module, control module service routines, and any required line driver. Each system with Entire Net-Work installed becomes a node in the network. Each node's adjacent links to other nodes are defined by name and driver type.

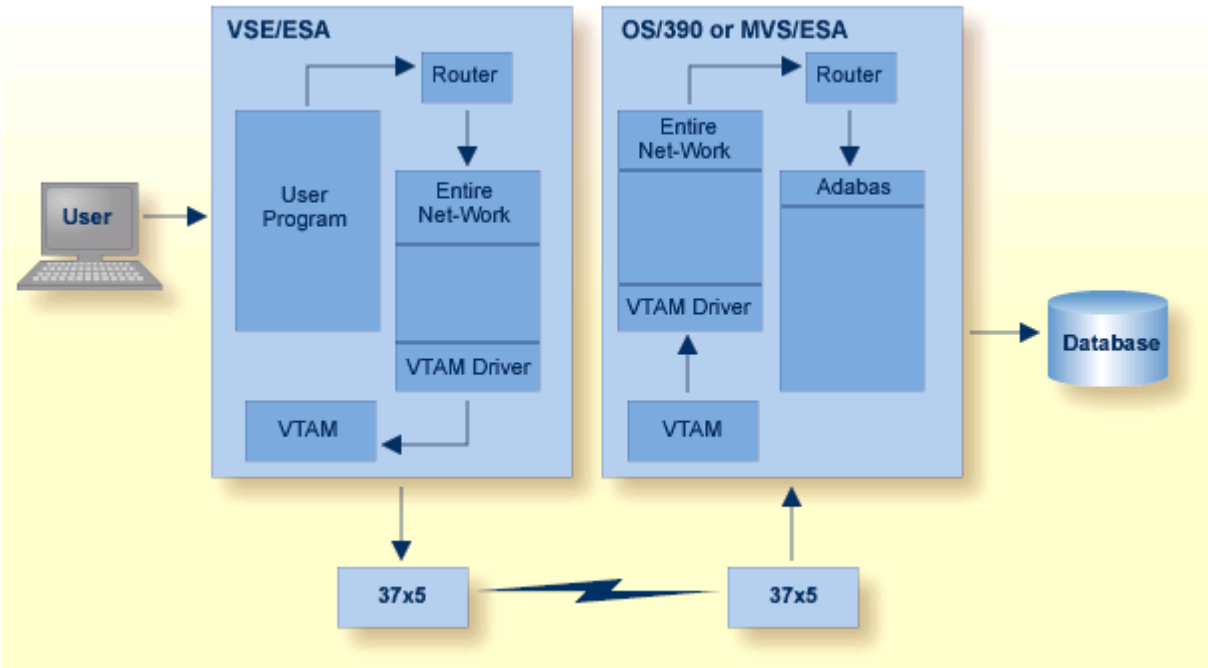
Each Entire Net-Work node maintains a request queue for incoming requests. This queue is similar to the Command Queue used by Adabas; it allows the node to receive Adabas calls from locally executing user/client programs, which Entire Net-Work then dequeues and transports to the nodes where the requested services reside.

Each local Entire Net-Work node also keeps track of all active network services, and therefore can determine whether the user's request can be satisfied or must be rejected. If the request can be serviced, the message is transmitted; otherwise, Entire Net-Work advises the calling user immediately, just as the Adabas router would do for a local database request.

Actual network data traffic is controlled by Entire Net-Work line drivers, which are interfaces to the supported communications access methods, such as VTAM, FCTC, XCF, and TCP/IP, or directly to hardware devices, such as channel-to-channel adapters (CTCAs). Each Entire Net-Work node contains only those line drivers required by the access methods active at that node. In addition, each line driver supports multiple connections to other nodes; this modular line driver design permits easy addition of new access method support to the system.

The Entire Net-Work XTI interface allows users to write their own client/server applications, typically in C, which are independent of the Adabas structures. XTI is an internationally accepted vehicle for creating truly portable applications. In theory, an application created according to XTI specifications can easily be ported to any other platform that supports the XTI implementation.

The Entire Net-Work XTI implementation supports communication between programs running on the same machine and programs running on different machines. Entire Net-Work is viewed as the transport provider from the application programmer's point of view.



For more information, refer to the *Entire Net-Work* documentation.

Natural Application Development Environment

High-level access to Adabas is provided by Natural, Software GmbH's advanced fourth generation application development environment and the cornerstone of Software GmbH's application engineering product family which includes analysis/design, code-generation, and repository facilities.

