ADABAS SQL Server

Installation and Operations Manual for OpenVMS

Manual Order Number: ESQ142-010VMS

This document applies to ADABAS SQL Server Version 1.4 for OpenVMS and to all subsequent releases.

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

Readers' comments are welcomed. Comments may be addressed to the Documentation Department at the address on the back cover.

TABLE OF CONTENTS

PREFACE

The ADABAS SQL Server is SOFTWARE AG's implementation of the ANSI/ISO Standard for the SQL database language.

The primary goal of the ADABAS SQL Server is to provide an ANSI/ISO compatible database language interface to Software AG's database management system ADABAS.

Using This Manual – Some Basic Information

This manual is part of a set of four ADABAS SQL Server manuals, version 1.4.2. Three manuals contain information for various environments from mainframe and UNIX to OpenVMS platforms. This fourth manual, the ADABAS SQL Server Installation and Operation Manual, is the exeception to that rule. It is produced separately for each operating system.

This manual describes the installation and operation of the ADABAS SQL Server under OpenVMS on VAX and Alpha AXP platforms, respectively.

The following is a summary of the manual's chapters and their contents:

Preface	gives a brief overview of the manual and other manuals you may need to install and operate the ADABAS SQL Server;
Chapter 1	explains the prerequisites, the installation procedure and points of special interest with regard to installing the ADABAS SQL Server.
Chapter 2	explains how to start, operate and terminate the ADABAS SQL Server and how to generate user applications.
Chapter 3	explains how to invoke and work with the ADABAS SQL Server Utilities.
Chapter 4	describes the various types of logging facilities and how to activate them.
Appendix A	is valid for all supported platforms, not only for OpenVMS. It explains the parameter processing language (PPL) which allows the configuration of the various components of the ADABAS SQL Server.
Appendix B	A complete overview of the software and process architecture of the ADABAS SQL Server as a folding diagram (last page of this manual) and a short description of the elements therein.

Who Should Read This Manual

This manual is for those who plan to perform the installation of the ADABAS SQL Server and for those who manage or maintain the ADABAS SQL Server (such as database administrators and system programmers) and SQL application developers.

Other Helpful Manuals

Other manuals you may need are:

- ADABAS SQL Server Reference Manual
- ADABAS SQL Server Programmer's Guide
- ADABAS SQL Server Messages and Codes Manual
- ADABAS for OpenVMS Set of Documentation
- ENTIRE NET-WORK for OpenVMS Set of Documentation (including CSCI)
- ENTIRE BROKER Set of Documentation
- ANSI/ISO Standards SQL (X3.135-1989, ISO 9075).

1

INSTALLING THE ADABAS SQL SERVER SYSTEM

This chapter describes preparing for and installing the ADABAS SQL Server and the ADABAS SQL Utilities.

The method used for installation is a mixture of manual and automated functions which are executed in a simple straightforward manner via command files.

Installation Activities Overview

The rest of this chapter describes the steps required to install the ADABAS SQL Server System and the installation verification of the same.

Step 1 Check Prerequisites

Step 2 Start the Installation

Step 3 Check Directory Structure

Step 4 Install the ADABAS SQL Server System

Step 5 Install the ADABAS SQL Server Demonstration System

Installation Procedure

Step 1 Check Prerequisites

Disk Space

To install the ADABAS SQL Server, approximately 8000 blocks (VAX) or 14000 blocks (Alpha AXP) of hard disk space are required.

OpenVMS Operating System Requirements

In order to install the ADABAS SQL Server you must be running:

OpenVMS Version 5.5 or above on VAX platforms or

OpenVMS Version 6.1 or above on Alpha AXP platforms. You must have access to the SYSTEM account on the machine or to an account with the user privilege SETPRV.

Other Software

Before the ADABAS SQL Server can be installed, the system must be prepared with SAGBASE, the prerequisite for installing SOFTWARE AG products in an OpenVMS environment. For further information about SAGBASE, please refer to SOFTWARE AG's *SAGBASE Installation and Operations Manual*.

To use the ADABAS SQL Server, ADABAS for OpenVMS Version 3.2.1 or higher must be installed.

To operate the ADABAS SQL Server in a client/server mode where client and server are located on the same node the ENTIRE BROKER/CSCI for OpenVMS Version 1.1.0.3 or higher is required. In case of client and server being on separate nodes, ENTIRE NET-WORK for OpenVMS Version 3.2.1 must be installed as well.

Note:
ENTIRE BROKER/CSCI is packed together with ENTIRE NET-WORK.

Quotas and Privileges for the ADABAS SQL Server

The quotas and privileges required by ADABAS are sufficient for the operation of the ADABAS SQL Server.

Each active server consists of one detached OpenVMS process and one sub-process per server thread (as specified in the server parameter file). The process quotas PRCLM and MAXJOBS must be set accordingly. The SYSGEN parameter MAXPROCESSCNT has to be altered accordingly.

Step 2 Start the Installation

Overview

The ADABAS SQL Server is installed using VMSINSTAL, the command procedure that is used to install software products in the OpenVMS environment. VMSINSTAL guides the user through the installation procedure step by step, and uses an Installation Verification Procedure (IVP) to verify whether the installation is successful.

During the installation procedure, the following SYSGEN parameters are checked:

- Maximum number of working sets (WSMAX);
- Number of free global sections;
- Number of free global pages.

If any of these parameters are inadequate, the command procedure *nodename*_ ESQGEN.COM, which changes the corresponding SYSGEN parameters, is created on SAG$ROOT:[ESQ], and may be run by the system manager after installation. *nodename* is the name of your local node. If there is no name defined for your local node, the name “NONAME” will be inserted by the installation procedure.

WSMAX must be set to at least 2048 for VAX or 25000 for Alpha AXP.

To install the images for the ADABAS SQL Server the following system resources are required:

	VAX	Alpha AXP
Global Sections	8	8
Global Pages/Pagelets	2660	5400

and each server with default settings requires:

	VAX	Alpha AXP
Global Sections	4	4
Global Pages/Pagelets	760	880

For each additional thread, 2 additional free global pages/pagelets are required.

Process Quotas

The following process quotas are required to start a server with default settings:

	VAX	Alpha AXP
ASTLM	>= DIOLM + 18	>= DIOLM + 18
BYTLM	>= 32000	>= 32000
DIOLM	>= 32	>= 32
FILLM	>= 100	>= 100
PGFLQUO	>= 32768	>= 32768
TQELM	>= 40	>= 25
WSEXTENT	>= 8192	>= 25000
WSQUOTA	>= 8192	>= 25000

The installation procedure checks these values and modifies them if they are insufficient for the DBA account.

Depending on the number of threads started, the following UAF parameters must have at least the following values:

BYTLM	= number of threads * 1000
FILLM	= number of threads * 9
PGFLQUO	= number of threads * 5100
PRCLM	= number of threads * 2
TQELM	= number of threads * 2

The parameter JTQUOTA must be set to at least 8192 for each account which needs the ADABAS SQL Server logical names in its job environment.

Note:
If the SYSGEN parameters had to be changed it is advisable to run an AUTOGEN to improve system performance.

