Adabas for Linux, UNIX and Windows

Adabas Security Features

Version 7.0.1

October 2022

ADABAS & NATURAL

This document applies to Adabas for Linux, UNIX and Windows Version 7.0.1 and all subsequent releases.

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

Document ID: ADAOS-SECFAC-701-20220622

Table of Contents

Adabas Security Features

This document describes the security facilities provided by Adabas and its subsystems.

Adabas Role-based Security (ADARBA) provides facilities for controlling access to database resources; for example:

- Restricting the execution of an Adabas utility;
- Restricting access to data via Adabas commands.

Adabas Password Security (ADASCR) provides facilities for controlling access and update to Adabas files; for example:

- Restricting user read and/or update requests on the basis of a whole file;
- Restricting user access to individual records within a file.

Adabas Encryption for Linux prevents the unauthorized analysis of Adabas container files. It provides functionality to encrypt ASSO and DATA container files.

Ciphering prevents the unauthorized analysis of Adabas container files.

SSL Trusted Relationship for Natural enables Natural to make use of the Role Based Access Control without providing user credentials or using the Adabas user exit 21.

The following topics are covered:

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: 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**.

In addition, you can also access the cloud product documentation via **https://www.softwareag.cloud**. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

Product Training

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

Tech Community

You can collaborate with Software AG 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 AG 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 AG resources.

Product Support

Support for Software AG 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 Role-Based Security (ADARBA)

Important: Before securing databases with Adabas Role-based Security, please familiarize yourself with the concepts and implementation of this feature. Once enabled, security cannot be disabled.

Adabas Role-based Security implements Role-Based Access Control (RBAC) and restricts access based on the roles assigned a user and the permissions that are assigned to the role.

With Adabas Role-based Security, you can control what end-users can do at both broad and granular levels. You can designate the roles assigned a user and align the roles and access privileges with your users' tasks. This means a user can be assigned a minimum of privileges, only the privileges essential to perform a specific task.

This feature includes the functionality:

Authentication
This provides a means of validating credentials against an authority.

Authorization for Direct Call Interface
This provides a means of restricting the usage of Adabas commands, which access a file, by assigning users a role which represents selective access privileges.

Authorization for Adabas Utilities
This provides a means of restricting the execution of Adabas utilities by assigning users a role which has the execute privilege.

Audit Trail
This provides a means of tracking access attempts and security violations.

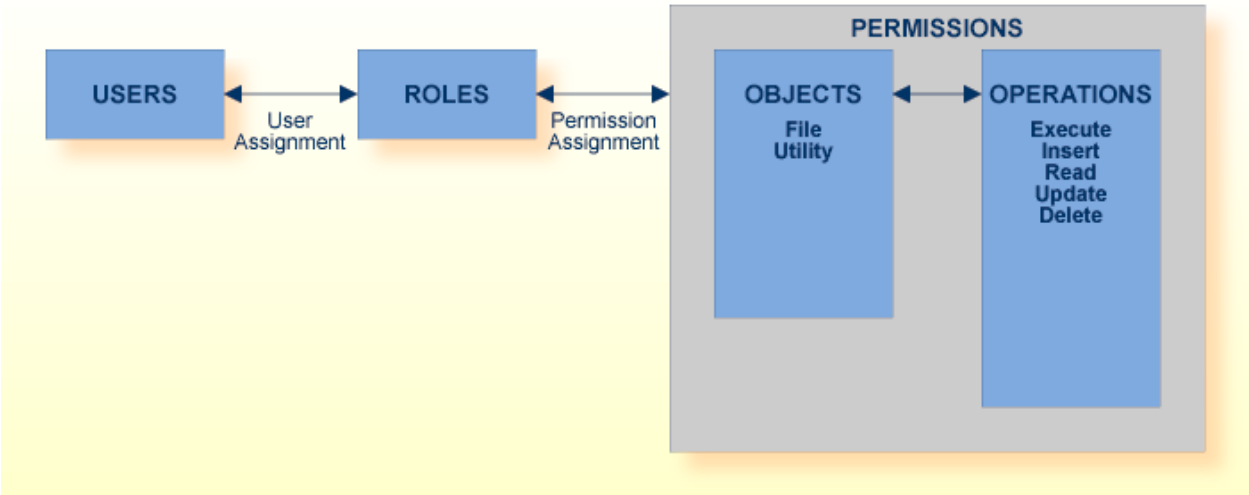
The utility ADARBA provides the functionality required to administer security definitions.

Adabas Role-Based Security Model

The RBAC reference model used by Adabas is based on the Core RBAC (RBAC0) of the ANSI Specification (ANSI INCITS 359-2004).

Core RBAC defines a minimum Role-Based Access Control system. This includes user-role assignment and permission-role assignment relations, considered fundamental in any RBAC system.

In the standard, the core RBAC model includes a set of sessions, where each session is a mapping between a user and an activated subset of roles that are assigned to the user. Sessions are currently not supported and are thus not part of the Adabas role-based security model.



The Adabas RBAC reference model enables the database or security administrator to implement the following:

- The assignment of USERS to ROLES,
- The assignment of ROLES to PERMISSIONS,
- where a PERMISSION consists of a set of OPERATIONS which can be performed on OBJECTS.

A *user* (USER) is defined as a human being. Although the concept of a user can be extended to include applications, the definition is limited to a person here for simplicity reasons.

A *role* (ROLE) is a job function within the context of an organization, with some associated semantics regarding the authority and responsibility conferred on the user assigned to the role.

Permissions (PERMISSIONS) are an approval to perform an operation on one or more RBAC protected objects.

An *operation* (OPERATION) is an action or function, such as the execution of an Adabas utility, Adabas command, etc.

An *object* (OBJECT) is a resource that is subject to access control, such as an Adabas database, file, etc.

Security Definitions

This section describes the location and content of the security definitions.

The security definitions are stored in the RBAC system file of a security-enabled database. The contents of the system file constitute the security repository.

The security repository stores granted privileges, the denial of specific operations is not supported. It is accessible when the database is online or offline.

Initial Security Definitions

The initial security definitions implement unrestricted access to all operations and can be adapted and extended as required.

The unrestricted access to operations is achieved via the **PUBLIC** user and **PUBLIC** role security definitions.

PUBLIC User

The **PUBLIC** user is a pre-defined security definition. It is a convenience definition that is used when the provided credentials cannot be validated. This user has the permissions that can be derived from the assigned roles.

The **PUBLIC** user may be modified as follows:

- The **PUBLIC** user may be dropped;
- The roles assigned to the **PUBLIC** user may be modified or removed as required. Additional roles may be assigned.

By default, the **PUBLIC** user is assigned the **PUBLIC** role.

PUBLIC Role

The **PUBLIC** role is a predefined security definition. All users have the permissions which are assigned to the **PUBLIC** role.

The **PUBLIC** role may be modified as follows:

- The **PUBLIC** role is a system definition and as such *should not be dropped*.
- The permissions assigned to the **PUBLIC** role may be modified or removed as required. Additional permissions may be assigned.

By default, the **PUBLIC** role has “legacy privileges”; for example, the privilege to execute to all Adabas utilities.

Authentication

Authentication provides a means of identifying a user, by having the user provide a valid user name and a valid password before access is granted.

The credentials are checked against an authentication authority, for example against an external authentication system such as LDAP, Active Directory, operating system, or an internal user repository. If the credentials match, the user is provided access to the database. If the credentials are at variance, authentication fails and access to the database is denied.

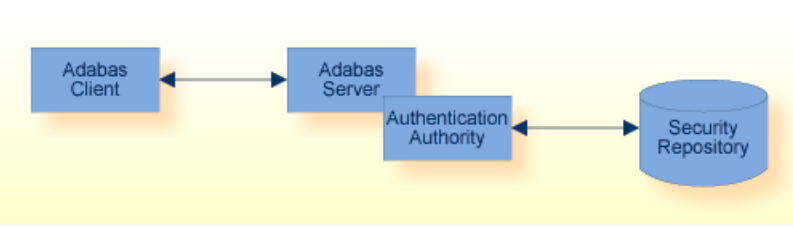
The results of the authentication check are protocolled in the audit trail log.

Please refer to the section *Configuration of the Infrastructure Security Libraries* for further information about the possible authentication authorities.

- Architecture
- Credentials

- Default State

Architecture



Adabas uses the Infrastructure Security Libraries to access the authentication authority for validation of credentials. These libraries enable the authentication of credentials against LDAP, Active Directory, and internal repository or the operating system.

The installation of the Infrastructure Security Libraries is mandatory for usage of this feature.

Please refer to the section on *configuration* for further details on how to configure the authentication feature.

Credentials

The credentials used by Adabas are user identification and password. The user identification consists of the user account name. In the case of Windows, the user identification also contains the domain of the user.

Adabas Direct Call Interface

Credentials can be provided by one of the following means:

- The application provides the credentials;
- The credentials are provided by the nucleus user exit 21.

The application should always provide the credentials to ensure that the application user can be identified. How applications credentials are supplied is described in the section *Developing Applications* of *Application Development*.

The nucleus user exit 21 can be used to transition existing applications to a secure database. Further information on this topic is provided in the section *Modifying Legacy Applications*.

The authentication authority that is to be used is defined in a database configuration file.

Adabas Utilities

Currently, the supported credentials are the local system credentials, which have been authenticated by the operating system.

- Unix: User ID
- Windows: Domain and User ID

Important: The Adabas utilities currently do not perform authentication checks. The local system credentials are used to perform the authorization check. This is subject to change in a future release.

Default State

By default, authentication is not enabled.

The configuration and usage of the authentication feature are described in *Configuration > Authentication* below.

Authorization

Authorization provides the means of restricting operations which are to be performed on a resource or, in other words, the execution of

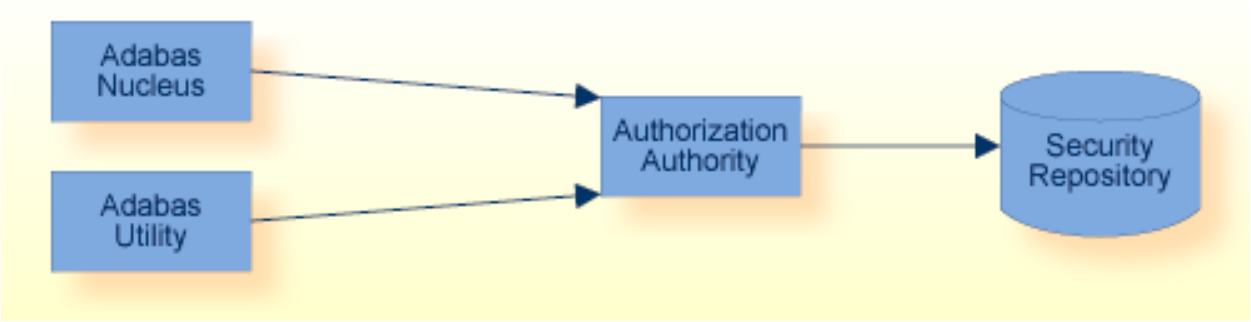
- An Adabas command on a file, or
- An Adabas utility on a database.

An access request can be issued by either the Adabas nucleus or an Adabas utility.

- The Adabas nucleus issues an access request prior to processing an Adabas command.
- An Adabas utility issues an access request prior to executing the requested utility.

The access request is validated by the authorization authority. It is validated against the security repository stored in the database. The credentials used to process the access request were validated previously by the authentication authority.

The results of the authorization check are protocolled in the audit trail log.



Adabas Direct Call Interface

Authorization for Adabas Direct Call Interface allows you to secure specific database files against unauthorized access via Adabas commands.