The access can either be directly from Natural to Adabas or using an Entire Access call via Adabas SQL Server.

Whereas the FDT defines the physical records in an Adabas file, Natural programs define and use logical views of the physical file to access the file. There can be two levels of views: data definition modules and user views.

Data definition modules (DDMs) are Natural modules that look much like the Adabas FDT. They consist of a set of fields and their attributes (type, format, length, etc.), and may contain additional specifications for reporting formats, edit masks, and so on.

A DDM can include all the fields defined in the FDT or a subset of them. There must be at least one DDM for each Adabas file. For example, the Adabas file Employees could have a DDM called Employees. The Natural statement READ EMPLOYEES BY NAME actually refers to the DDM rather than the physical file; the DDM links the Natural statement to the Adabas file.

You can define multiple DDMs for an Adabas file. Multiple DDMs are a way of restricting access to fields in a file. For example, a DDM for a program used by managers could include fields that contain restricted information; these fields would not be included in a DDM for a general-use program. In a workstation group, a database administrator may define a standard set of DDMs for the group.

A new Adabas FDT can be created from an existing Natural DDM. Conversely, Adabas can generate or overwrite a DDM automatically when an FDT is created or changed.

Note: When you delete a field from an Adabas file, you must also eliminate it from Natural programs that reference it.

A Natural user view often contains a subset of the fields in a DDM. User views can be defined in the Data Area Editor or within a program or routine. When a user view references a DDM, the format and length do not need to be defined, since they are already defined in the DDM. Note that in a DDM or user view, you can define the sequence of fields differently from the FDT sequence.

Adabas access is field-oriented: Natural programs access and retrieve only the fields they need. Natural statements invoke Adabas search and retrieval operations automatically.

Adabas supports a variety of sequential and random access methods. Different Natural statements use different Adabas access paths and components; the most efficient method depends on the kind of information you want and the number of records you need to retrieve.

For more information, refer to the *Natural* documentation.

Predict Data Dictionary System

Predict, the Adabas data dictionary system, is used to establish and maintain an online data dictionary. Because it is stored in a standard Adabas file, it can be accessed directly from Natural.

A data dictionary contains information about the definition, structure, and use of data. It does not store the actual data itself, but rather data about data or metadata. Containing all of the definitions of the data, the dictionary becomes the information repository for the data's attributes, characteristics, sources, use, and interrelationships with other data. The dictionary collects the information needed to make the data more useful.

A data dictionary enables the DBA to better manage and control the organization's data resources. Advanced users of data dictionaries find them to be valuable tools in project management and systems design.

Database information may be entered into the dictionary in online or batch mode. The description of the data in the Adabas dictionary includes information about files, the fields defined for each file, and the relationship between files. The description of use includes information about the owners and users of the data in addition to the systems, programs, modules and reports that use the data. Dictionary entries are provided for information about

- network structures
- Adabas databases
- files, fields, and relationships
- owners and users
- systems, programs, modules and reports
- field verification (processing rules)

Standard data dictionary reports may be used to

- display the entire contents of the data dictionary
- print field, file, and relationship information
- print field information by file

For more information, refer to the *Predict* documentation.