Installation Kit Structure

Software products which are to be run under OpenVMS are assembled into product kits. These kits are physically distributed as one or more save sets (files created by the OpenVMS backup utility).

The file name of each save set must be identical to the product name; it must also be assigned a unique file type that reflects the order in which the save sets are installed.

An ADABAS SQL Server installation kit, therefore, consists of several save sets with the following structure:

- **ESQ014.A**

  Standard contents of a product kit's primary save set
  - **KITINSTAL.COM**
    Command procedure invoked by VMSINSTAL to install the ADABAS SQL Server for OpenVMS
  - **KIT.DAT**
    Internal information used by KITINSTAL.COM
  - **HELP.COM**
    help information used by VMSINSTAL
  - **ESQ014.RELEASE_NOTES**
    Product release notes which can be selected with the N option from VMSINSTAL

- **ESQ014.B** Patch level 0 specific files
- **ESQ014.C** Patch level 0 example files
- **ESQ014.D** Patch level 1 specific files
- **ESQ014.E** Patch level 1 example files

etc.

If there are no changes for a specific part, the corresponding save sets contained in the kit may be empty or missing.

Example of an Initial Installation

```
$ @sys$update:vmsinstal

     OpenVMS Alpha AXP Software Product Installation Procedure V7.0

It is 25-JUL-1996 at 18:20.

Enter a question mark (?) at any time for help.

%VMSINSTAL-W-NOTSYSTEM, You are not logged in to the SYSTEM account.

* Do you want to continue anyway [NO]? y

* Are you satisfied with the backup of your system disk [YES]?

* Where will the distribution volumes be mounted: DISK:

Enter the products to be processed from the first distribution volume set.

* Products: ESQ

* Enter installation options you wish to use (none):

The following products will be processed:

  ESQ V1.4

     Beginning installation of ESQ V1.4 at 18:21

%VMSINSTAL-I-RESTORE, Restoring product save set A ...

+------------------------------------------------------------------------+
|                                                                        |
|         Software AG Product Installation Procedure                     |
|                                                                        |
+------------------------------------------------------------------------+

  You are preparing a(n)
  ADABAS SQL SERVER (OpenVMS) Version 1.4.2 initial installation
  on a DEC 3000 Model 300 with the cpu-id %X00000000/00000000

%ESQ-I-NOTEXIST, File SAG$ROOT:[ESQ]VERSION.DAT does not exist,
                 it will be provided by this installation

%ESQ-I-VERPL, ADABAS SQL SERVER (OpenVMS) V1.4.2-0
                 initial installation

%ESQ-I-INSTINFO, The sysgen parameters are setup properly
                 to use the ADABAS SQL Server Version 1.4.2

* Do you want to copy the Examples ?              [YES]:
```

```
%ESQ-I-SPACEOK, This ADABAS SQL SERVER (OpenVMS) installation
                requires 13764 blocks

%ESQ-I-DIREXIST, Directory SAG$ROOT:[000000.ESQ] created.

%ESQ-I-DIREXIST, Directory SAG$ROOT:[ESQ.V142] created.

%ESQ-I-DIRCREATED, Directory SAG$ROOT:[ESQ.V142.EXAMPLES] created.

  This ADABAS SQL SERVER kit contains a READ_ME_FIRST file,
  named READ_ME_FIRST.142.
  It will be moved to directory SAG$ROOT:[ESQ]
  by this installation procedure.
  Please read this file after VMSINSTAL has finished.

%ESQ-I-MOVE, Moving file READ_ME_FIRST.142 to SAG$ROOT:[ESQ]

* Print READ_ME_FIRST.142 (queue SYS$PRINT)?      [YES]: n

%VMSINSTAL-I-RESTORE, Restoring product save set B ...

%VMSINSTAL-I-RESTORE, Restoring product save set C ...

%ESQ-I-MOVE, Moving files to their target directories ...

%ESQ-I-MOVE, Moving login procedure LOGIN.COM to SAG$ROOT:[ESQ]

* Move STARTUP_ESQ.COM to SYS$STARTUP ?           [YES]:

%ESQ-I-MOVE, Moving startup procedure STARTUP_ESQ.COM
               to SYS$STARTUP

* Enable STARTUP_ESQ.COM using SYSMAN ?           [YES]:

%ESQ-I-INSTINFO, Remove file entry STARTUP_ESQ.COM in
               system startup database on node NODENAME

%ESQ-I-INSTINFO, Add file entry STARTUP_ESQ.COM in
               system startup database on node NODENAME
               phase END mode DIRECT

%ESQ-I-SETPROT, Setting protection on new files ...
```

```
%ESQ-I-INSTINFO, Modify account DBA with settings required by
                 ADABAS SQL SERVER (OpenVMS)
%ESQ-I-INSTINFO, required privilege PRMGBL
%ESQ-I-INSTINFO, required default privilege PRMGBL
%ESQ-I-INSTINFO, validating WSQUOTA   >= 25000
%ESQ-I-INSTINFO, validating WSEXTENT  >= 25000
%ESQ-I-INSTINFO, validating DIOLM     >= 32
%ESQ-I-INSTINFO, validating ASTLM     >= 50
%ESQ-I-INSTINFO, validating FILLM     >= 100
%ESQ-I-INSTINFO, validating TQELM     >= 40
%ESQ-I-INSTINFO, validating BYTLM     >= 32000
%ESQ-I-INSTINFO, validating PGFLQUO   >= 32768

%ESQ-I-EXECUTE, Executing startup procedure
               SYS$STARTUP:STARTUP_ESQ.COM

%ESQ-I-CREATE, Creating patch level file SAG$ROOT:[ESQ.V142]ESQ_PL.DAT

%ESQ-I-CREATE, Creating version file SAG$ROOT:[ESQ]VERSION.DAT

          Starting Verification Command Procedure for
          ADABAS SQL SERVER (OpenVMS) 1.4.2

%ESQ-I-VERIFY, IVP completed successfully.

        The ADABAS SQL SERVER (OpenVMS) 1.4.2
        initial installation completed successfully

        For further installation steps, please refer to
        installation notes of ADABAS SQL SERVER (OpenVMS) V 1.4.2.

     Installation of ESQ V1.4 completed at 18:26

    Adding history entry in VMI$ROOT:[SYSUPD]VMSINSTAL.HISTORY

    Creating installation data file: VMI$ROOT:[SYSUPD]ESQ014.VMI_DATA

Enter the products to be processed from the next distribution volume set.
* Products:
     VMSINSTAL procedure done at 18:27

$ logout
```

Step-by-step Installation Description

The ADABAS SQL Server is installed by executing the following steps.

1 **Invoke VMSINSTAL as follows**:

Log in to the system manager's account.

Note:
For an update installation, the UAF parameter JTQUOTA of the account which is used for the installation must be set to 8192, because the job table is used to define all necessary logical names.

Establish the default directory SYS$UPDATE:

```
$ set default sys$update
```

Start the installation procedure by entering the following command:

```
$ @vmsinstal
```

During the installation procedure, a number of general information messages are displayed. Read all messages carefully and follow any advice they may provide.