This feature uses role-based security definitions to restrict and monitor the usage of the Adabas commands, which can be performed on a specific file in the database.

If a matching access privilege is found, the user is authorized to execute the requested operation, otherwise the request is rejected.

Attempts to execute a controlled operation, here an Adabas command, are documented in an audit trail. The configuration of the audit trail log is described in *Configuration > Audit Trail* below.

The following table is a list of controlled access operations on files:

Operation	Description	Commands
READ	Read data stored in file	L1, L2, L3, L4, L5, L6, L9, S1, S2, S4, S8, S9
INSERT	Insert data into the file	N1, N2
UPDATE	Update data that is stored in a file	A1
DELETE	Delete data that is stored in a file	E1
ANY	This is a convenience notation. It includes the READ, INSERT, UPDATE and DELETE operations.	All of the commands above

Note: In the current version, the execution of Adabas commands, which are not listed above, is not restricted. This will change in a future release.

The operation names above are the external representation of the security definitions.

When using the LIST function of the ADARBA utility, the internal representation of the security definition is shown.

Operation	Description	LIST Layout
READ	Read data stored in a file	ada.dml.read
INSERT	Insert data into a file	ada.dml.insert
UPDATE	Update data that is stored in a file	ada.dml.update
DELETE	Delete data that is stored in a file	ada.dml.delete

Adabas Utilities

Authorization for Adabas Utilities allows you to secure a specific database against unauthorized access via Adabas utilities.

This feature uses role-based security definitions to restrict and monitor the usage of the Adabas utilities, which can be performed on a specific database.

If a matching access privilege is found, the user is authorized to execute the requested operation, otherwise the request is rejected. The authorization of a request implies the privilege to execute the Adabas utility.

All attempts to execute an Adabas utility are documented in an audit trail.

The following table is a list of possible access operations on Adabas utilities; all values are case-sensitive:

Operation Name	Utility	Description
ada.uti.bck	ADABCK	Backup and restore database or files
ada.uti.clp	ADACLP	Command log report
ada.uti.cmp	ADACMP	Compression of data
ada.uti.cvt	ADACVT	Convert a database from a previous version
ada.uti.dbm	ADADBM	Database modification
ada.uti.dcu	ADADCU	Decompression of data
ada.uti.dev	ADADEV	Disk space management
ada.uti.ela	ADAELA	Event anaylitics administration
ada.uti.elp	ADAELP	Event log report
ada.uti.err	ADAERR	Error file report
ada.uti.fdu	ADAFDU	File definition
ada.uti.fin	ADAFIN	File information report
ada.uti.inv	ADAINV	Creating, removing and verifying inverted lists
ada.uti.mon	ADAMON	Monitoring the database nucleus

Operation Name	Utility	Description
ada.uti.mup	ADAMUP	Mass add and delete
ada.uti.nuc	ADANUC	Starting the database, defining nucleus parameters
ada.uti.opr	ADAOPR	Operator utility
ada.uti.ord	ADAORD	Reorder the database or export/import files
ada.uti.plp	ADAPLP	Protection log printout
ada.uti.pri	ADAPRI	Print Adabas blocks
ada.uti.rba	ADARBA	RBAC administration
ada.uti.rec	ADAREC	Recovery of database or files
ada.uti.rep	ADAREP	Database report
ada.uti.scr	ADASCR	Manage and enable security functionality
ada.uti.uld	ADAULD	File unloading
ada.uti.vfy	ADAVFY	Database consistency check

Note: The operation names listed above are subject to change.

Default State

By default, authorization is not enabled.

The configuration and usage of the authorization feature are described in *Configuration > Authorization* below.

Audit Trail

An audit trail entry is written to the audit trail log file for each authentication and authorization attempt. The audit trail logs both successful and failed attempts to access the database.

The content of each audit trail entry provides the following information on the attempted access:

- Timestamp of access attempt;
- Information about the application or utility, which attempted access;
- User and session identification of access attempt;
- Information about the attempted operation;
- Information about the object of the operation.

The following requirements apply to the audit trail log file:

- The audit entries are appended to the log file;

- The size of the log file should be monitored and the log file should be backed-up or moved as required.

Authentication

The location and layout of the audit trail log file is described in the section *Authorization for Adabas Direct Call Interface*.

Authorization for Adabas Direct Call Interface

Location

The audit trail log is database-specific and is located in the database directory. The file name is *NUCADT.log*.

Layout of Audit Trail Entry

The layout of the audit trail log file is CSV format. The values of the audit trail entry are separated by comma ",".

The following table contains a description of the audit trail entries for authentication and authorization, the values provided in each audit trail entry are dependent on the type of entry.

Column	Description	Value
Timestamp	Timestamp	
Security Mode	Security Mode	A[ctive] W[arn]
Result	Processing result - access is allowed or not	YES NO
DBID	Database ID	
DBName	Database name	
Session ID	Client information, as specified in the Adabas User Queue	
ET User	ET User	
Security User	User credential provided for authentication	
RBAC User	User credential used to determine access privileges	
RBAC Role	Role with which access privilege was given	

Column	Description	Value
Operation	Adabas command category	INSERT READ UPDATE DELETE
Command	Adabas command code	
File Number	Adabas file number	
File Name	Adabas file name	
Authority	Authentication or Authorization authority	SSX RBAC
Response Code	Adabas response code	
Subcode	Adabas subcode	
SSX Response	SSX response	
SSX Message	SSX message	

Authorization for Adabas Utilities

In this release, the *adaaudit.ini* configuration file defines the layout and location of the audit trail.

Please refer to the section *Location of Configuration and Logging Files* for further information about the location, configuration and content of the audit trail for Authorization for Adabas Utilities.

Important: The location and configuration of the audit trail log file for Authorization for Adabas utilities is deprecated. It is subject to change in a future release.

Layout of Audit Trail Entry.

Column	Description	Prefix	Values
Timestamp	Timestamp		yyyy-mm-ddThh:mm:ssZ
Message Type	Message indicator	%AUTHORIZATION-	I[nformation] E[rror]
Hostname	Host name of machine	HOST=	<hostname>
Opsys Version	Name and version of operating system	OS=	<operating_system>
Credentials	Name of user account	USER=	[<domain>/]<user>
Operation	Name of attempted operation	OPERATION=	<operation>
DBID	Database ID	DBID=	<number>
Result	Result of authorization	AUTHORIZED=	YES NO

Default State

The audit trail is automatically enabled when either authentication or authorization is enabled.

Please refer to the section *Location of Configuration and Logging Files* for further information on the configuration of the contents of the audit trail log file.

The configuration and usage of the authorization feature are described below.

Configuration

This section describes the configuration of the Adabas Role-based Security feature.

Initial Security Configuration

An initial security configuration is created during installation.

- Security (MODE ADABAS) is enabled in the security configuration files;
- Authentication is not enabled;
- Authorization is not enabled;
- Audit Trail is not enabled.

Security Configuration Files

The following settings in the security configuration file *adaauth.ini* are mandatory:

- ACTION=YES
- MODE=ADABAS

Please refer to the section *Configuration of Adabas Role-Based Security* for further information on the file location and content.

Note: The security configuration files will be deprecated in a future release.

Authentication

Important: Database security cannot be disabled once it has been activated.

Enable Authentication

Use the SECURITY function of the ADADBM utility to enable the authentication. The security mode can either be set to WARN or ACTIVE.

The security mode ACTIVE implies that only authenticated users are allowed access to the database. Security violations, such as authentication errors, are protocolled as “Error” in the audit trail.

The security mode WARN is intended for transitioning applications to use a secure database. It implies that all users are allowed access to the database. Security violations, like authentication or authorization errors, are protocolled as “Warning” in the audit trail. In case of a security violation, access to the operation is not rejected.

Once enabled, the security mode can only be changed from mode WARN to ACTIVE.

When you introduce this feature, Software AG strongly recommends that you initially start with security mode WARN.

Please refer to the section *Nucleus user exit 21* for detailed information on how to provide authentication credentials for legacy applications.

Configure Authentication Authority

The following types of authentication authority are supported:

- Authentication Type OS (Operating System)
- Authentication Type TEXT (Internal User Repository)
- Authentication Type LDAP
- Authentication Type ADSI

These settings are described in the section *Configuration of the Infrastructure Security Library*, where you can find example templates for the different authentication types. These templates are not complete as some of the settings are customer-specific and must be modified where necessary.

These configuration settings are stored in the section [SSX_CONFIGURATION] of the *DBnnn.INI* file. Use the administration command ADAINI to set and modify these settings.

Authorization

Enable Authorization

Use the RBAC_FILE function of the ADADBM utility to load the RBAC system file.

Define Security Definitions

Use the ADARBA utility to manage customer-specific security definitions.

Please refer to the examples in section *Getting Started* for further information.

Note: The RBAC System File is closely related to the database. It has to be backed up and restored together with the complete database and cannot be restored individually.

Audit Trail

Authentication and Authorization for Direct Call Interface

The audit trail log is written to the file *NUCADT.log,* which is located in the database directory.

The configuration settings are stored in the section [AUDIT_TRAIL] of the *DBnnn.INI* file. Use the administration command ADAINI to set and modify these settings.

The following AUDIT_TRAIL settings can be used to configure the audit trail:

ACTION
Enable or disable Audit Trail logging for authentication and authorization.

FILTER
Log ALL audit trail entries or just the entries for REJECTED authentications and authorizations.

Please refer to the section *Configuration Files* for further information on AUDIT_TRAIL settings .

Authorization for Utilities

The configuration settings for the audit trail log file are stored in the section [AUDIT] of the configuration file *adaaudit.ini*.

The following options are available:

- The item FORMAT defines the layout of an audit log entry.
- The item SEPARATOR defines the character to be used to separate values in CSV format.
- The item LOG_FILE defines the location and file name of the audit log.

Please refer to the section *Configuring Adabas Role-based Security* for further information.

Administration

The utilities required to configure and administer the authorization feature are:

- ADADBM
- ADAINI
- AREP
- ADARBA

ADADBM

- Use the SECURITY function to enable Adabas RBAC Security;
- Use the RBAC_FILE function to create the RBAC system file.

Note: These functions require that the database is offline.

ADAINI

- Use the administration command ADAINI to set and modify the configuration of security features;
- Configure the authentication authority;
- Configure the audit trail;
- Display the configuration settings.

ADAREP

- Use the SUMMARY function to display the database system files and whether security is enabled;
- The SECURITY setting is displayed if security is enabled;
- The RBAC system file information is displayed if the RBAC system file is defined.

ADARBA

- Use the CREATE, DROP, GRANT, REVOKE functions to create, modify and delete security definitions;
- Use the LIST functions to display the security definitions.

Note: These functions require that the database is online.

Performance Considerations

Authentication

This feature could have a detrimental effect on the overall system performance for the following reasons:

- Due to the effort required to authenticate the credentials;
- Due to the large number of entries being written to the audit trail log file.

Authorization for Direct Call Interface

This feature could have a detrimental effect on the overall system performance for the following reasons:

- Due to the effort required to authenticate the credentials;
- Due to the effort required to determine the user's access privileges;
- Due to the large number of entries being written to the audit trail log file.

Authorization for Adabas Utilities

This feature only has minimal impact on the overall system performance.

Audit Trail

Due to the high number of synchronized file I/Os, this feature could have a detrimental influence on the overall system performance.

The impact can be minimized by using the AUDIT_TRAIL parameter setting FILTER=REJECTED for authentication and authorization for Direct Call Interface requests.