Index

A

accept
  security-by-value criterion
    overview, 104
Access methods
  random, 65
access methods
  description, 63
  sequential, 63
ACF2
  Adabas interface to, 105
Adabas
  add-on products, 111
  bridges to VSAM, DL/I, IMS/DB, TOTAL, SESAM, 112
  definition of, 6
  implementing SAF security with, 106
Adabas Bridge for DL/I
  overview, 114
Adabas Bridge for VSAM
  overview, 112
  use of transparency table, 112
Adabas Caching Facility
  online services, 117
  overview, 116
  read-ahead caching, 116
Adabas Cluster Services
  overview, 117
Adabas Delta Save Facility
  online services, 120
  overview, 120
Adabas Direct Access Method (ADAM), 66
  bypassing the inverted lists, 66
  random access retrieval, 66
Adabas Online System
  overview, 123
  requirement for delta save, 120
  security, 109
  use with Adabas Cluster Services, 118
Adabas Parallel Services
  multiple thread processing, 125
  overview, 125
  parallel processing, 125
Adabas Review
  hub (server) component, 126
  interface (client) component, 127
  interface calls, 127
  overview, 125
  sample environment, 127
Adabas SAF Security
  router, 107
Adabas SQL Gateway, 129
Adabas Statistics Facility
  data fields, 130
  online menu system, 130
  report type
    critical, 131
    critical trend, 131
    general evaluation, 131
    trend evaluation, 131
  store profiles, 130
Adabas Transaction Manager, 133
Adabas Vista, 134
ADACDC utility, 84
ADACHK utility
  description, 92
ADACMP utility
  compress/decompress data, 78
ADACNV utility, 85
ADADBS utility
  database functions, 85
  file functions, 86
  other functions, 87
ADADEF utility
  define a database, 89
ADADRU utility
  database resource usage, 93
ADAFRM utility
  format Adabas direct access (DASD) data sets, 89
ADAINV utility
  create a descriptor or couple two files, 90
Adalink
  definition of, 11
ADALOD utility
  load a file into Adabas, 80
ADAM (see Adabas Direct Access Method (ADAM))
  bypassing the inverted lists, 21
  estimation using ADAMER utility, 93
ADAMER utility
  ADAM estimation, 93
ADAORD utility
  reorder databases and files, 90
ADAPLP utility
  print data protection records, 81
ADARAI utility
  database recovery aid, 82
ADAREP utility
  produce database status report, 94
  produce save tape status report, 94

ADARES utility
database recovery and restart, 82
adarsp052, 48
ADARUN parameters
affected by spanned records, 56
ADASAV utility
save/restore database/files, 83
ADASEL utility
select and write data protection log, 84
ADAULD utility
unload an Adabas file, 80
ADAUSER
link with Adabas API, 11
ADAWRK utility
Work part 1 recovery reports, 94
ADAZAP utility
modifying physical database blocks, 91
ADAZIN utility
print database information, 95
address
areas
by operating system, 12
address converter
function of, 23
secondary address converter, 23
alphanumeric fields
no conversion option (NV), 43
API
link applications to Adabas, 11
Associator
component of Adabas, 17, 20
function of, 10
read commands (L9, LF), 61
reorder
using utility, 91
audit routines
ADACHK utility, 92
ADAREP utility, 94
description, 92
autobackout, 73
autorestart, 73
buffer flush check, 74

B

Backout
remove changes between checkpoints, 82
backout, 72
backup routines
ADAPLP utility, 81
ADASEL, 84
description, 81
binary occurrence counter (BOC)
definition of, 46
blocks
reorder
using utility, 91
BOC (see binary occurrence counter (BOC))
bridges
to VSAM, DL/I, IMS/DB, TOTAL, SESAM, 112
buffer flush
database status after, 73
from I/O buffer, 9
session interruption during, 74
buffer manager
augmented by Caching Facility, 116

C

CA-ACF2
use with Adabas SAF Security, 105
CA-Top Secret
using with Adabas SAF Security, 105
cache
Work parts 2 and 3, 116
capturing changed data, 84
Checkpoint
system file, 26
Checkpoints
reapply changes between (ADARES REGENERATE), 83
checkpoints
command to write, 62
CICS
operation with Adabas, 7
ciphering
of critical data, 103
collation descriptor, 50
Com-plete
operation with Adabas, 7
command ID
release
using command, 62
Command log, 25
copy from disk to sequential data set (CLCOPY), 82
merge across a cluster, 82
commands
database modification, 61
database query (Sx, 60
logical transaction control (ET/BT), 61
operator, 14
read (Lx), 61
special housekeeping, 62
types of direct calls, 60
Compression
forward index, 22
control routines
description, 92
converting databases between Adabas versions, 85
coupling two files, 90
creating a descriptor, 90
CTCA
driver with Entire Net-Work, 7

D

Data compression
options
fixed-storage (FI), 42
null-value suppression, 42
data compression
default, 19
options, 19
data definition
CR (insert-only system field) field option, 39
DE (descriptor) field option, 38
DT (date-time edit mask) field option, 40
FI (fixed format) field option, 42
field options