The following messages are displayed when the procedure is started:

```
VAX/ALPHA AXP Software Product Installation Procedure Vx.y

It is <dd-mmm-yyyy> at <hh:mm>.
Enter a question mark (?) at any time for help.
```

where <dd-mmm-yyyy> and <hh:mm> are the current date and time.

If DECnet is active on the system, the following message appears:

```
$VMSINSTAL-W-DECNET, Your DECnet network is up and running.
```

If other users are accessing the system, the following message appears:

```
$VMSINSTAL-W-ACTIVE, The following processes are still active:
<name>
.
.
.
* Do you want to continue anyway [NO]?
```

where <name> refers to the process name of a user logged into the system. Enter YES and continue processing; the installation of the ADABAS SQL Server is not affected if users are active.

The following message is then displayed:

```
* Are you satisfied with the backup of your system disk [YES]?
```

It is not necessary to back up the system disk because the files and directories that are installed by the ADABAS SQL Server can be removed easily. The installation of the ADABAS SQL Server does not affect any files on the system directories.

Press RETURN if you are satisfied.

The following message is then displayed:

```
* Where will the distribution volumes be mounted:
```

Enter the name of the device on which the distribution medium is to be mounted.

2 **Now enter the products to be processed from the first distribution volume set.**

```
* Products: esq
* Enter installation options you wish to use (none):
```

If the distribution medium is not already mounted on the specified device, VMSINSTAL asks for the distribution medium to be mounted on the device specified when VMSINSTAL was invoked or when the response to the device prompt was entered. If, for example, the ADABAS SQL Server is to be installed from the device DISK:, VMSINSTAL will display the following:

```
Please mount the first volume of the set on DISK:.
```

VMSINSTAL then displays the following:

```
* Are you ready ?
```

You should now mount the first volume of the distribution medium.

Enter YES and press RETURN when the volume has been mounted. VMSINSTAL now attempts to mount the distribution medium. If it succeeds, a message is displayed, e.g.:

```
$MOUNT-I-MOUNTED, <label> mounted on _DISK:

The following products will be processed:
       ESQ V1.x

       Beginning installation of ESQ V1.x at <hh:mm>

    $VMSINSTAL-I-RESTORE, Restoring product saveset A ...
```

If you specify “N” in response to the options prompt at the start of this step, the following is displayed if online release notes are part of this product shipment:

```
1.  Display release notes
2.  Print release notes
3.  Both 1 and 2
4.  Copy release notes to SYS$HELP
5.  Do not display, print or copy release notes

* Select option [2]:
```

Select one of the options. Depending on the option specified, you may be prompted for a queue name.

VMSINSTAL then displays the following message:

```
* Do you want to continue the installation?  [NO]:
```

The following message is then displayed:

```
$VMSINSTAL-I-REMOVED, Product's release notes have been moved to SYS$HELP
```

3 **Create the command procedure <node_name>_ESQGEN.COM.**
It changes SYSGEN parameters, if some system parameters have not been set to the values required to run the ADABAS SQL Server.

VMSINSTAL then displays the following prompt:

```
* Do you want to execute <node_name>_ESQGEN.COM ?  [YES]:
```

Press RETURN to execute the command procedure <node_name>_ESQGEN.COM. Enter NO and press RETURN if you do not want to execute the command procedure. Because <node_name>_ESQGEN.COM changes system parameters, VMSINSTAL displays the following to ask whether you want to reboot the system:

```
* Reboot the system after the installation ?        [NO]:
```

Enter YES and press RETURN if the system is to be rebooted after the installation has completed. This makes the new system parameter settings available.

4 ***VMSINSTAL now displays the following prompt:***

```
* Do you want to copy the Examples ?                [YES]:
```

Press RETURN to copy the examples. Enter NO if the examples are not to be copied to the examples directory. If the installation kit does not contain any example files, this prompt will not be displayed.

The ADABAS SQL Server installation procedure continues after this prompt and displays messages that provide information about the status of the installation (see sample installation).

5 **VMSINSTAL then displays the following prompt:**

```
* Move STARTUP_ESQ.COM to SYS$STARTUP?          [YES]:
```

Before the ADABAS SQL Server can be used, the procedure STARTUP_ESQ.COM must be executed. SOFTWARE AG recommends that the procedure is executed during system startup. By default, STARTUP_ESQ.COM is moved to the SYS$STARTUP directory; press RETURN to accept this default. If you enter NO, STARTUP_ESQ.COM is moved to the directory SAG$ROOT:[ESQ].

If the default value of the last prompt is accepted, VMSINSTAL then displays the following prompt:

```
* Enable STARTUP_ESQ.COM using SYSMAN?          [YES]:
```

Press RETURN to generate an entry in the system startup database in order to execute the procedure automatically during system startup. The following entry will be generated:

```
Phase     Mode    File
--------- ------- -----------------
END        DIRECT  STARTUP_ESQ.COM

Enabled on <node_name>
P1: STARTUP
```

where <node_name> is the name of your local node. If there is no name defined for your local node, ”All Nodes” is inserted.

VMSINSTAL continues by setting the protections on the files provided, validating the DBA account, executing STARTUP_ESQ.COM and creating the files ESQ_PL.DAT in directory ESQ$VERSION and VERSION.DAT in directory ESQ$MAIN.

The Installation Verification Procedure (IVP), which is part of the installation procedure, runs automatically after the installation.

The ADABAS SQL Server is now installed.

The ADABAS SQL Server Startup Procedure

The procedure SYS$STARTUP:STARTUP_ESQ.COM defines the current version of the ADABAS SQL Server.

The version number can be passed to the procedure via the parameter P1. Alternatively, if P1 is “STARTUP” or not specified, the version number is taken from the first line of the file VERSION.DAT.

The logical names which are used in the ADABAS SQL Server environment and their definitions for version 1.4.2 are as follows:

```
"ESQ$EXAMPLES"     =    "SAG$ROOT:[ESQ.V142.EXAMPLES]"
"ESQ$MAIN"         =    "SAG$ROOT:[ESQ]"
"ESQ$NODDIR"       =    "SAG$ROOT:[ESQ.node]"
"ESQ$VERSION"      =    "SAG$ROOT:[ESQ.V142]"
"ESQLNK"           =    "ESQ$VERSION:ESQLNK142.EXE"
"ESQSRVRT"         =    "ESQ$NODDIR:ESQ_ROUTING.DAT"
"ESQTHS"           =    "ESQ$VERSION:ESQTHS142.EXE"
"ESQVOLIB"         =    "ESQ$VERSION:ESQVOLIB.EXE"
"ESQVOSYS"         =    "ESQ$VERSION:ESQVOSYS.EXE"
```

If these logical names are not already present in the system table, they are added to it with the definitions shown above.

If these logical names are already present in the system table (with definitions, for example, from a previous installation of the software), they are added instead to the job table with the definitions shown above.