The following configuration options have a detrimental influence on performance, and should be used with care:

Feature	Option	Explanation
Audit Trail	Filter	A large number of user sessions or Adabas commands will result in a large number of security entries being written to the audit trail log file. The size of the audit trail log file increases rapidly with the numbers of user sessions or Adabas commands. Default value: FILTER = ALL Recommended value: FILTER = REJECTED

Feature	Option	Explanation
SSX Logging/Diagnostics	nativeloglevel	Multiple user sessions attempting to write diagnostic information concurrently to the security infrastructure log file. Default value: None Recommended value: 0 or None

Application Development

This section covers the following topics:

- Developing Applications
- Modifying Legacy Applications
- Error Handling
- Messages and Codes

Developing Applications

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

The following Adabas client functions are provided to manage client sessions and set credentials:

Step	Function	Description
1	lnk_set_adabas_id()	Set the session identification.
2	lnk_set_uid_pw()	Set the authentication credentials for a specific database.

Details about the above Adabas client functions can be found in the section *Calling Adabas with Authentication* in the section *Calling Adabas* in the *Command Reference*.

It is recommended to use OPTIONS=OPEN_REQUIRED.

Modifying Legacy Applications

Without modification, legacy applications will receive the nucleus response 200 “security violation”, when accessing secured databases.

The Adabas nucleus user exit 21 can be used to set authentication credentials via the Adabas server API functions. The routine is called when the processing of a session begins.

This routine should be used as briefly as possible. It is intended for use during the transition period, until all applications use and support the Adabas security authentication feature.

For further details, see the nucleus user exit 21 in the section *User Exits and Hyperexits*.

Error Handling

Security Mode Active

If a security violation occurs during authentication processing, Adabas issues a response code, backs out the user's current transaction, and closes the user session.

- Response code 9, subcode SE
- Response code 200, subcode 31

If a security violation occurs during authorization processing, Adabas issues a response code,

- Response code 200, subcode 175

Security Mode WARN

The error handling for security mode WARN is intended for transitioning to a secure database. The error handling is described below. It differs to that of mode ACTIVE.

- The current transaction is not backed out;
- The response code returned indicates “success” (response 0);
- A “Warning” entry is entered in the audit trail log file.

Messages and Codes

The following Adabas nucleus response codes indicate that a security issue has occurred:

Authentication

Response Code	Explanation	Action
9/SE	The credentials have been modified since the start of the session. Hence the credentials are invalid, and the user session has been terminated.	Ensure that the credentials are not set or modified during a user session. To proceed processing, set the credentials and open a new user session.
200/31	The credentials could not be validated by the authentication authority.	Ensure that the credentials have been provided and that they are valid.

Authorization

Response Code	Explanation	Action
200/175	The user identified by the provided credentials does not have the privileges to perform the requested operation.	Ensure that the credentials have the required privileges.

Getting Started

This section covers the following topics:

Prerequisites

In order to execute the steps described in the following section, you will require a database.

The following prerequisites are mandatory:

- The Infrastructure Security Libraries are installed.
- The security feature is enabled.

Authentication has the following prerequisites:

- The security mode is enabled.
- The user credentials that are to be validated by the authentication authority.

Authorization has the following prerequisites:

- The RBAC system file is loaded.
- Security definitions will be required.

You, or an administrator, will require the necessary access privileges:

- To modify the security configuration files.
- To create and configure a database.
- To start and stop a database.

Important: When starting with the Adabas RBAC security feature, it is recommended to create a database and to initially use the SECURITY mode WARN. Once the Adabas role-based security feature has been enabled, it cannot be disabled.

Configuration

This section describes how to configure Adabas role-based security (ADARBA).

Security Configuration Files

1. Edit the security configuration file *adaauth.ini*.
2. To enable Adabas role-based security, set the following in the section [AUTHZ]:
   - ACTION=YES
   - MODE=ADABAS
3. By default, the following settings do not require modification:
   - AUDIT_FILE - location of the Audit configuration file.
   - RBAC_FILE - this setting has been deprecated.

The modification of the Audit configuration file settings is described in the section *Audit Trail*.

Authentication

The following steps describe how to enable authentication.

1. Ensure the database is offline.
2. Configure the authentication authority for the database.
3. Enable authentication by setting the SECURITY mode.
4. Start the database.

Note: Authentication is not required and does not have to be enabled for Authorization for Adabas Utilities.

The authentication examples below use the internal user repository as the authentication authority.

Example: Create Credential in an Internal User Repository

This example shows how to create a user, password credential in an internal user repository.

```
> ssxtxtpasswd -f SAGInternalUserRepository.txt -c -p mypsw myuid

Hash: ↩
bOEOAPEEEJBKv+4zOELiYcFqY7qFhlLZz1ha7Ztf7j/drJHGy2MLOLXEu/kX7TD52Aj7XfwiZ+vpIl9DqRbVkA==
User entry for "myuid" successfully added
```

The contents of the file *SAGInternalUserRepository.txt* are shown below:

```
*
*
* SAG Internal User Repository
*
version:3.0
*
user:myuid:$6a$bOEOAPEEEJBKv+4zOELiYcFqY7qFhlLZz1ha7Ztf7j/drJHGy2MLOLXEu/kX7TD52Aj7XfwiZ+vpIl9DqRbVk
```

Please refer to the section *Configuring the Infrastructure Security Libraries* for further information about how to create and administer an internal user repository.

Example: Configure the Authentication Authority

This example shows the configuration of authentication with a text file. This file contains authenticated credentials and passwords. Please refer to the section describing the ***Configuration*** for further information on the creation and administration of an authentication text file.

```
DBnnn.INI

> adaini dbid=nnn add topic=DB_PARAMETER topic=SSX_CONFIGURATION item=authType=TEXT
> adaini dbid=nnn add topic=DB_PARAMETER topic=SSX_CONFIGURATION ↩
item=internalRepository=SAGInternalUserRepository.txt
```

Please refer the section *Validating the Configuration* for an example on how to display the SSX_CONFIGURATION settings.

Example: Enable Security Mode

This example shows how to enable the security mode, which enables authentication. When starting with the Adabas RBAC security feature, it is recommended to initially use the SECURITY mode WARN.

Use the SECURITY function of the ADADBM utility to activate the authentication feature.

```
> adadbm dbid=nnn security=warn
%ADADBM-I-STARTED,      15-AUG-2018 11:13:39, Version 6.7.0.0
%ADADBM-I-DBOFF, database 224 accessed offline
%ADADBM-I-FUNC, function SECURITY executed
```

Authorization

The following steps describe how to enable authorization.

1. Ensure the database is offline.
2. Load the RBAC system file.
3. Start the database.

Example: Load the RBAC System File

Use the RBAC_FILE function of the ADADBM utility to load the RBAC system file.

```
> adadbm dbid=nnn rbac_file=250
%ADADBM-I-STARTED,      15-AUG-2018 15:26:58, Version 6.7.0.0
%ADADBM-I-DBOFF, database 224 accessed offline
%ADABCK-I-STARTED,      15-AUG-2018 15:26:58, Version 6.7.0.0
%ADABCK-I-DBOFF, database 224 accessed offline

%ADABCK-I-BKREAL, new RABN allocations for file 7
Restore files dumped on 27-JUL-2018 09:00:19

Database 1, RBAC-FILE

File 7 renumbered to file 250
File   250, RBAC-FILE       , loaded on 27-JUL-2018 09:00:14


%ADABCK-I-IOCNT, 52 IOs on dataset DATA
%ADABCK-I-IOCNT, 228 IOs on dataset ASSO
%ADABCK-I-IOCNT, 228 IOs on dataset BCK001
%ADABCK-I-TERMINATED,   15-AUG-2018 15:27:19, elapsed time: 00:00:21
%ADADBM-I-ACLLOAD, RBAC-FILE loaded in file 250

%ADADBM-I-IOCNT, 4 IOs on dataset WORK
%ADADBM-I-IOCNT, 13 IOs on dataset ASSO
%ADADBM-I-TERMINATED,   15-AUG-2018 15:27:20, elapsed time: 00:00:22
```

Please refer the section *Validating the Configuration* for an example of how to display the RBAC system file information.

Audit Trail

Authentication and Authorization Adabas Direct Call Interface

The audit trail log file for authentication and authorization for Adabas direct call interface is located in the database directory. The file name is *NUCADT.LOG*.

The following steps describe how to enable the audit trail for authentication and authorization for Adabas direct call interface.

1. Configure the audit trail for authentication and authorization for Adabas direct call interface.
2. Restart the database, so that the modifications are active.

The following example shows the configuration of the audit trail for authentication and authorization for Adabas direct call interface.

```
DBnnn.INI

> adaini dbid=nnn add topic=DB_PARAMETER topic=AUDIT_TRAIL item=FILTER=ALL
> adaini dbid=nnn add topic=DB_PARAMETER topic=AUDIT_TRAIL item=ACTION=YES
```

Please refer the section *Validating the Configuration* for an example of how to display the audit trail settings.

Authorization Adabas Utilities

The following steps describe how to enable the audit trail for authorization for Adabas utilities.

1. Edit the security configuration file *adaaudit.ini*.
2. To enable the audit trail, set the following in the section [AUDIT]
   - FORMAT = <file layout>
   - SEPARATOR = <separator_character>
   - LOG_FILE = <log file name>

The modification of the audit trail configuration file settings is described in the section *Configuration for Authorization for Adabas Utilities*.

Validating the Configuration

The following examples show how you can validate the configuration of authentication and authorization.

Example: Database Report - Security Configuration

Use the SUMMARY function of the ADAREP utility to display the SECURITY mode setting and the RBAC system file information.

When enabled, the SECURITY mode will be displayed with a value of either WARN or ACTIVE. When not enabled, the SECURITY mode is not displayed.

When loaded, the file number of RBAC system file will be shown.

```
> adarep dbid=nnn summary

%ADAREP-I-STARTED,      15-AUG-2018 15:37:23, Version 6.7.0.0

Summary of Database nnn          15-AUG-2018 15:37:23


DATABASE NAME                    EXAMPLE-DB
DATABASE ID                        224
MAXIMUM FILE NUMBER LOADED         255
SYSTEM FILES                       255 (CHK),   251 (SEC),   254 (USR)
                                   250 (RBAC)
ACTUAL FILES LOADED                  8
CURRENT PLOG NUMBER                  4
CURRENT CLOG NUMBER                  0
SECURITY                           WARN


Container  Device      Extents in Blocks    Number of      Block    Total Size
  File     Type          from          to     Blocks        Size     (Megabytes)
-------------------------------------------------------------------------------

 ASSO1     file             1       2,000        2,000      2,048           3.91
 ASSO2     file         2,001      10,000        8,000      2,048          15.63

 DATA1     file             1       2,000        2,000      4,096           7.81
 DATA2     file         2,001      10,000        8,000      4,096          31.25

 WORK1     file             1       2,000        2,000      4,096           7.81

-------------------------------------------------------------------------------
                                                                           66.41
                                                                           =====


%ADAREP-I-IOCNT, 17 IOs on dataset ASSO
%ADAREP-I-TERMINATED,   15-AUG-2018 15:37:23, elapsed time: 00:00:00
```

Example: Display Authentication Configuration

Use the SHOW function of the administration command ADAINI to display the authentication configuration.

Show the configuration of the authentication authority.

```
> adaini dbid=nnn show topic=DB_PARAMETER topic=SSX_CONFIGURATION

authType=TEXT
internalRepository=path_and_name_ssxuser_file
```

Display Audit Trail Configuration

Use the SHOW function of the administration command ADAINI to display the audit trail configuration.