overview, 37
LA (long alphanumeric) field option, 43
LB (large object) field option, 43
MU (multiple-value) field option, 46
NB (no blank compression) field option, 48
NC (null not counted) field option, 48
NN (not null) field option, 48
NU (null value suppression) field option, 42
NV (no conversion) field option, 43
PE (periodic group) field option, 46
SY (system field) field option, 39
TZ (time zone) field option, 41
UQ (unique descriptor) field option, 38
XI (index exclusion) field option, 38
data dictionary
function and use, 142
data protection area
command to write information to, 62
data redundancy
logical, 29
physical, 29
Data Storage
component of Adabas, 17
function of, 10
reorder
using utility, 91
repair blocks, 83
data Storage
read commands (L1-L6), 61
database
accessing from programs, 60
definition of, 16
definition of single physical, 125
maintaining integrity of, 68
modification commands (A1, E1, N1/N2)
overview, 61
query commands (Sx)
overview, 60
repair after nucleus abend, 73
restructure
using utility, 91
single physical
defined, 125
supported models, 8
database information
ADAZIN utility, 95
database modification routines
ADACDC utility, 84
ADACNV utility, 85
ADADBS utility, 85
ADADEF utility, 89
ADAFRM utility, 89
ADAINV utility, 90
ADAORD utility, 90
ADAZAP utility, 91
creating a descriptor or coupling two files, 90
defining a database, 89
description, 84
formatting DASD, 89
database resource usage
ADADRU utility, 93
date-time edit mask (DT) field option
description, 40
deadlock
avoiding resource, 70
defining a database, 89
description (DE) field option
description, 38
descriptor
collation, 50
definition of, 38
hyperdescriptor, 50
phonetic, 51
subdescriptor, 51
superdescriptor, 51
value to order inverted list, 21
direct calls, 10
types of commands, 60
DL/I
bridging files to Adabas, 114

E

elementary
field type, 46
empty field counter
definition of, 19
Entire Net-Work
operation with Adabas, 7
SAF Security Interface
description, 109
supported drivers, 7
using with Adabas Cluster Services, 118
XCF line driver, 119
ET logic, 68
Event Replicator for Adabas, 135
Event Replicator Target Adapter, 136
Execute Channel Program (EXCP)
read and write, 116
expanded files
definition of, 28

F

FDT (see Field Definition Table (FDT))
Field Definition Table (FDT)
definition of, 30
field options
CR (insert-only system field), 39
DE (descriptor), 38
DT (date-time edit mask), 40
FI (fixed format), 42
LA (long alphanumeric), 43
LB (large object), 43
MU (multiple-value), 46
NB (no blank compression), 48
NC (null not counted), 48
NN (not null), 48
NU (null value suppression), 42
NV (no conversion), 43
PE (periodic group), 46
SY (system field), 39
TZ (time zone), 41
UQ (unique descriptor), 38
XI (index exclusion), 38
field type, 46
fields
definition of, 16

elementary, 46
group, 32
levels, 32
multiple value, 46
parent, 49
periodic group, 46
short names, 36
file coupling
logical
definition of, 27
physical
definition of, 27
files
definition of, 16
restructure
using utility, 91
security
access/update level, 104
by password, 104
system, 26
fixed format (FI) field option
description, 42
format Adabas DASD, 89
format ID
command to delete global, 62

G

group
field type, 32

H

hold facility
command to release record hold status, 62
command to set record hold status, 62
housekeeping commands, 62
hyperdescriptor, 50

I

I/O Buffer
algorithm for, 9
purpose of, 9
IMS
bridging IMS/DB files to Adabas, 114
operation with Adabas, 7
inclusive length byte
definition of, 19
index exclusion (XI) field option
description, 38
insert-only system field (CR) field option
description, 39
Intercepts
OPEN or CLOSE, 112
inverted list
associator element, 21
function of, 21
normal index (NI), 21
upper index (UI), 21
ISN sequence, 63
ISNs
definition of, 17
reusing, 17
use in spanned records, 56

L

large object (LB) field option
description, 43
logical sequence, 64
logs
types of, 25
long alphanumeric (LA) field option
description, 43

M

modes of operation
multiuser, 12
single-user, 12
modifying physical database blocks, 91
multi-index searches, 63
multiclient files
definition of, 29
security use, 103
multifile searches, 62
multiple-value (MU) field option
description, 46
multiple-value fields
field type, 46

N

Natural
DDM, 141
use with Adabas, 141
user view, 141
Natural Security, 109
no blank compression (NB) field option
description, 48
no conversion (NV) field option
description, 43
normal index (NI), 21
not null (NN) field option
description, 48
nucleus
cluster
definition of, 125
definition of, 9
number per operating system image, 117
null not counted (NC) field option
description, 48
null suppression (NU) field option
description, 42
null value
SQL
meaning of, 48

O

OpenEdition MVS
support for, 11
operating systems
supported, 117
Operations
highlights, 6
operations

environments supported, 7
overview of Adabas, 8
TP monitors supported, 7
operator commands, 14