To enforce the use of the system table, de-assign the logical name ESQ$VERSION from the system table before calling the procedure. This is done with the following command:

```
$ DEASSIGN/SYSTEM/EXEC ESQ$VERSION
```

The procedure STARTUP_ESQ.COM installs the images defined by the logical names ESQLNK, ESQVOLIB, ESQVOSYS, ESQTHS and ADACSL. The privileges CMKRNL, SYSGBL and PRMGBL are required. If they are not set, no images will be installed. The installation of the images can be performed version-specific:

- If the procedure is called without a parameter or with the parameter STARTUP, the images for all versions defined in the file VERSION.DAT will be installed.
- If the procedure is called with a version string as the parameter value, for example 142, only the specific images for version 1.4.2 will be installed.

Installing the ADABAS SQL Server in a Cluster Environment

In a cluster environment, the ADABAS SQL Server must be installed on each node on which the ADABAS SQL Server is to be used.

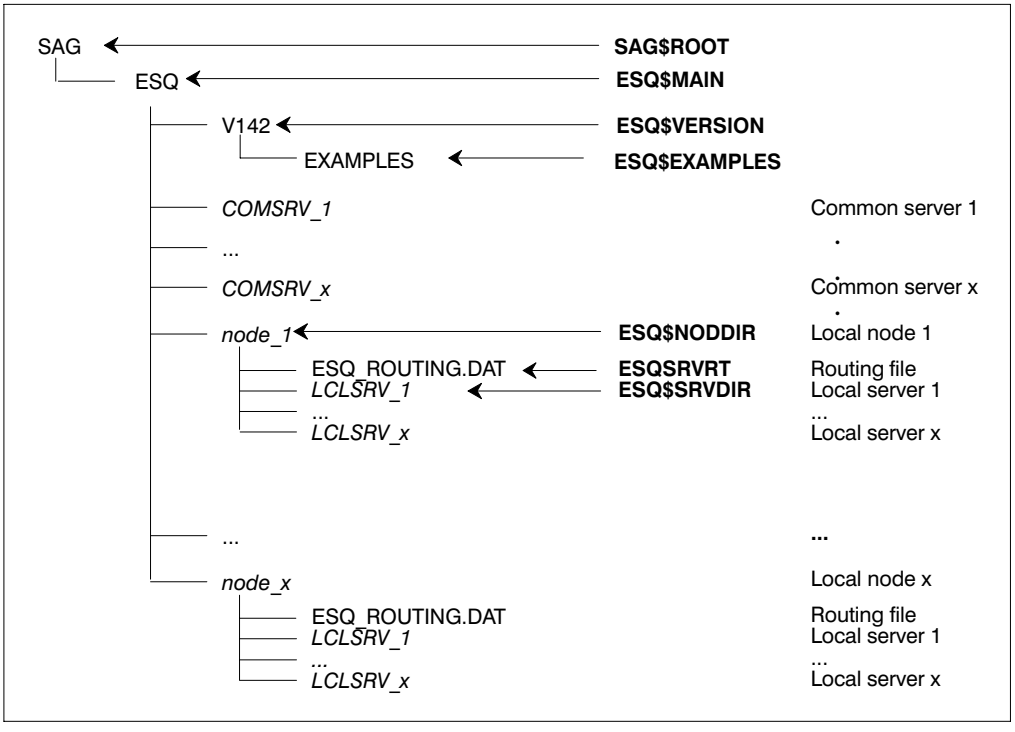
Depending on the preceding SAGBASE installation, the initial ADABAS SQL Server installation within a cluster will choose either the common or specific root directory. Each node on which the ADABAS SQL Server is to be used must first be prepared by installing SAGBASE. The installation procedure checks for an existing ADABAS SQL Server installation in the cluster.

A subsequent cluster installation will then consist of:

- checking the SYSGEN parameters on the node in question;
- providing the SYSMAN entry for this specific node;
- executing the procedure SYS$STARTUP:STARTUP_ESQ.COM;
- specifying the node name in the file ESQ$VERSION:ESQ_PL.DAT.

Step 3 Check Directory Structure

The upper portion of the ADABAS SQL Server directory structure is generated during the VMSINSTAL steps as described in Step 2: Start the Installation. The lower portion will be generated during the execution of Step 4: Install the ADABAS SQL Server System later in this chapter. The logical names pointing to the specified directory are created automatically during the installation.



ESQ$MAIN

This directory contains all currently installed versions of the ADABAS SQL Server together with the environments for the individual servers.

During installation, the following files are created on this main directory:

LOGIN.COM	Login procedure for the ADABAS SQL Server database administrator (DBA). LOGIN.COM runs the version-dependent command procedure ESQ$VERSION: SYMBOLS.COM which defines the symbols that are necessary to run the ADABAS SQL Server under OpenVMS. Set up your working environment so that LOGIN.COM runs automatically every time you log in to OpenVMS.
VERSION.DAT	ADABAS SQL Server version definition file.
ESQ.LOG	Common ADABAS SQL Server log file. This file will be created when executing ESQGEN.COM.

ESQ$VERSION

This directory contains the executable images, command files, etc. for the current version of the ADABAS SQL Server.

ESQ$EXAMPLES

This directory contains the command files and source files needed to build the SAGTOURS demonstration system, which can be used for the installation verification of the ADABAS SQL Server.

ESQ$NODDIR

This directory contains the server directories for the local server environments generated on the local node. The directory will be created when executing ESQGEN.COM.

ESQ_ROUTING.DAT	ADABAS SQL Server routing file. This file will be copied by ESQGEN.COM from ESQ$VERSION into this directory.

ESQ$MAIN:[COMSRV_x]

This directory contains the server-specific files for the common server *COMSRV_x,* namely: parameter files, log files, etc.

ESQSRV.LOG	ADABAS SQL Server log file for the server *COMSRV_x*. This file will be created when executing ESQGEN.COM.

ESQ$NODDIR:[LCLSRV_x]

This directory contains the server-specific files for the local server *LCLSRV_x* on node *node_x,* namely: parameter files, log files, etc.

ESQSRV.LOG	ADABAS SQL Server log file for the server *LCLSRV_x*. This file will be created when executing ESQGEN.COM.

Step 4 Install the ADABAS SQL Server System

To run the ADABAS SQL Server, it is necessary to create an SQL catalog and the SQL error message text file in an ADABAS database. This is done as follows:

1 If the database does not already exist, create it using DBGEN. Make sure that the sizes of the containers DATA, WORK and ASSO are adjusted to fit your needs.

2 Customize the files ESQCAT1.FDUINP, ESQCAT2.FDUINP, ESQCAT3.FDUINP and ESQERR.FDUINP in the directory ESQ$VERSION to meet your current ADABAS environment. The default values are as follows:

ESQCAT(1–3).FDUINP

```
DSSIZE   =    100B
MAXISN   =   5000
NISIZE   =   1000B
UISIZE   =    100B
```

ESQERR.FDUINP

```
DSSIZE   =    40B
MAXISN   =   1500
NISIZE   =    20B
UISIZE   =    10B
```

Before loading the SQL catalog and SQL error message text file, you should verify that there is sufficient free storage in the database. This can be done using the ADABAS utility ADAREP:

```
$ arep db=dbid, free_space
```

Analyze the output and ensure that there is sufficient space in the ADABAS database. If there is insufficient space, use the ADABAS utility ADADBM to add new space to the database.