Show the audit trail configuration for authentication and authorization for Adabas direct call interface.

```
> adaini dbid=nnn show topic=DB_PARAMETER topic=AUDIT_TRAIL

ACTION=YES
FILTER=ALL
```

Example: Security Infrastructure Initialized

The security infrastructure is initialized during the start of the ADANUC utility. This infrastructure is used for authentication and indicates that the authentication feature is enabled and that the SSX_CONFIGURATION parameters have been processed.

```
> adanuc dbid=nnn

%ADANUC-I-STARTED,      15-AUG-2018 16:21:59, Version 6.7.0.0

%ADANUC-I-SSXINI, SSX Security Infrastructure initialized
%ADANUC-I-CREATED, dataset NUCTMP1, file D:\ ADADATADIR\dbnnn\NUCTMP1.nnn created
%ADANUC-I-PLOGCRE, plog NUCPLG, file 'D:\ADADATADIR\dbnnn\NUCPLG.0001' created
%ADANUC-I-DBSTART, Database nnn, session 5 started, 15-AUG-2018 16:22:02
```

Example: Display Security Definitions

The examples below show the initial values of the security definitions stored in the security repository. Start the database and display the security definitions.

```
> adarba dbid=nnn list,user
```

This shows the user *PUBLIC*.

```
> adarba dbid=nnn list,role
```

This shows the role *PUBLIC*.

```
> adarba dbid=nnn list,assignment,user
```

This shows the assignment *PUBLIC,PUBLIC*. That is the role *PUBLIC* has been assigned to the user *PUBLIC*.

Authentication

The usage of authentication requires that the credentials be set by the application, or be provided by the Adabas nucleus user exit 21. Please refer the section ***Application Development*** for further information on the setting credentials.

Example: Use ADATST to validate Credentials

The example below show how ADATST can be used to verify the authentication of credentials.

```
> adatst

%ADATST-I-STARTED,      16-AUG-2018 12:11:22, Version 6.7.0.0

> adatst: ;-------- Database ID
> adatst: dbid=<dbid>

> adatst: ;-------- Credentials (without trailing whitespace)
> adatst: sec_uid=<userid>
> adatst: sec_pwd=<password>

> adatst: ;-------- Session Open
> adatst: cc=op
> adatst: rb=.
> adatst: go

  Command : OP     Test-Nr : 1 Started
%ADATST-I-NORMAL, normal successful completion
  Command : OP     Test-Nr : 1 Completed

> adatst: ;-------- Session Close
> adatst: cc=cl
> adatst: go

  Command : CL     Test-Nr : 2 Started
%ADATST-I-NORMAL, normal successful completion
```

```
  Command : CL     Test-Nr : 2 Completed

> adatst: quit

%ADATST-I-TERMINATED,   16-AUG-2018 12:11:22, elapsed time: 00:00:00
```

A nucleus response code 200 with subcode 31 indicates a security violation; the authentication check in the external security system failed.

Possible causes of a security violation are:

- The credentials are invalid.
- The configuration of the authentication authority is invalid.
- The authentication authority is not accessible.

A nucleus response code 9 with subcode SE indicates that the credentials have been modified; for example by altering the credentials during the open session.

Example: Use ADATST to validate the usage of Nucleus user exit 21

The example below shows how ADATST can be used to verify that valid credentials are provided by nucleus user exit 21. The user exit is called to provide credentials if the calling application has not provided credentials.

```
> adatst

%ADATST-I-STARTED,      16-AUG-2018 12:11:22, Version 6.7.0.0

> adatst: ;-------- Database ID
> adatst: dbid=<dbid>

> adatst: ;-------- Session Open
> adatst: cc=op
> adatst: rb=.
> adatst: go

  Command : OP     Test-Nr : 1 Started
%ADATST-I-NORMAL, normal successful completion
  Command : OP     Test-Nr : 1 Completed

> adatst: ;-------- Session Close
> adatst: cc=cl
> adatst: go

  Command : CL     Test-Nr : 2 Started
%ADATST-I-NORMAL, normal successful completion
  Command : CL     Test-Nr : 2 Completed

> adatst: quit
```

```
%ADATST-I-TERMINATED,   16-AUG-2018 12:11:22, elapsed time: 00:00:00
```

A nucleus response code 200 with subcode 31 indicates a security violation; the authentication check in the external security system failed.

Possible causes of a security violation are:

- Nucleus user exit 21 is not enabled.
- The provided credentials are invalid.
- The configuration of the authentication authority is invalid.
- The authentication authority is not accessible.

Authorization for Direct Call Interface

This section describes how to secure access to application data, for example allowing all users READ access to the EMPLOYEES file, and at the same time restricting the privilege to modify data.

In this example, you will create and modify the security definitions. In the end, only users assigned the HR_department role are able to modify data in the EMPLOYEES file, but all users are allowed to READ the file.

To achieve this, you need to:

- Create a HR_department role;
- Grant the role permission to modify the EMPLOYEES file;
- Create an HR_userid user;
- Grant the user the HR_department role;
- Grant the *PUBLIC* role permission to READ the EMPLOYEES file.

Example: Secure access to the EMPLOYEES File

The examples below show the security definitions required to restrict access to the EMPLOYEES file.

```
> adarba
> adarba: dbid=nnn

> adarba: create,role=HR_department
> adarba: grant,operation=ANY,object=11,to,role=HR_department

> adarba: create,user=HR_userid
> adarba: grant,role=HR_department,to,user=HR_userid
```

```
> adarba: grant,operation=READ,object=11,to,role=PUBLIC
```

The following security definitions are created or modified:

- Created:
  - Role: *HR_department*
  - User: *HR_userid*
  - Permission to Operation and Object: *ANY, File 11 (EMPLOYEES-NAT)*
- Modified:
  - Role: *PUBLIC*
  - Permission to Operation and Object: *READ, File 11 (EMPLOYEES-NAT)*

Note: The role name and the user name are site-specific.

The user name used for authorization is the same as the one used for authentication.

The user name must be defined for both authorization and authentication.

List Security Definitions

The example below shows the security definitions which secure access to the EMPLOYEES file.

```
> adarba
> adarba: dbid=nnn


> adarba: ; ---------- USERS
> adarba: list,user
PUBLIC
HR_userid


> adarba: ; ---------- ROLES
> adarba: list,role
PUBLIC
HR_department


> adarba: ; ---------- USER-ROLE ASSIGNMENTS
> adarba: list,assignment,user
PUBLIC,PUBLIC
HR_department,HR_userid


> adarba: ; ---------- PERMISSIONS
> adarba: list,assignment,permission
```

```
ada.uti....,DBID.CURRENT,PUBLIC
ada.uti....,DBID.CURRENT,PUBLIC
ada.uti....,DBID.CURRENT,PUBLIC

ada.dml.delete,FILE.00000011,HR_department
ada.dml.insert,FILE.00000011,HR_department
ada.dml.read,FILE.00000011,HR_department
ada.dml.update,FILE.00000011,HR_department

ada.dml.read,FILE.00000011,PUBLIC
```

Authorization for Adabas Utilities

This section describes how to secure access to an Adabas utility, for example the ADADBM utility.

In this example, you will create and modify the security definitions. By default, all users with the *PUBLIC* role have access to and can execute the ADADBM utility. In the end, only users assigned the database administrator role are able to execute the utility.

To achieve this, you need to:

- Create a *database administrator* role;
- Grant the role permission to execute the utility;
- Create a user that is to be assigned the *database administrator* role;
- Grant the user the *database administrator* role;
- Revoke the permission to execute the utility from the role *PUBLIC*.

Example: Secure access to the ADADBM utility

```
> adarba
> adarba: dbid=nnn

> adarba: create,role=database_administrator
> adarba: grant,operation=ada.uti.dbm,to,role=database_administrator

> adarba: create,user=dba_userid
> adarba: grant,role=database_administrator,to,user=dba_userid

> adarba: revoke,operation=ada.uti.dbm,from,role=PUBLIC
```

In this example, the following security definitions are created or modified:

- Created:
  - Role: *database_administrator*
  - User credentials: *dba_userid*
  - Permission to operation: *ada.uti.dbm*

- Modified:
  - Role: *PUBLIC*
  - Permission to operation: *ada.uti.dbm*

Note: The role name and the user credentials are site-specific. The user credentials used for authorization are local system credentials.

Example: List Security Definitions

The example below shows the security definitions that secure access to the ADADBM utility.

```
> adarba
> adarba: dbid=nnn


> adarba: ; ---------- USERS
> adarba: list,user
PUBLIC
dba_userid


> adarba: ; ---------- ROLES
> adarba: list,role
PUBLIC
database_administrator


> adarba: ; ---------- USER-ROLE ASSIGNMENTS
> adarba: list,assignment,user
PUBLIC,PUBLIC
database_administrator,dba_userid


> adarba: ; ---------- PERMISSIONS
> adarba: list,assignment,permission
ada.uti....,DBID.CURRENT,PUBLIC
ada.uti....,DBID.CURRENT,PUBLIC
ada.uti....,DBID.CURRENT,PUBLIC

ada.uti.dbm,DBID.CURRENT,database_administrator
```

Infrastructure Security Library

The infrastructure security libraries are used to access the authentication authority. The following sections contain configuration templates for the `SSX_CONFIGURATION` parameters, organized by type.

Authentication Type OS (Operating System)

The security definitions for authentication type OS are managed by the local operating system.

```
[SSX_CONFIGURATION]

# This is a sample properties file for the case
# when authType is OS and the user database is
# the local operating system -
# On Unix Systems it is using PAM authentication
# On Windows a local LogonUser()

# Specifies the authentication type.
# Is Required: Yes
# Valid values: {"OS", "TEXT", "LDAP", "ADSI"}
# Default Value: None

  authType=OS

# Specifies the explicit path of the privileged daemon process.
# Specify this parameter -
# if the sagssxauthd2 executable file is not in the current directory.
# Valid value is the valid path to the sagssxauthd2 module.
# Default Value: None
# Note: UNIX only.

##authDaemonPath

# Specify a default group name here to be returned
# with any of the group results that are returned by the repository manager.
# A valid value is any valid group name.
# Default Value: None
# Optional.

##defaultGroup
```

```
# If this parameter is specified, its value is used at authentication time
# when domain name is not specified by the user.
# If a domain name is specified, the value of this parameter is not used.
# A valid value is any valid domain name.
# Default Value: None
# Optional.

##defaultDomain

# Specifies how to access data.
# Valid values are:
# o true  - Access is under the account of the running process.
# o false - Access is under the impersonated user ID of the logged on user.
# Default Value: FALSE
# Note: Windows only.
# Optional.

##noImpersonation

# Specifies the local machine name (on which the user is authenticated).
# The machine name is added before users and groups;
# for example,machine_name\user.
# Valid values are:
# o true - If set to TRUE (and there is no domain field), you are authenticated ↩
against the local machine only.
# o false - You are authenticated on the domain that you logged on.
# Default Value: FALSE
# Optional.

##unixAddMachineName

# Specifies the log level.
# Is Required: No
# Valid values:
#    0 - No logging
#    Min: 1
#    Max: 6
# Default Value: None

##nativeLogLevel=0

# Specifies the log file.
# Is Required: No
# Valid values:
#    fully qualified file name
# Default Value: None

##nativeLogFile=SAGSSXCLIENTA_SSX.LOG

[SSX_CONFIGURATION-END]
```

Authentication Type TEXT (Internal User Repository)