P

Padding area
function of, 18
parallel participant table (PPT)
dynamic allocation, 53
initialization processing, 53
overview, 52
parent field
of special descriptor, 49
password
protection
overview, 104
periodic group (PE) field option
description, 46
periodic groups
field type, 46
restrictions on using, 47
phonetic descriptor, 51
physical sequence, 63
PINSAF, 106
PPT (see parallel participant table (PPT))
Predict
using with Adabas, 142
profile
resource/user
description of a, 105
Protection log, 25
copy sequential data set (COPY), 82
copy to sequential data set (PLCOPY), 82

R

RABNs
definition of, 17
fencing, 116
location, 117
RACF
Adabas interface to, 105
using with Adabas SAF Security, 105
random access methods
ADAM, 66
overview, 65
records
definition of, 16
hold and release, 69
resource deadlock, 70
spanned, 54
structure of, 30
recovery, 72
Recovery log, 25
recovery routines
ADARAI utility, 82
ADARES, 82
description, 81
region
address space as a, 12
reject
security-by-value criterion, 104
Remote processing, 118
reordering databases and files, 90
Replicator
system file, 26
reporting
spanned record considerations, 57
reporting routines
description, 92
reports
database audit reports, 94
database information reports, 95
database resource usage reports, 93
database status reports, 94
Work part 1 recovery reports, 94
resources
access/update privileges for, 105
response code 52 (ADARSP052), 48
restart, 72
processing after system failure, 73
restoration routines
ADASAV utility, 83
description, 81
router
Adabas SAF Security, 107

S

SAF Security Interface
Adabas, 106
Entire Net-Work, 109
SAF-based security system
securing Software GmbH products with, 109
searches
complex, 62
multi-index, 63
multifile, 62
secondary address converter, 23
secondary record segmentation, 55
Security
system file, 26
security
options available, 100
package (non-Software AG)
general description, 105
general operation, 105
spanned records, 57
sequential access methods
ISN sequence, 63
logical sequence, 64
overview, 63
physical sequence, 63
session
Adabas, 12, 72
command to
close, 62
open, 62
types of, 12
user, 12, 72
utility, 12
Sort
data set for, 25
space
management, 17
spanned records

ADARUN parameters affected, 56
allowing, 55
description, 54
ISN use, 56
reporting, 57
secondary record segmentation, 55
securing, 57
structure, 55
stored procedure
definition of, 11, 66
subdescriptor, 51
subfield, 51
subscription logging
system file, 26
superdescriptor, 51
superfield, 51
SYSACF application
online cache maintenance, 117
SYSFILE
system file, 26
system field (SY) field option
description, 39
system fields
definition, 32
SY and CR field options, 39

T

TCP/IP
driver with Entire Net-Work, 7
Temp
data set for, 25
threads
multithread processing, 9
size and number, 9
threshold (protection) levels
overview, 104
TIAM
operation with Adabas, 7
time zone (TZ) field option
description, 41
timeout controls
non-activity time limit, 72
transaction time limit, 72
Top Secret
Adabas interface to, 105
TP monitor
overview in Adabas operation, 11
transaction
control commands (ET/BT), 61
definition of, 68
Trigger
system file, 26
trigger
definition of, 11, 66
triggers and stored procedures
overview, 66
TSO
operation with Adabas, 7
tuning routines
ADAMER utility, 93
description, 92

U

unique descriptor (UQ) field option
description, 38
universal encoding support (UES)
no conversion field option (NV), 43
Updating
reapply backed-out update, 83
updating
competitive, 69
exclusive control, 71
upper index (UI), 21
user
exclusive control
updating, 71
isolating within a file, 29, 103
multiuser operating mode, 12
profile, 105
program
relationship to Adabas operation, 10
session, 12
definition of, 72
single-user operating mode, 12
user data
read
using direct call command, 62
User exits
controlling cipher codes with, 103
utilities
overview, 10, 78
session
definition of, 12
UTM
operation with Adabas, 7

V

value
security by
overview, 104
VSAM
bridging files to Adabas, 112
VTAM
driver with Entire Net-Work, 7

W

wide-character fields
no conversion option (NV), 43
Work
component of Adabas, 17
function of, 10, 24
Work part 1 recovery reports
ADAWRK utility, 94

X

XCF
driver with Entire Net-Work, 7
XCF line driver, 119
XCF member
definition, 119