3 Before starting your ADABAS database, ensure that the nucleus parameters are set as required for the ADABAS SQL Server. These values may have to be adapted according to the size of the container files for your ADABAS database. The ideal values are:

```
LBP  =   1000000          LDEUQP   =   150000          LFP  =    20000
LP   =      1000          LS       =    10000          LU   =    33000
LWP  =    100000          NAB      =       64          NH   =     1000*
```

Note:
** The NH parameter should be set by the DBA depending on the local requirements. For the installation verification of the ADABAS SQL Server we recommend a value of 1000.*

Note:
** The Migration Utility requires a WORK Block Size of 8192 with the LP parameter set to 1000.*

4 Start the ADABAS database.

5 Generate one or more server environments within the ADABAS SQL Server.

```
$ esqgen  servername  dbid  catalog_base_file  error_file [desc][sec][common].
```

The expected values for the specification of the above parameters are:

servername	is the name of the server to be generated with a maximum length of 8 characters. Server names are always treated as uppercase,
dbid	is the ADABAS database ID for the catalog and error files,
catalog_base_file	is the ADABAS file number which is to be used for the catalog base file. IMPORTANT: This file number and the next two file numbers in addition must be free prior to server generation, because the catalog occupies three consecutive ADABAS files.
error_file	is the ADABAS file number for the ADABAS SQL Server error texts which may be shared by several servers,
[*desc*]	is an optional short description of the server with a maximum length of 35 characters. A description must be entered, whereby an empty string qualifies, when either keyword 'sec' and/or 'common' is specified. If the description contains an empty string/blanks or lowercase characters, it must be enclosed in doubles quotes.
`[sec]`	is an optional keyword for a server to be generated with security features. If omitted, a server specification without security features is assumed. For details refer to the following pages, section **To be Considered when Deciding for/against a Server with Security Feature**s for a brief overview or to the *ADABAS SQL Server Programmer's Guide*, chapter **The ADABAS SQL Server Security Concept.**

[common]	optional keyword for a common server. If omitted, a local server specification is assumed. A local server can only be activated on the same node where the corresponding ESQGEN command was executed. A common server can be activated on any node that shares the ESQ$MAIN file system with the node where the corresponding ESQGEN command was executed.

The result of the installation process so far is a new catalog (three consecutively placed ADABAS files containing meta data of the schema and table structures) and an SQL error message text file (one ADABAS file containing SQL error message text records) within your ADABAS database. Furthermore, a server-specific directory (ESQ$SRVDIR, set by an ESQSET command as described below) has been generated to contain server-specific parameter template files, log files, trace files, etc. Furthermore, the user DBA has been created without a password.

To be Considered when Deciding for/against a Server with Security Features

To switch between security/non-security servers:

If an ADABAS SQL Server has been generated without the security feature, it is not easily possible to change it to a version that supports security features, and vice versa.

- The following steps are necessary to change from a non-security version to a security version:
  - use the Migration Utility to extract all information from the catalog. This results in a file containing numerous relevant DDL statements.
  - edit this file and manually enter all necessary GRANT statements.
  - generate a new server with the security feature. Use the amended Migration Utility file to populate the new catalog via the BASIC Interactive Facilities (esqint).
- The following steps are necessary to change from a security version to a non-security version:
  - produce a Migration Utility output file.
  - edit the file and remove all GRANT statements.
  - generate a new server without the security feature. Use the amended Migration Utility file to populate the new catalog via the BASIC Interactive Facilities (esqint).

The following three points are valid in either version, security or non-security:

- the USER must still be defined via the CREATE USER statement.
- the user DBA is the only one authorized to create a schema. The owner of a schema is the only one authorized to create tables, views, etc. This is true, whether the ADABAS SQL Server has been generated with or without the security feature.
- full password support is included, the CREATE/DROP/ALTER USER statements can be executed as well as the CONNECT statement with user specification.

Consequences of Generating a Server Without the Security Feature:

If the ADABAS SQL Server is generated ***without*** the security features the following consequences apply:

- at runtime, no security check is performed. Access rights to a specified table, etc. will not be checked. For this reason, the performance of the server will be more favorable than in a security version.
- the execution of GRANT/REVOKE statements is not possible.
- those tables established to hold privilege-related data in the catalog will be created but will be empty (for example, the table: table_privileges).

Consequences of Generating a Server With the Security Feature:

The following point must be regarded if an ADABAS SQL Server is generated ***with*** the security features:

- The user DBA is treated like any other user. A SELECT statement against any of the INFORMATION_SCHEMA tables will return results for those objects only which have been GRANTed access to. To see all information, the DBA should use the DBA_SCHEMA tables (which are only accessible to the DBA). For further details, see ”Appendix C – The ADABAS SQL Server Catalog Structure” of the Programmer’s Guide.

Verification

To verify that the catalog and the SQL error message text file for the generated server have been loaded correctly, use the ADABAS utility ADAREP. The following files should have been established by the esqgen command:
ESQCAT1_*servername*
ESQCAT2_*servername*
ESQCAT3_*servername*
ESQERR_*servername*

If, for some reason (e.g., not enough space in the database) an ‘esqgen’ aborts or concludes unsuccessfully, the file system directory for the server can be removed by typing:

```
$ esqremove servername
```

The catalog and error text files in ADABAS can be removed by typing:

```
$ esqremove servername /all
```

Once the problem has been solved, re-try the server generation using ‘esqgen’ as described in Step 4, instruction No. 5.

6 Customize the server parameter files. Set a newly created server to your current default by typing:

```
$ esqset servername
```

The logical name ESQ$SRVDIR now points to the server-specific OpenVMS directory. This directory holds three parameter files which are preset with default values but may be customized to your needs:

- **ESQSRV.PAR (server parameter file)**
  This parameter file is used when starting a server process (for client/server processing), for example, to adjust the number of thread processes started by the server, depending on the number of concurrent sessions.
- **ESQPC.PAR (precompiler parameter file)**
  This parameter file is used when starting the precompiler.
- **ESQRUN.PAR (runtime parameter file)**
  This parameter file is used when starting an application using an ADABAS SQL Server.

For details on how to modify the above parameters refer to **Appendix B — The Parameter Processing Language (PPL)** in this manual.

7 Assign a password to the user DBA

During the installation of the server, a DBA user was created without a password. If the server has been created with the security feature turned on, you may wish to assign a password to the DBA account now.

```
$ esgint

ESQ User: dba
Password: (carriage return)

esqint: alter user dba set password "<new password>";
```

The password will have to be dropped again if you later wish to install the Demonstration System (SAGTOURS) as described below.

8 The ADABAS SQL Server installation is now complete.

Step 5 Install the ADABAS SQL Server Demonstration System (SAGTOURS)

The ADABAS SQL Server Demonstration System allows you to establish an environment from which the ADABAS SQL Server installation can be tested. Before this demonstration system can be installed, a server environment, for example a server named SAGTOURS, must have been generated via the esqgen command.