The security definitions for authentication type TEXT are stored in a text file. The definitions can either be database-specific or be shared by multiple databases.

```
[SSX_CONFIGURATION]

# This is a sample properties file for the case
# when authType is TEXT and the user database is
# an SAG Internal User Repository
# created by the ssxtxtpasswd utility

# Specifies the authentication type.
# Is Required: Yes
# Valid values: {"OS", "TEXT", "LDAP", "ADSI"}
# Default Value: None

  authType=TEXT

# Specifies the internal repository file
# which has been created with ssxtxtpasswd utility
# Is Required: No
# Valid values:
#   fully qualified file name
# Default Value: None

  internalRepository=<fullpath>/<filename>.<ext>

# Specifies the log level.
# Is Required: No
# Valid values:
#    0 - No logging
#    Min: 1
#    Max: 6
# Default Value: None

##nativeLogLevel=0

#  Specifies the log file.
#  Is Required: No
#  Valid values:
#   fully qualified file name
#  No default value

##nativeLogFile=SAGSSXCLIENTA_SSX.LOG

[SSX_CONFIGURATION-END]
```

Authentication Type LDAP

```
[SSX_CONFIGURATION]

# This is a sample properties file for the case
# when authType is LDAP and the user database is OpenLDAP

# Specifies the authentication type.
# Is Required: Yes
# Valid values: {"OS", "TEXT", "LDAP", "ADSI"}
# Default Value: None

  authType=LDAP

# Specifies which server type will be used.
# Is Required: No
# Valid values: {"ActiveDirectory", "SunOneDirectory", "OpenLdap"}
# Default value: "OpenLdap"

  serverType=OpenLDAP

# Property name that denotes a user entry.
# Is Required: No
# Valid values: (attribute name according to LDAP conventions)
# Default Value: None

  userIdField=cn

# Enumeration of LDAP objectclasses that the user entries use in
# the target LDAP server.
# Is Required: No
# Valid values: (Comma separated list of objectclass names,
#  according to LDAP conventions)
# Default value - depending on serverType:
#  OpenLdap:
#  "top,person"
#  SunOneDirectory:
#  "top,person,organizationalperson, inetorgperson"
#  ActiveDirectory:
#  "top,person,organizationalPerson,user"

  personObjClass=inetOrgPerson

# Enumeration of LDAP objectclasses that the group entries use in
# the target LDAP server.
# Is Required: No
# Valid values: (Comma separated list of objectclass names,
#  according to LDAP conventions)
# Default value - depending on serverType:
#    OpenLdap:
#    "top,groupOfUniqueNames"
```

```
#    SunOneDirectory:
#    "top,groupofuniquenames"
#    ActiveDirectory:
#    "top,group"

  groupObjClass=groupOfUniqueNames

# Property name that denotes a group entry.
# Is Required: No
# Valid values: (attribute name according to LDAP conventions)
# Default value: cn

  groupIdField=cn

# Property name of a user entry that points to the group that
# the user is member of.
# Is Required: No
# Valid values: (attribute name according to LDAP conventions)
# Default value:
#  depending on serverType:
#  OpenLdap:
#  "ou"
#  SunOneDirectory:
#  NULL
#  ActiveDirectory:
#  "memberOf"

  personGrpAttr=ou

# Property name of a group entry that points to users (members)
# Is Required: No
# Valid values: (attribute name according to LDAP conventions)
# Default value:
#  depending on serverType:
#  OpenLdap:
#  "uniqueMember"
#  SunOneDirectory:
#  "uniqueMember"
#  ActiveDirectory:
#  "member"

  groupPrsAttr=uniqueMember

# Seconds how long auth. user remains in cache.
# Is Required: No
# Valid values:
#  0 - No cache
#  Min: 1, Max: No limit
# Default value: 180

  cacheTime=12
```

```
# Specify the max. number of cached users that have been successfully
# authenticated. When the cache overflows, the oldest entry is removed.
# Is Required: No
# Valid values:
#  0 - No cache
#  Min: 1, Max: No limit
# Default value: 300

  cacheSize=4

# Time (in seconds) how long to ignore any further authentication
# requests for a particular User-Id.
# Is Required: No
# Valid values:
#  Min: 1, Max: No limit
# Default value: 100

  denyTime=4

# Number of invalid logon attempts.
# Is Required: No
# Valid values:
#  Min: 1, Max: No limit
# Default value: 3

  denyCount=3

# Specifies an output file for logging.
# Is Required: No
# Valid values: (Valid log file path)
# Default Value: None

  logCallback=true

# Specifies the log level.
# Is Required: No
# Valid values:
#    0 - No logging
#    Min: 1
#    Max: 6
# Default Value: None

##nativeLogLevel=0

#  Specifies the log file.
#  Is Required: No
#  Valid values:
#   fully qualified file name
#  No default value

##nativeLogFile=SAGSSXCLIENTA_SSX.LOG
```

```
# Default group to be automatically included for all requests
# that return any groups
# Is Required: No

##defaultGroup=DefGroup

# BaseBindDN where to find the users.
# Is Required: Yes
# and should contain the most detailed DN to find the users

#  personBindDn=ou=User,o=Org,dc=mycom,dc=com

# BaseBindDN where to find the groups.
# Is Required: Yes
# and should contain the most detailed DN to find the groups

##groupBindDn=ou=Groups,o=Org,dc=mycom,dc=com

# Attribute name of the password.
# Required when changeing the password
# Is Required: Not always
# Default value:
#  depending on serverType:
#  OpenLdap:
#  "userPassword"
#  SunOneDirectory:
#  "userPassword"
#  ActiveDirectory:
#  "unicodePwd"

##passwdField=userPassword

# Allow to pass a complete BaseBindDN
# via the domain parameter.
# Is Required: No
# Valid values: 0, 1

##allowdomainasbasebinddn=0

# Allow to specify which fields to search for as properties
# of a user entry
# Is Required: No
# Valid values: string, for example: "cn,sn,description"

##personPropAttr

# Allow to specify which fields to search for as properties
# of a group entry
# Is Required: No
# Valid values: string, for example: "cn,description"

##groupPropAttr
```

```
# Allow to use the special secure authentication using SASL,
# providing the directory supports this mechanism.
# Is Required: No
# Valid values: 0, 1 (default: 0)

##ldapSaslBind

# Allow to switch from a non-secure connection to a TLS connection,
# providing the directory supports this mechanism.
# of a group entry
# Is Required: No
# Valid values: 0, 1 (default: 0)

##ldapStartTls

# By default, the first "dc=" occurrence within the distinguished name
# name string denotes the domain name.
# If additional abbreviations want to be defined, one can use
# the following 2 parameter.
# Example:  Short="RnD;Admins;board"
#       with ↩
Long="ou=Rnd,ou=user,dc=mycom,dc=com;ou=Administrators,dc=mycom,dc=com;ou=VIP,dc=mycom,dc-com"

##ldapDomainShort
##ldapDomainLong

# If NOT the automatic domain name should be used to compose
# the canonical user id (SSXGetCanonicalUserId_A/W),
# specify this part of the ID here.

##canonicalDomainName

# Three algorithms are supported to find the groups of a user:
# "ru", recurse up: take the group pointer from the user entry
#                   and continue to search up for all groups
#                   found
# "rd", recurse down: search for all groups that have the
#                     user as member (no recursion)
# "cp", computed property: use a special field in the user
#                          entry to find all groups
#                          --> computedGroupProp retired
# Default: "ru"

##resolveGroups

# If resolveGroup is set to "cp", this parameter must provide
# the field name to look for in the user entry that denotes
# the user groups
# Default: None

##computedGroupProp=
```

```
# If the LDAP connection is protected by SSL/TLS, this
# parameter must be set.
# Valid Values: 0, 1
# Default: 0

##ldapSSLConnection=1

[SSX_CONFIGURATION-END]
```

Authorization Type ADSI

```
[SSX_CONFIGURATION]

# This is a sample properties file for the case
# when authType is ADSI and the user database is Active Directory

# Specifies the authentication type.
# Is Required: Yes
# Valid values: {"OS", "TEXT", "LDAP", "ADSI"}
# Default Value: None

  authType=ADSI

# Specifies the name of the AD Forest.
# Is Required: No, but should be specified
# Example: "dc=mycom,dc=com"
# (with a possible domain called "dc=eur,dc=mycom,dc=com")
# Default Value: None

##adsiForestDn

# Seconds how long auth. user remains in cache.
# Is Required: No
# Valid values:
#  0 - No cache
#  Min: 1, Max: No limit
# Default value: 180

  cacheTime=12

# Specify the max. number of cached users that have been successfully
# authenticated. When the cache overflows, the oldest entry is removed.
# Is Required: No
# Valid values:
#  0 - No cache
#  Min: 1, Max: No limit
# Default value: 300

  cacheSize=4
```

```
# Time (in seconds) how long to ignore any further authentication
# requests for a particular User-Id.

# Is Required: No
# Valid values:
#  Min: 1, Max: No limit
# Default value: 100

  denyTime=4

# Number of invalid logon attempts.
# Is Required: No
# Valid values:
#  Min: 1, Max: No limit
# Default value: 3

  denyCount=3

# Specifies an output file for logging.
# Is Required: No
# Valid values: (Valid log file path)
# Default Value: None
#   nativeLogFile=SIN_SSX.log

  logCallback=true

# Specifies the log level.
# Is Required: No
# Valid values:
#    0 - No logging
#    Min: 1
#    Max: 6
# Default Value: None

##nativeLogLevel=0

#  Specifies the log file.
#  Is Required: No
#  Valid values:
#   fully qualified file name
#  No default value

##nativeLogFile=SAGSSXCLIENTA_SSX.LOG

# In case the scope for the node to access users needs to be limited,
# one can specify a particular subtree:
# Example: "ou=user,ou=Rnd,dc=mycom,dc=com"

##adsiPersonBindDn

# In case the scope for the node to access groups needs to be limited,
# one can specify a particular subtree:
```

```
# Example: "ou=groups,ou=Rnd,dc=mycom,dc=com"

##adsiGroupBindDn

# By default, the first "dc=" occurrence within the distinguished name
# name string denotes the domain name.
# If additional abbreviations want to be defined, one can use
# the following 2 parameter.
# Example:  Short="RnD;Admins;board"
#       with   ↩
Dn="ou=Rnd,ou=user,dc=mycom,dc=com;ou=Administrators,dc=mycom,dc=com;ou=VIP,dc=mycom,dc-com"

##adsiDomainShort
##adsiDomainDn

# If NOT the automatic domain name should be used to compose
# the canonical user id (SSXGetCanonicalUserId_A/W),
# specify this part of the ID here.

##canonicalDomainName

# Three algorithms are supported to find the groups of a user:
# "ru", recurse up: take the group pointer from the user entry
#                   and continue to search up for all groups
#                   found
# "rd", recurse down: search for all groups that have the
#                     user as member (no recursion)
# "cp", computed property: use a special field in the user
#                          entry to find all groups
#                          --> computedGroupProp retired
# Default: "ru"

##resolveGroups

# If resolveGroup is set to "cp", this parameter must provide
# the field name to look for in the user entry that denotes
# the user groups
# Default: None

##computedGroupProp=

[SSX_CONFIGURATION-END]
```

Creating Internal User Repository Files

The following section describes the creation and administration of the internal user repository file, which is used with Authorization Type TEXT.

You can create and/or modify internal user repository files with the `ssxtxtpasswd` tool.

To start the `ssxtxtpasswd` tool, you use a command prompt. When you start the tool, you enter a user name and a password which are then encrypted (SHA512 and Base64) and provided in the result text file. The tool adds new or replaces existing user credentials in the text file.

Note: When you enter a user name, you can use only digits, Latin letters, and the following characters: ! ( ) - . ? [ ] _ ~ . When you enter a password, you can use only digits, Latin letters, and the following characters: !"#$%&'()*+,-./:;<=>?[\]^_`{|}~.

Example: Usage of ssxtxtpasswd tool

```
Tool to create or update an entry in the SSX text file based
user repository.

Usage: ssxtxtpasswd [-f filename] [-c] [-p password] [-d | -e] userId

Use "-c" to create a new file.
      Usually, the file should exist and user entries are
      replaced/added.

Use "-p" to provide the password on the command line instead
      via an extra prompt.

Use "-d" to remove the specified user entry from the text file.

Use "-e" to check, whether the userId is already stored in the text file.

Note: The password usually will be read via a non-echo command input.
      When no filename is specified, a default of "ssx_user" is assumed.
```

Example: Add User and Password

```
ssxtxtpasswd -f SAGInternalUserRepository.txt -c -p mypsw myuid

Hash: ↩
bOEOAPEEEJBKv+4zOELiYcFqY7qFhlLZz1ha7Ztf7j/drJHGy2MLOLXEu/kX7TD52Aj7XfwiZ+vpIl9DqRbVkA==
User entry for "myuid" successfully added
```

Contents of SAGInternalUserRepository.txt

```
*
*
* SAG Internal User Repository
*
version:3.0
*
user:myuid:$6a$bOEOAPEEEJBKv+4zOELiYcFqY7qFhlLZz1ha7Ztf7j/drJHGy2MLOLXEu/kX7TD52Aj7XfwiZ+vpIl9DqRbVkA==
```

3 Adabas Password Security (ADASCR)

Introduction

The Adabas database security utility ADASCR provides facilities for controlling access and update to Adabas files.

Adabas supports two classes of data access/update security: the first restricts user read and/or update requests on the basis of a whole file; the second restricts user access to individual records within a file.

Protection at File Level

Adabas files are security-protected if a file protection level greater than zero is assigned to the file. File protection levels are assigned separately for access (i.e. read) and update.

A user can access/update a security-protected file by entering a password with a permission level that is equal to or greater than the file protection level. Protection levels and password permission levels are assigned with the security utility ADASCR.

The file numbers that can be protected are limited by the block size of the ASSO1 container file. If the block size is 2KB, only files in the range 1 - 2047 can be protected (for 3KB the limit is 3071).

Protection at Record Level - Security by Value

Security by Value extends the Adabas File Protection Level security by enabling a user to further define separate data access and update restrictions according to the content of one or more fields in a data record.

Security by Value criteria are defined by using the ADASCR security utility. Each password may include value criteria for up to 99 separate Adabas files.

Record level security can only be used with security-protected files.

File Protection Levels

An Adabas file can be security-protected by assigning an access protection level and/or an update protection level greater than zero to the file.

File protection levels range from 0 to 15, with 15 being the maximum level of protection. A protection level of 0 means that all users can access/update the file. A value of 15 prevents all users from accessing/updating the file.

All Adabas files that have no protection levels assigned have a default protection level of zero. All users can access and/or update such files.

File	Protection Level Number	
	Access	Update
10	7	11
11	2	2
12	4	4

User Passwords

Separate access and update levels for each Adabas file can be assigned for each password. Password permission levels can be assigned in the range from 0 to 14.

Password	FILE 10 ACC/UPD	FILE 11 ACC/UPD	FILE 12 ACC/UPD
PASSWRD1	4/0	4/0	4/0
PASSWRD2	2/2	2/2	2/2
PASSWRD3	14/0	0/0	14/0
PASSWRD4	14/14	14/14	14/14
PASSWRD5	7/7	0/0	7/0

If the password access/update permission level is equal to or greater than the file access/update protection level, the password can be successfully used to access/update the file in question.

Using the examples for file protection levels and for password permission levels shown above,

- PASSWRD1 can be used to access file 11 or file 12.
- PASSWRD2 can be used to access/update file 11.
- PASSWRD3 and PASSWRD5 can each be used to access file 10 or file 12.
- PASSWRD4 can be used to access/update file 10, file 11 or file 12.

Security by Value Criteria

Separate access and update criteria for each Adabas file can be assigned for each password. Each value criterion consists of an Adabas search buffer and an associated value buffer.

The search and value buffers are constructed in the same manner as for regular Adabas search commands, including the use of non-descriptor fields and multiple value fields. However, soft-coupling and sub-, super-, hyper- and phonetic descriptor fields are not supported in Security by Value search criteria.

For further information on the syntax and construction of search buffers and value buffers, see the Command Reference Manual, Calling Adabas, Search and Value Buffers.

Value checking is performed only if data storage is either read or updated by the Adabas command.

The following table illustrates which criteria are tested for the various Adabas commands:

Adabas Command	Security by Value Check Performed	
	Test Criterion	Test Data
A1	Update	Before Image
E1	Update	Before Image
L1 - L6	Access	Before Image
L9	*(Access to index only, no value check performed)*	
N1, N2	Update	After Image
S1(*)	Access	Before Image

Note: * An S1 command with a valid format buffer is handled in the same manner as if an S1 command followed by an L1 command with a given User ISN had been issued.

Security by Value criteria are ignored if the associated security level of the requested file is zero.

Adabas Security Processing

Using the password entered by a user and the file-protection information defined for the file, Adabas checks whether the user is authorized to access/update a given Adabas file. If the file is not security-protected, Adabas ignores any password that is entered. The following describes Adabas security processing for read and update commands.

Adabas determines whether the file to be accessed or updated is security-protected. If there is no security protection, security processing stops.

Security Response Codes

If the file to be processed is security-protected, Adabas checks whether the password supplied is defined in the password table.

If the password is undefined or if no password has been supplied, response code 201 is returned.

If the password is defined but not valid for the file to be processed, response code 202 is returned.

If the password is valid for the file to be processed, Adabas checks the permission level associated with this password against the file's access or update protection level. Response code 200 is returned if the password's permission level is less than the file's protection level.

If the permission level is sufficient, the password is further checked for Security by Value criteria for the current file. If a search criterion has been defined for the supplied password, this is tested against either the before or after image of the data, depending on the Adabas command issued. If the Security by Value check is unsuccessful, response code 200 is returned, otherwise the user's request is finally granted and the Adabas command is processed.

The following is a summary of the response codes that may be returned by Adabas security processing:

RESPONSE 200

Explanation	Security violation detected.
User Action	Supply the correct password.

RESPONSE 201

Explanation	The password specified was not found.
User Action	Supply the correct password.

RESPONSE 202

Explanation	An attempt was made to use a file for which the user is not authorized.
User Action	Supply the correct password.

If an ET logic user receives the response codes 200 - 202, processing continues as if the user had exceeded the transaction time limit (see the TT parameter of ADANUC in the Utilities Manual for further information).

All Security by Value security violations cause response code 200 to be returned.

4 Adabas Encryption for Linux

Adabas Encryption for Linux provides functionality to protect data-at-rest from unauthorized access. With Adabas Encryption for Linux, data-at-rest refers to ASSO and DATA container files. Access to the information in these containers is controlled by means of encryption.

Prerequisite

A license for Adabas Encryption for Linux (AEL) is required.

Encryption of Data-At-Rest

Encryption Object

In the context of Adabas Encryption for Linux, data-at-rest refers to ASSO and DATA container files.

Encryption Algorithm

Encryption is the process of converting plain data into cipher text which cannot be deciphered without access to the encryption or decryption key.

Decryption is the process of converting encrypted data back to its original form. Decryption of encrypted data requires the decryption key; without the decryption key the original information cannot be recovered.

Adabas Encryption for Linux supports Advanced Encryption Standard (AES) in XTS mode, a symmetric-key algorithm, where the same key is used for both processes, encrypting and decrypting the data.

XTS-AES with a key length of 128 bits and XTS-AES with a key length of 256 bits are supported.

Encryption Process

When a database is created, the ASSO and DATA container files of an encrypted database are encrypted when the database is formatted. When the database is accessed, data in the ASSO and DATA container files is encrypted when written to disk, and decrypted on read access.

The encryption settings and the key to encrypt and decrypt data (the encrypted data encryption key) are stored in the database's ASSO container. This enables transparent access to the database for Adabas utilities and application programs.

The key to encrypt and decrypt the data encryption key (the key encryption key) is held in an external key management system. Adabas Encryption for Linux supports a local file-based key management system and AWS key management service.

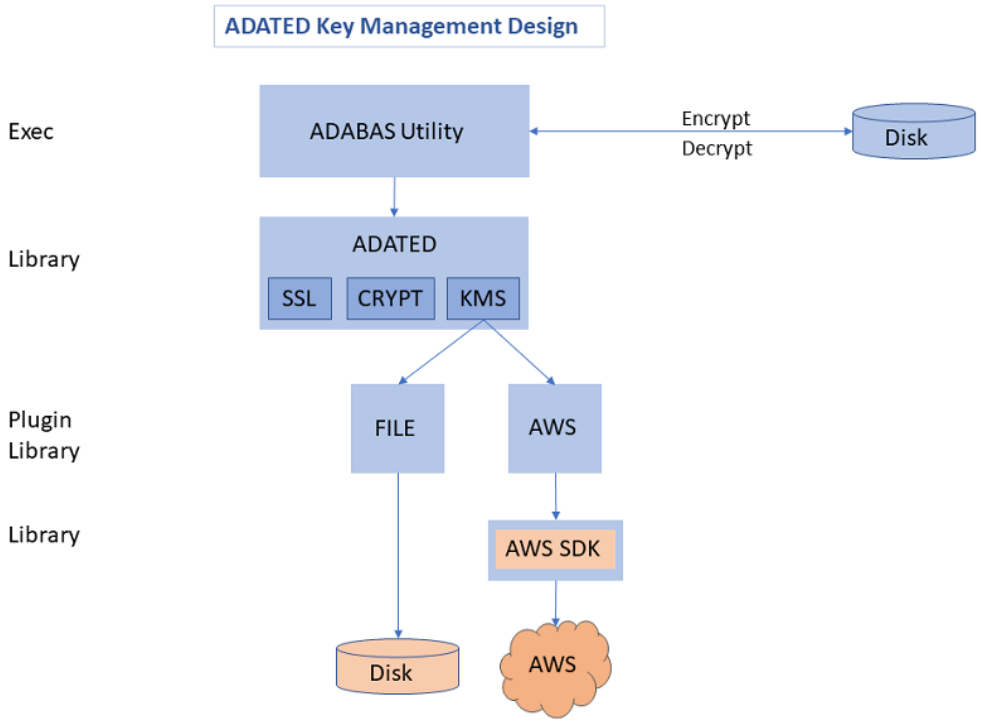
Note: Database encryption cannot be disabled.

Key Management System

A key management system (KMS), also known as a cryptographic key management system (CKMS) or enterprise key management system (EKMS), is an integrated approach for generating, distributing, and managing cryptographic keys for devices and applications.

Adabas currently uses two types of key management systems: a file-based management system on the local disk and the Key Management Service provided by Amazon.

This section describes how Adabas uses the KMS.



Adabas provides the ADATED library for the cryptographic functionality. The KMS function of ADATED uses plugin libraries, also provided by Adabas, for each supported key management system.

Adabas uses two types of encryption keys, the key encryption key (KEK) and the encrypted data encryption key (DEK). The KEK encrypts the DEK, which in turn encrypts the data in the database. The advantage of using two keys is that you can regularly change the key encryption key, so only the data encryption key is decrypted and newly encrypted. The data itself does not have to be decrypted and encrypted again.