The SAGTOURS system consists of five SQL tables CONTRACT, CRUISE, PERSON, SAILOR and YACHT, and optionally BOOKING, contained as files in your ADABAS database. The table BOOKING is needed only, if the demonstration package for the ADABAS SQL SERVER / WWW Common Gateway Interface will be installed.

Example:
Create the SAGTOURS demonstration system tables by typing the following:

```
$ set default ESQ$EXAMPLES
$ esqset SAGTOURS
$ @crttabs 4 5 6 7 8 9
```

The create table command file (CRTTABS.COM) creates the ADABAS file CONTRACT with file number 4, CRUISE with file number 5, PERSON with file number 6, SAILOR with file number 7, YACHT with file number 8, and BOOKING with file number 9. In addition, the schema SAGTOURS is created with an owner of ‘esq’. This schema contains the tables established using CRTTABS.COM.

Note:
The tables are established with the schema identifier SAGTOURS.

Any warnings that may be produced by the Precompiler should be ignored as they are of informational value only.

The ADABAS Utility ADAREP can be used to verify that the ADABAS files containing the SAGTOURS tables have been successfully created.

If, for any reason, the creation of the SAGTOURS tables aborted, or was completed unsuccessfully, for example, because the database is too small, the SAGTOURS system can be cleared by typing:

```
$ set default ESQ$EXAMPLES
$ @drptabs
```

which drops all tables already created. Once the cause of the failure has been located and repaired, the SAGTOURS system can be restarted using “crttabs.com” as described above.

To perform a single row SELECT on the SAGTOURS tables, type either one of the following:

```
$ @seltabs        (for all tables in succession)

or (for individual tables)

$ ESQSCHEMAID:=SAGTOURS

$ esqmak sel_cont
$ esqrun sel_cont      ! (table CONTRACT)

$ esqmak sel_crui
$ esqrun sel_crui      ! (table CRUISE)

$ esqmak sel_pers
$ esqrun sel_pers      ! (table PERSON)

$ esqmak sel_slr
$ esqrun sel_slr       ! (table SAILOR)

$ esqmak sel_yht
$ esqrun sel_yht       ! (table YACHT)
```

To perform a DECLARE CURSOR operation on the SAGTOURS tables, type either one of the following:

```
$ @crstabs             (for all tables in succession)

or (for individual tables)

$ ESQSCHEMAID:=SAGTOURS

$ esqmak crs_cont
$ esqrun crs_cont      !(table CONTRACT)

$ esqmak crs_crui
$ esqrun crs_crui      !(table CRUISE)

$ esqmak crs_pers
$ esqrun crs_pers      !(table PERSON)

$ esqmak crs_slr
$ esqrun crs_slr       !(table SAILOR)

$ esqmak crs_yht
$ esqrun crs_yht       !(table YACHT)
```

To drop the SAGTOURS tables after having verified the successful installation of BASIC, type the following:

```
$ @drptabs            (for all tables in succession)
```

The SAGTOURS tables can also be used to verify that the BASIC Interactive Facility has been successfully installed. The SAGTOURS tables should not be dropped beforehand.

Verify that the tables have been dropped by using the ADABAS Utility ADAREP to see that the ADABAS files no longer exist.

To start the BASIC Interactive Facility, type the following:

```
$ esqint
```

The system prompt requests user ID and password. In the case, where the security feature is not turned on, press ENTER to bypass this input. If the security feature is turned on enter 'esq' for the user and a RETURN for the password.

```
ESQ User: esq     (string 'esq' should be entered by the user)
Password:         (press carriage return)
```

To display all tables owned by the current user, type the following:

```
esqint: select * from information_schema.tables;
```

To display all columns of table SAILOR, type the following:

```
esqint: select *  from information_schema.columns
                  where table_schema = "SAGTOURS" and table_name = "SAILOR";
```

Installation File Lists

Directory ESQ$MAIN:

LOGIN.COM	Product-specific login procedure
VERSION.DAT	Product Version File

Directory ESQ$VERSION:

Image Files:

ESQBIF.EXE	Basic Interactive Facility
ESQC.EXE	C Precompiler
ESQCOB.EXE	COBOL Precompiler
ESQCGI.EXE	WWW Common Gateway Interface Program
ESQERL.EXE	Precompiler Error Logger Utility
ESQGTD.EXE	Generate SQL Table Descriptions Utility
ESQINI.EXE	Server Initialization Image
ESQKILL.EXE	Server Termination (Kill) Utility
ESQMIG.EXE	Migration Utility
ESQOPR.EXE	Operator Utility
ESQSRV.EXE	Server Thread Image

Shareable Image Files:

ESQLNK142.EXE	Client/Server Interface
ESQTHS142.EXE	Server Thread Image (for ESQSRV)
ESQVOLIB142.EXE	Virtual Operating System (VO)
ESQVOSYS142.EXE	VO Protected Code

User Command Files:

ESQCC.COM	Compile 'C' Application
ESQGEN.COM	Generate Server Environment
ESQINT.COM	Interactive SQL Facility
ESQKILL.COM	Terminate (Kill) Server
ESQLINK.COM	Link ADABAS SQL Server Application
ESQLOG.COM	Show Server Log File
ESQMAK.COM	Make ADABAS SQL Server Application
ESQMEM.COM	Show ADABAS SQL Server Shared Memories
ESQMIGRATE.COM	Start Migration Utility
ESQPC.COM	Precompile ADABAS SQL Server Application
ESQPROC.COM	Show Active ADABAS SQL Server Processes
ESQREMOVE.COM	Remove Server Environment
ESQRUN.COM	Run ADABAS SQL Server Application
ESQSET.COM	Set Default Server
ESQSHOW.COM	Show Server(s)
ESQSTART.COM	Start Server
ESQSTOP.COM	Stop Server
START_ESQSRV.COM	Start Server
SYMBOLS.COM	Define ADABAS SQL Server Symbols
ESQCVT.COM	File Conversation Utility

Internally Used Command Files:

ESQADU.COM	Activate ADABAS Utility
ESQCGM.COM	Check For Global Memory
ESQDBV.COM	Check ADABAS Version
ESQEXE.COM	Activate Server Thread
ESQFDU.COM	Activate ADAFDU Utility
ESQLOD.COM	Activate ADALOD Utility
ESQLOGGER.COM	Write Logs to ESQ$MAIN:ESQ.LOG
ESQSED.COM	String Substitution Utility
ESQSRV.COM	Start Server
ESQTAIL.COM	Dynamically Show Tail Of Textfile

Miscellaneous Files:

ESQCAT.SQL;1	Catalog Meta Data File
ESQCAT1.FDU:1	Input File for ADABAS FDU Utility (FDU/FDT Information)
ESQCAT1.FDUINP;1	Input File for ADABAS FDU Utility (Command Statements)
ESQCAT2.FDU;1	Input File for ADABAS FDU Utility (FDU/FDT Information)
ESQCAT2.FDUINP;1	Input File for ADABAS FDU Utility (Command Statements)
ESQCAT3.FDU;1	Input File for ADABAS FDU Utility (FDU/FDT Information)
ESQCAT3.FDUINP;1	Input File for ADABAS FDU Utility (Command Statements)
ESQERR.CMPINP;1	Input File for ADABAS CMP Utility
ESQERR.DAT;1	Error Messages Data File
ESQERR.FDU,1	Input File for ADABAS FDU Utility (FDU/FDT Information)
ESQERR.FDUINP;1	Input File for ADABAS FDU Utility (Command Statements)
ESQERR.MUPINP;1	Input File for ADABAS MUP Utility
ESQPC.PAR;1	Precompiler Parameter File (Template)
ESQRUN.PAR;1	Run-time (Client) Parameter File (Template)
ESQSRV.PAR;1	Server Parameter File (Template)
ESQ_ROUTING.DAT;1	Server Routing File (Template)
ESQLINK.OPT	Linker Options File for ADABAS SQL Server applications
ESQ_PL.DAT	Patch Lever File (internal use).