Important: If the KEK is lost or corrupted, the data in the database is also lost regardless of which KMS is used.

Creating a Database with Encryption

To create a new encrypted database (GCB), acquire the keys for each database. Adabas automatically creates a new KEK with a unique name within the KMS. The KMS API provides a function to create an encrypted DEK. Adabas stores the encrypted DEK along with the unique name of the corresponding KEK in the GCB. The GCB block is the only block in the database that is not encrypted.

Select the target KMS by configuring the KMSTARGET parameter for the ADAFRM or ADABCK utility. The possible values are FILE (for a file-based KMS) and AWS (for the KMS provided by Amazon).

Running a Database with Encryption

When a database starts, the KEK name and the encrypted DEK are read from the ASSO container in the GCB. Both items are passed to the KMS to decrypt the DEK, which is kept in-memory for the entire time the database is running. To decrypt/encrypt any other block from the Adabas containers, you must use the decrypted DEK.

Blocks from the containers are decrypted during the read operation and stored in the buffer pool decrypted. Modified blocks are encrypted during the write operation of a buffer flush and written encrypted back to disk.

File-Based Key Management System

The file-based key management system uses a local disk file to store the key encryption keys. Database operations are performed only with the KMS file.

Select the file-based KMS by configuring the KMSTARGET=FILE parameter for either the ADAFRM or ADABCK utility when creating a database. The selected KMS type is stored in the database.

The file to store the keys is specified by the ADAKMSFILE environment variable. You can specify a fully qualified path name or directory name. The default location is $ADADATADIR/etc and the default file name is adatedkmsfile.db.

The KMS file is stored encrypted in the file system. Nevertheless, access to it must be restricted with operating system features. File permissions must allow access only to the user/group that will run Adabas utilities.

Amazon Key Management Service

Amazon Web Services (AWS) provides a built-in key management service which you can use when Adabas is on an AWS virtual machine. The AWS KMS allows for the creation and storage of the key encryption key. The encrypted DEK is passed to the AWS KMS for decryption.

Select the AWS KMS by configuring the KMSTARGET=AWS parameter for either the ADAFRM or ADABCK utility when creating a database. The selected KMS type is stored in the database.

Within AWS, you must configure your virtual machine to be able to access the AWS Key Management System. The Adabas AWS plugin uses a number of AWS APIs and KMS resources, which are managed through AWS Identity and Access Management (IAM). To use the APIs and access the specific KMS resources you must create an IAM role with a permission policy. The IAM role must be assigned to an EC2 instance (Amazon Elastic Compute Cloud) so that the EC2 instance can assume that role during its lifetime. The IAM role can be used for multiple EC2 instances.

For more information on how to create an IAM role and assign it to an EC2 instance, see the AWS Identity and Access Management and Amazon Elastic Compute Cloud documentation.

Administration

Database Creation

Encrypted databases are created with Adabas utilities ADAFRM and ADABCK.

- ADAFRM

  The control parameters ENCRYPTION and KMSTARGET specify the encryption algorithm and the key management system to be used. The formatted ASSO and DATA container files are encrypted according to the parameter values.

- ADABCK RESTORE/OVERLAY

  The control parameters ENCRYPTION and KMSTARGET specify the encryption algorithm and the key management system to be used. The formatted ASSO and DATA container files are encrypted according to the parameter values.

  As the target database is created and containers are encrypted, it should not already exist. For existing databases, these ADABCK control parameters ENCRYPTION and KMSTARGET will be rejected.

Database Report

The encryption settings include the encryption algorithm, the key management system and name of the key encryption key.

The ADAREP control parameter SUMMARY shows the encryption settings of an encrypted database.

The ADABCK control parameter SUMMARY shows the encryption settings of an Adabas backup copy of an encrypted database.

Database Backup and Restore

The Adabas backup copy of an encrypted database contains the encryption settings such as the encryption algorithm and key management system of the encrypted database. The data itself is not encrypted.

Restore as new database

To restore the Adabas backup copy of an encrypted database as a new database you can

- omit the control parameter ENCRYPTION. The encryption settings are taken from the Adabas backup copy.
- specify the control parameter ENCRYPTION. The encryption settings are taken from the parameter.

To restore an Adabas backup copy of an encrypted database as a new non-encrypted database, set the control parameter ENCRYPTION=NO.

Restore to an existing database

To restore an Adabas backup copy to an existing database, whether encrypted or not encrypted, omit the control parameter ENCRYPTION. The data is restored according to the target database's encryption settings.

Database Access

Adabas Utilities

Access to an encrypted database is transparent to Adabas utilities. The encryption information in the database's ASSO container is interpreted and data is encrypted and decrypted accordingly. This implies that non-encrypted data that is loaded into an encrypted database is encrypted when it is written to the container files. And encrypted data is decrypted on read access.

Adabas Direct Call Interface

Access to an encrypted database is transparent to the Adabas direct call interface. The encryption information in the database's ASSO container is interpreted, and data are encrypted and decrypted accordingly. This implies that non-encrypted data that is stored in an encrypted database is encrypted when it is written to the container files. And encrypted data is decrypted on read access.

Getting Started with Adabas Encryption for Linux

Prerequisite

A license for Adabas Encryption for Linux (AEL) is required.

Configuration

Adabas Encryption for Linux does not require any specific configuration.

Important: If the Adabas file-based key management system is used, all Adabas utilities require access to the file *$ADADATADIR/etc/adatedkmsfile.db*. Otherwise an encrypted database becomes unusable.

Creating an Encrypted Database

In order to create an encrypted database use either ADAFRM to format a new database, or ADABCK to restore or overlay an Adabas backup copy of a non-encrypted database.

- **ADAFRM**

```
ADAFRM ENCRYPTION=AES_256_XTS | AES_128_XTS
       KMSTARGET=FILE (default) | AWS
```

Example 1:

The following example shows how to create an encrypted database with the encryption algorithm AES_256_XTS. The Adabas file-based key management system is used (default: KMSTARGET=FILE)

```
adafrm dbid=100 datasize=100 assosize=100 worksize=100 encryption=aes_256_xts
```

- **ADABCK**

```
ADABCK  ENCRYPTION=AES_256_XTS | AES_128_XTS
        KMSTARGET=FILE (default) | AWS
```

Example 1:

The following example shows how to restore an encrypted database with the encryption algorithm AES_256_XTS from an non-encrypted Adabas backup copy. The Adabas file-based key management system is used (default: KMSTARGET=FILE). In this example, BCK001 is a backup copy of a non-encrypted database.

```
adabck dbid=100 restore=* encryption=aes_256_xts
```

Example 2:

The following example shows how to overlay an encrypted database with the encryption algorithm AES_128_XTS from an non-encrypted Adabas backup copy. The Adabas file-based key management system is used. In this example, BCK001 is a backup copy of a non-encrypted database.

```
adabck dbid=100 overlay=* encryption=aes_128_xts kmstarget=file
```

Note: The control parameter ENCRYPTION is rejected if the database to be created already exists.

Checking the Settings of an Encrypted Database

ADAREP SUMMARY and ADABCK SUMMARY can be used to show the encryption settings of an encrypted database.

- **ADAREP**

```
ADAREP SUMMARY
```

Example:

The following example shows the encryption settings of an encrypted database.

```
adarep dbid=100 summary
```

- **ADABCK**

```
ADABCK SUMMARY
```

Example:

The following example shows the encryption settings of an encrypted database.

```
adabck dbid=100 summary
```

Both reports show the current encryption algorithm, the key management system and the name of the key encryption key.

Creating a Non-encrypted Database from an Encrypted Database

ADABCK, can be used to restore or overlay a non-encrypted database from the backup of an encrypted database.

- **ADABCK**

```
ADABCK  ENCRYPTION=NO
```

Example:

The following examples show how to restore or overlay a non-encrypted database from an Adabas backup copy of an encrypted database. In these examples, BCK001 is a backup copy of an encrypted database.

```
adabck dbid=100 overlay=* encryption=no
```

```
adabck dbid=100 restore=* encryption=no
```

Using the Database

Access to an encrypted database is transparent to the Adabas utilities and to the Adabas direct call interface. The data are encrypted on write access, and decrypted on read access, according to the databases's encryption settings.

Example:

The following examples show that Adabas utility calls work transparently for both encrypted and non-encrypted databases. If necessary, non-encrypted data are encrypted during import or load operations.

```
adaord dbid=100 import=(1-60)
```

```
adavfy dbid=100 file=* data index
```

5 Ciphering

Important: The implementation of the ciphering feature is different to that which is available in Adabas for mainframes. The cipher code in Adabas for UNIX and Windows is static and is not provided in Additions 4.

The purpose of the ciphering in Adabas for UNIX and Windows is to prevent unauthorized analysis of Adabas container files; e.g. via file dumps, editors, etc.

Unlike ciphering on the mainframe, it does not prohibit unauthorized access to the data; as both database utilities and database applications can access the data without the cipher code.

Adabas can cipher the data that it stores in container files. This, however, only applies to the data records that are stored in the Data storage, but not the values stored in the inverted lists on the Associator.

Ciphering prevents the unauthorized analysis of Adabas container files. If ciphering is enabled (see below), data records are ciphered when they are stored in a database by either the Adabas nucleus or by the mass update utility ADAMUP. The data records are then deciphered when they are requested by a user or application. This means that the ciphering is completely transparent to the user or application.

Ciphering can be enabled for individual Adabas files. This is done when defining the file with ADAFDU by setting the `CIPHER/NOCIPHER` option. The ciphering process uses internal parameters in order to achieve a maximum level of security. In some systems, identical fields and records present a possible security risk: if an unauthorized user can decipher one, the other can also be deciphered. The Adabas ciphering process, however, treats identical fields and records as follows:

- Two identical fields within one record will be ciphered differently;
- Two identical records within one Adabas file will be ciphered differently;
- Two Adabas files with identical contents will be ciphered differently.

The following example demonstrates this on the basis of two fields in a record which both contain the value `TEST' (representations are hexadecimal):

```
Record 1  Unciphered=0x54455354  Ciphered=0xDD022537
Record 2  Unciphered=0x54455354  Ciphered=0x55EF0A51
```

Note: The ciphered values shown above are just examples, and do not represent the actual ciphering mechanisms used.

The Adabas ciphering mechanism is characterized by the following features and restrictions:

- System files (checkpoint and security) cannot be ciphered.
- ADAM key files cannot be ciphered.
- The output files produced by the utilities ADACMP (compression) and ADAULD (unload) are not ciphered.
- The data saved on files produced by the backup utility ADABCK, and the EXPORT files produced by the export utility ADAORD are ciphered.
- The restart and recovery records that are written to the WORK and PLOG files are ciphered.
- The output produced by the FILE function of the report utility ADAREP contains information about file ciphering.

6 Security Considerations

This section describes means or actions that that can or should be taken to secure (“harden”) the database.

Using the UNIX Group Concept

If the Adabas users belong to different UNIX groups, you can restrict the Adabas access to databases assigned to this group.

Note: This feature is only available for UNIX, not for Windows platforms.

Example:

Assume you have two UNIX groups called Production and Test. There are users belonging to the group Production, who should have access only to the production databases, and there are users belonging to the group Test, who should have access only to the test databases. Assume you have the following users for starting the database:

- dbaprod belongs to the group Production and should start the production databases
- dbatest belongs to the group Test and should start the test databases

The following is necessary to restrict the Adabas access to users of the group to which the databases belong:

- You must use two different NET_WORK_IDs, even if you are not using Net-Work. Because Adabas does not know if a Net-Work server will be started later, Adabas creates a shared memory section common with Net-Work, a GDT (global database table). The permission for GDT access is restricted to the group to which the Adabas nucleus belongs. Therefore, starting a nucleus fails if the same GDT is accessed as used by another nucleus belonging to a different group.

  You can use different GDTs if you start the nucleus with a different NET_WORK_ID because a separate GDT is created for each NET_WORK_ID. NET_WORK_ID is an environment variable, which must be set when the Adabas nucleus is started - two NET_WORK_IDs are considered to be equal if the first character is equal. If the environment variable NET_WORK_ID is not set, an empty NET_WORK_ID is used.

  In this example, you could start the production databases after setting NET_WORK_ID to P, and the test databases after setting NET_WORK_ID to T.
- The nucleus must be started with the parameter ADABAS_ACCESS=GROUP. Assume that you start in this example the Production databases with ADABAS_ACCESS=GROUP, but the Test databases with ADABAS=ACCESS=ALL (or without the parameter ADABAS_ACCESS). Then only the Production users can access the production databases, but all users can access the test databases.

Note: If you are using Net-Work, it is also necessary to start different Net-Work servers for different groups. You must take care to ensure that it is not possible for users to access databases via Net-Work for which they have no permissions.

Securing Administration of Adabas Role-Based Security

This section describes how to secure access to the ADARBA utility, which provides the functionality to administer the security definitions.

By default, all users with the **PUBLIC** role have access to and can execute the ADARBA utility.

The example below shows how access to ADARBA can be restricted by:

- Creating a *security administrator* role;
- Granting the role permission to execute the utility;
- Creating a user credential that is to be assigned the *security administrator* role;
- Granting the user *security administrator* role; and
- Revoking the permission to execute the utility from the role PUBLIC.

```
> adarba
> adarba: dbid=nnn

> adarba: create,role= security_administrator
> adarba: grant,operation=ada.uti.rba,to,role= security_administrator

> adarba: create,user=dbasec_userid
> adarba: grant,role= security_administrator,to,user=dbasec_userid

> adarba: revoke,operation=ada.uti.rba,from,role=PUBLIC
```

In this example, the following security definitions are created or modified:

- Created:
  - Role: *security_administrator*
  - User Credentials: *dbasec_userid*
  - Permission to Operation: **ada.uti.rba**
- Modified:
  - Role: **PUBLIC**
  - Permission to Operation: **ada.uti.rba**

Notes:

1. The role name and the user credentials are site-specific.
2. The credentials used for authorization must also be defined for authentication.

Preventing Loss of Administration Privileges

Lock Out Scenario

A 'lock out' scenario is possible. You must take appropriate action to prevent the 'loss of administration permissions'.

There is no recovery from a loss of administration privilege or a lock out scenario. This is a scenario in which no individual has access to the administration utilities, and the administration privilege for the security definitions has been lost.

The situation can be avoided by ensuring access to the following administration utilities.

- ADABCK
- ADADBM
- ADARBA

Recommended procedures to prevent a lock out scenario are to:

- Create more than one administration role;
- Assign access privileges to these administration roles;
- Assign the administration roles to more than one user credential.

In order to prevent an accidental loss of privileges, it is important to understand the processing logic behind the DROP function in ADARBA.

Drop Role Function

The DROP ROLE function in ADARBA deletes the following security definitions:

- All references to the ROLE definition are deleted:
  - The permissions granted to the ROLE are revoked;
  - The ROLE assignments to all USERS are revoked.
- The ROLE definition is deleted.

The function performs a cascading delete without issuing a warning message.

A loss of privileges scenario can occur if the administrator role is dropped, and no other role has access to the administration utilities.

Securing PUBLIC Access Privileges

You should review and restrict access privileges assigned to the role **PUBLIC**. This includes the roles assigned to the **PUBLIC** user, and the privileges assigned to the **PUBLIC** role.

PUBLIC access is provided when the RBAC system file has not been loaded, even though security is enabled.

You should take the following actions to secure access privileges assigned to **PUBLIC**:

- Ensure that the RBAC system file is loaded;
- Review the roles that are assigned to the **PUBLIC** user;
- Review the privileges that are granted to the **PUBLIC** role.

Securing Configuration Files

This section describes how to secure the configuration files used to configure authorization for Adabas utilities.

File	Description
`adaauth.ini`	Configuration of Authorization for Adabas Utilities
`adaaudit.ini`	Configuration of Audit Trail
`adarbac.ini`	Role-Based Access Control Definitions

To secure the configuration files, please ensure the following:

- **READ-ACCESS**
  All users, which execute an Adabas utility, must be able to read these files.
- **WRITE-ACCESS**
  Only the administrator of a file should have write access to the file.

The location of the configuration files is platform-dependent and is described in the section *Configuration of Authorization for Adabas Utilities* of the *Extended Operation* documentation.

Securing the Audit Trail Log File

This section describes how to secure the audit trail log file used by the Authorization for Adabas utilities.

File	Description
`adaaudit.log`	Audit Trail log file for Authorization for Utilities

To secure the audit trail log file, please ensure the following:

- **READ/WRITE-ACCESS**
  All users, which execute an Adabas utility, must be able to write to the audit trail log file.

The location of the audit trail log file is set via the `LOG_FILE` option in the *adaaudit.ini* configuration file and is described in the section *Configuration of Authorization for Adabas Utilities* of the *Extended Operation* documentation.

7 SSL Trusted Relationship for Natural

The main goal is to use the Adabas RBAC facility from Natural as client. Natural is at the moment not able to provide the user and password through the Adabas Client (using the `lnk_set_uid_pw()` function). The *SSL Trusted Relationship* should overcome this restriction. Neither Natural itself, nor any customer application should need to be changed.

The idea behind the *SSL Trusted Relationship* is that a client provides certificates over an SSL connection which are validated in Adabas. If the client provides certificates and they are successfully validated by Adabas, Adabas would assume that the client is allowed to access Adabas without presenting a password. The user id for the RBAC facility in Adabas will be taken from the so called *Adabas Session* or *Adabas ID*. The client (Natural) must make sure that the session (especially the user id) will be set up correctly.

Restrictions

We restrict the mechanism of trusted relationship on Natural as client. Only if these three conditions are fulfilled, Adabas trusts the incoming request:

1. Client uses ADATCPS (SSL) for the communication with Adabas
2. Client provides valid certificates that can be validated by Adabas
3. The client is Natural

The client is responsible to provide the user that will be authorized with the RBAC facility of Adabas. The user is taken from the Adabas Session (Adabas ID).

Adabas Operation

Adabas will check the validity of certificates at the connection level. As soon as a client connects via ADATCPS, Adabas will check whether the client provides client certificates. If so, they are validated in any case, regardless the setting of the SSLVERIFY nucleus parameter. A failed verification might not be an error, for example, if the SSLVERIFY is not set, clients need not to provide certificates, but credentials. If the SSLVERIFY option is set and the client certificate is invalid, then the connection will be rejected in any case. Adabas can be accessed in the traditional way in parallel. If using a secured database (RBAC), the clients must provide the user id and password (via the Adabas Client Interface) or use the user exit 21.

Client Configuration

On the client side ADATCP must be configured to use SSL and SSL certificates. This is done in normal cases in the $ACLDIR/config/dbmapping.txt file.

Example:

```
<dbid> = ↩
adatcps://<host>:<sslport>?cert_file=<path>/client_cert.pem&key_file=<path>/client_key.pem
```

Adabas Configuration

The Adabas nucleus must be configured to use SSL and SSL certificates. In any case, Adabas will not start without server certificates when SSL should be used. That is the minimum requirement.

To enable Adabas to accept trusted Natural users, the nucleus must be configured with the parameter:

```
SSLTRUST = NATURAL
```

Example:

```
[NUCPARMS]
ADATCP
PORTNUMBER = 0
SSLCAFILE = /atc/certs/ca_chain.pem
SSLCERTFILE = /etc/certs/adabas_cert.pem
SSLKEYFILE = /etc/certs/adabas_key.pem
SSLPORTNUMBER = 56220
SSLTRUST = NATURAL
SSLVERRIFY = 0
...
[NUCPARMS-END]
```

Setting the `PORTNUMBER` parameter to '0' disables the classic TCP/IP port and only SSL can be used.

8 GDPR Compliance

Adabas on Linux, UNIX and Windows stores the following personal data in log and audit files:

- Hostname of the machine
- User Identification (Local system credentials)

Therefore, so that your applications can be GDPR compliant, it is important for the Adabas administrator to be aware of the following files where personal data may be stored.

- Adabas Role-based Security
- Adabas Audit Trail
- Adabas Command Log
- Adabas Log file

Basic Actions to be taken

It is the responsibility of the database administrator to configure and delete files containing personal data. Adabas does not provide functionality to automatically delete personal data.

To ensure that personal data is not stored, the following is recommended:

- Disable logging features (when possible)
- Define a schedule and procedure to delete log files

The location and configuration of the database features storing personal data are described below.

Adabas Role-Based Security

When the Adabas Role-based Security feature is enabled, the following personal data is stored in the RBAC system file as a security definition:

- User Identification

This information is required and is used to determine the access privileges of a user.

Use the utility ADARBA to manage the security definitions.

Adabas Audit Trail

When the Adabas audit trail feature is enabled, the following personal data is logged:

- User Identification
- Hostname of the Adabas client
- Process Identification
- Timestamp of Access Request
- ET data
- Name of the access operation and the object of the operation

The location and configuration of the audit trail is feature-specific, for further information:

- **Authentication and Authorization for the Adabas Direct Call Interface**
  Please refer to ***Configuration*** in the section *Adabas Role-Based Security (ADARBA)*.
- **Authorization for the Adabas Utilities**
  Please refer to the section *Location of Configuration and Logging Files* under *Configuration of Adabas Role-Based Security* in the *Adabas Extended Operations* documentation.

The Audit Trail for Authorization for the Adabas Direct Call Interface is automatically enabled with the feature and cannot be disabled.

Adabas Command Log

When the Adabas Command Log feature is enabled, the following personal data is logged:

- User Identification
- Hostname of the Adabas client
- Process Identification
- Timestamp
- Adabas Command

When enabled, a command log file is created in the database directory. The file is identified by the value of the environment variable `NUCCLG`. The file can have multiple extends.

The Adabas Command Log feature can be enabled or disabled via the `LOGGING` control parameter of either utility:

- ADANUC

- ADAOPR

Please refer to either the *ADANUC* or *ADAOPR* section of the *Adabas Utilities*, for further information on the `LOGGING` control parameter.

Adabas Log File

When the Adabas log feature is enabled, the following personal data is logged:

- User Identification
- Process Identification
- Timestamp
- Name of the Adabas Utility with the requested function

The configuration file *ADABAS.INI* contains the location and configuration of the Adabas log file.

By default, the file name is *$ADADATADIR\etc\ADABAS.LOG*. The logging functionality is configured using the topic NODE_PARAMETER and its subtopics:

Subtopic	Description
ANALYSER	Enable the Adabas Extended Operations (AEO) feature
LOGGING	The Adabas log file is configured with the items: ■ ACTION - enables logging and ■ LOG_PATH - defines the file location
ARCHIVE_LOGFILE	Enables the archiving of the Adabas log file
ALERT	The Adabas log file entry can be processed by an alert routine. This is enabled and configured with the items: ■ ACTION - enables alert processing and ■ ACTION_ROUTINE - defines the alert routine

Please refer to the section *ADABAS.INI* under *Configuration Files* in the *Adabas Extended Operations* documentation, for further information on the location and configuration of the Adabas log file.