2

OPERATING THE ADABAS SQL SERVER

This chapter contains information pertaining to the operation and administration of the ADABAS SQL Server, the ADABAS SQL Utilities and user applications which communicate with the ADABAS DBMS using ANSI/ISO SQL (Structured Query Language) in an OpenVMS environment.

Overview of ADABAS SQL Server Commands

This is a brief overview of the server commands as defined for the OpenVMS DCL command line interface. A detailed description of each command can be found in the this manual, depending on the context where they apply.

Note:
Before any of the following commands can be accessed, ESQ$MAIN:LOGIN.COM must have been executed.

Generate one or more Server Environments

Generate (common/local) server environments:

```
$ esqgen   servername dbid catalog_base_file error_file [desc] [sec] [common]
```

Starting the Utilities:

Start BASIC Interactive SQL:

```
$ esqint [[servername] [communication destination]]
```

Start the Migration Utility:

```
$ esqmigrate  server_name dir_fnr  [[-v] [-t] [-n] [-s] [-h]]
```

Start the Generate Table Description Utility:

Command line operation mode:

```
$ define/user sys$output output_file
$ esqgtd  [option]  database_name  database_number  file_number  [ddm_name]
```

Input file operation mode:

```
$ define/user sys$output output_file
$ define/user sys$input input_file
$ esqgtd
```

Creating and Executing an Application

Precompile a module:

```
$ esqpc    module [/list] [/nowarnings]
```

Compile a module:

```
$ esqcc    module [compiler_options]
```

Link a multi-module application:

```
$ esqlink  module [module_1 ... module_x]
```

Create a single-module application:

```
$ esqmak   module [precompiler_options] [compiler_options] [link_option]
```

Run an application:

```
$ esqrun   module
```

Driving an ADABAS SQL Server:

Set a default server:

```
$ esqset   servername
```

Verify the default server / all known servers:

```
$ esqshow  [/all]
```

Start a server:

```
$ esqstart [servername] [user_id]
```

Terminate a server

```
$ esqopr [servername]
esqopr: shutdown
```

```
$ esqstop  [servername]
```

```
$ esqopr [servername]
esqopr: abort
```

```
$ esqkill  servername
```

Start the Operator Utility:

```
$ esqopr   [servername]
```

Display the server log file

```
$ esqlog   [servername] [/tail]
```

Display shared memories used by the ADABAS SQL Server:

```
$ esqmem
```

Display error text:

```
$ esqerr error_number
```

Remove a server environment:

```
$ esqremove servername [/all]
```

Starts the tool which converts SYSTRANS files into a format acceptable to the GTD Utility:

```
$ esqcon systrans_file_name [/del]
```

Creating and Executing an Application

An application program is prepared for execution in a 3-step procedure (precompile, compile and link).

For single-module SQL applications the ADABAS SQL Server provides a command for these 3 steps. To execute this command enter the following:

```
$ esqset  servername
$ esqmak  prog [precompiler_options] [compiler_options] [linker_options]
```

For multi-module applications or if the steps are to be executed separately, the procedures – as described in the following sections – apply.

Precompile

The ADABAS SQL Server Precompiler currently supports the 3rd generation host languages C and COBOL. The respective precompilers and commands are found in the directory defined by the logical name ESQ$VERSION, which was created during the installation of the ADABAS SQL Server.

If the application program design and coding are complete, the source statements are ready to be prepared for execution. Before a program can be executed, it must be compiled by the respective host language compiler. Before this compilation, however, the SQL statements embedded in the 3rd generation host language must first be prepared by the ADABAS SQL Server Precompiler for compilation as host language statements.

The ADABAS SQL Server Precompiler scans every statement of the program source and produces a modified program in which every SQL statement has been replaced by host language statements such as variable definitions and calls to the ADABAS SQL Server. Make sure that the ADABAS nucleus containing the SQL catalog and error files is active.

Before the precompiler can be started, the current server has to be set by entering the following:

```
$ esqset  servername
```

The precompiler is started by entering the following:

```
$ esqpc  modulename  [/list]  [/nowarnings]  [/keep]
```

The optional parameter /list generates a precompiler listing file and the optional parameter /nowarnings suppresses precompiler warnings. The optional parameter /keep results in the output file ESQ$OUTPUT being kept inspite of errors.

The precompiler searches in the current working directory for a precompiler source file according to the file extension convention shown in the following table, and then starts a language-dependent precompilation. For example, if the precompiler finds the source file with the name *modulename*.pc first, the host language precompiler for the language C is started.

The following table shows the file extension convention for precompiler source files:

Programming Language	Precompiler Input File	Precompiler Output File	Precompiler Listing
C	*modulename*.**CPC** or *modulename*.**PC**	*modulename*.**C**	*modulename*.**PCLIS**
COBOL	*modulename*.**COBPC** or *modulename*.**PCOB**	*modulename*.**COB**	*modulename*.**PCLIS**

The symbol ESQSCHEMAID, if set, determines the default schema ID for the application module. The default value for ESQSCHEMAID is the user ID.

The symbol ESQLIBRARY, if set, determines the library name under which the application module's meta programs are stored in the SQL catalog at runtime. The default value for ESQLIBRARY is the user ID. Setting this variable allows distinction between different sets of applications.

By default, the precompilers uses the server-specific parameter file ESQPC.PAR from the server directory ESQ$SRVDIR. If entry changes in the parameter file are necessary, there are two recommended procedures:

For permanent user- and server-specific changes enter:

```
$ copy esq$srvdir:esqpc.par esq$srvdir:esqpc.par_uid
$ edit esq$srvdir:esqpc.par_uid
```

where *uid* is the OpenVMS user name.

For temporary working-directory-specific changes enter:

```
$ copy esq$srvdir:esqpc.par *.*
$ edit esqpc.par
```

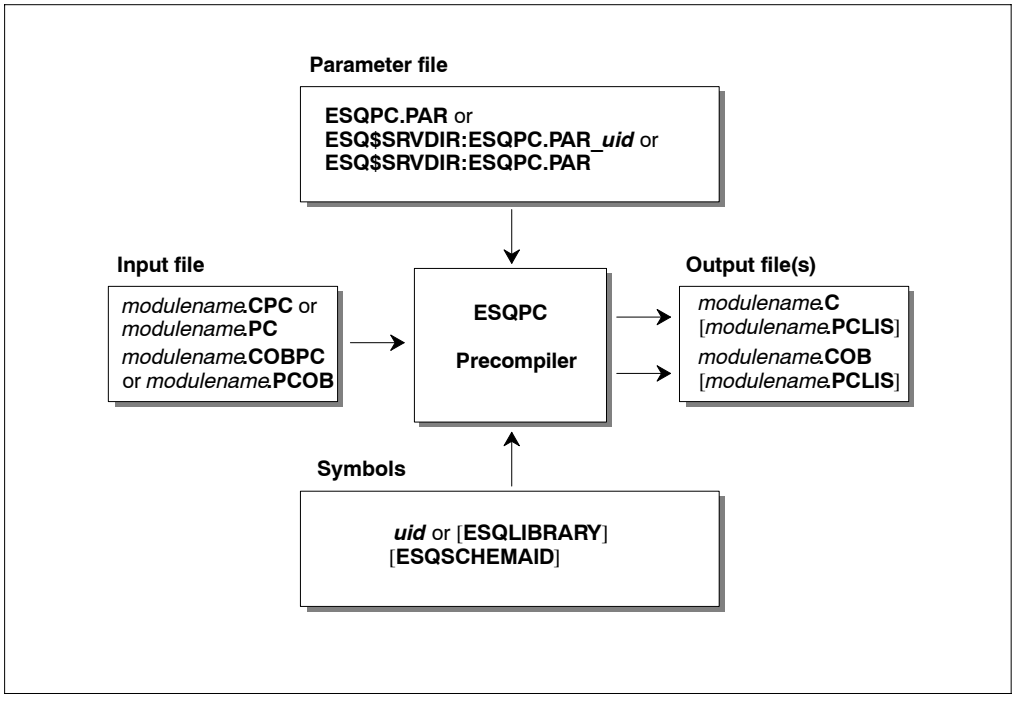


Figure 2-1: Precompilation data flow chart for an SQL application module

Compile

After precompilation, the application program is compiled using the standard compilation procedure suitable for the host language. For SQL applications the ADABAS SQL Server provides a compilation command. To start the compilation process enter the following:

```
$ esqcc  modulename  [compiler_options]
```

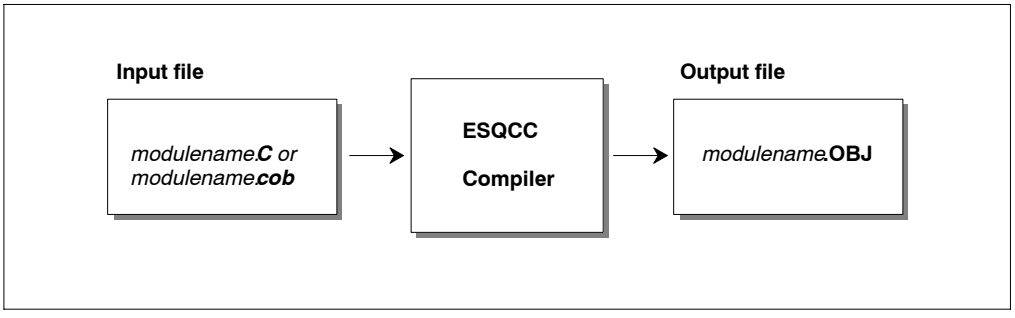


Figure 2-2: Compilation data flow chart for an SQL application module

Note that the ADABAS SQL Server works with the D_FLOAT data format. This is the C-Compiler default under OpenVMS for VAX.

Under OpenVMS for Alpha AXP, a C−application is compiled with /float=d to ensure that the D_FLOAT data format is presented to the ADABAS SQL Server.

Under OpenVMS for either VAX or Alpha AXP, a DEC Cobol−application is compiled with the ANSI_FORMAT qualifier.

Link

Linking is performed after compilation. The logical name ESQLNK must be set at link time pointing to the shared image ESQLNK142.EXE which will be linked to the application. To start the linking process enter the following:

```
$ esqlink prog [,module_1 ... ,module_x] [linker_options]
```

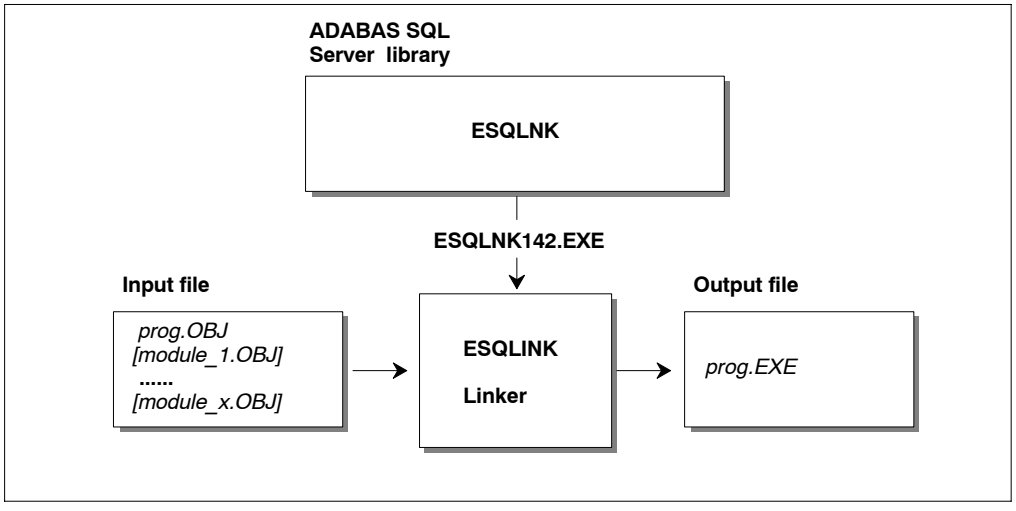


Figure 2-3: Link data flow chart for an SQL multi−module application

Execute

Before an ADABAS SQL Server application can be executed, the default server must have been set via the following command:

```
$ esqset  servername
```

To re-direct input for or output from the application, the logical names ESQ$INPUT and ESQ$OUTPUT may be defined before entering the ESQRUN command. The modification of SYS$INPUT and SYS$OUTPUT may result in unpredictable conditions.

To run the application enter the following:

```
$ esqrun  prog  [command arguments]
```

The optional command arguments are passed on directly to the application.

The symbol ESQSILENT, if set, prevents log lines to generated by the ESQRUN command.

The symbol ESQSCHEMAID, if set, determines the default SQL schema ID for the application module. For details, refer to Appendix B of this manual: **The Parameter Processing Language**, section: **Global Default Schema Identifier Setting**.

By default, the runtime system uses the server specific parameter file ESQRUN.PAR from the server directory ESQ$SRVDIR. If any changes in the parameter file are necessary, there are two recommended procedures:

For permanent user- and server-specific changes, enter:

```
$ copy esq$srvdir:esqrun.par esq$srvdir:esqrun.par_uid
$ edit esq$srvdir:esqrun.par_uid
```

where *uid* is the OpenVMS user name.

For temporary working-directory-specific changes, enter:

```
$ copy esq$srvdir:esqrun.par *.*
$ edit esqrun.par
```

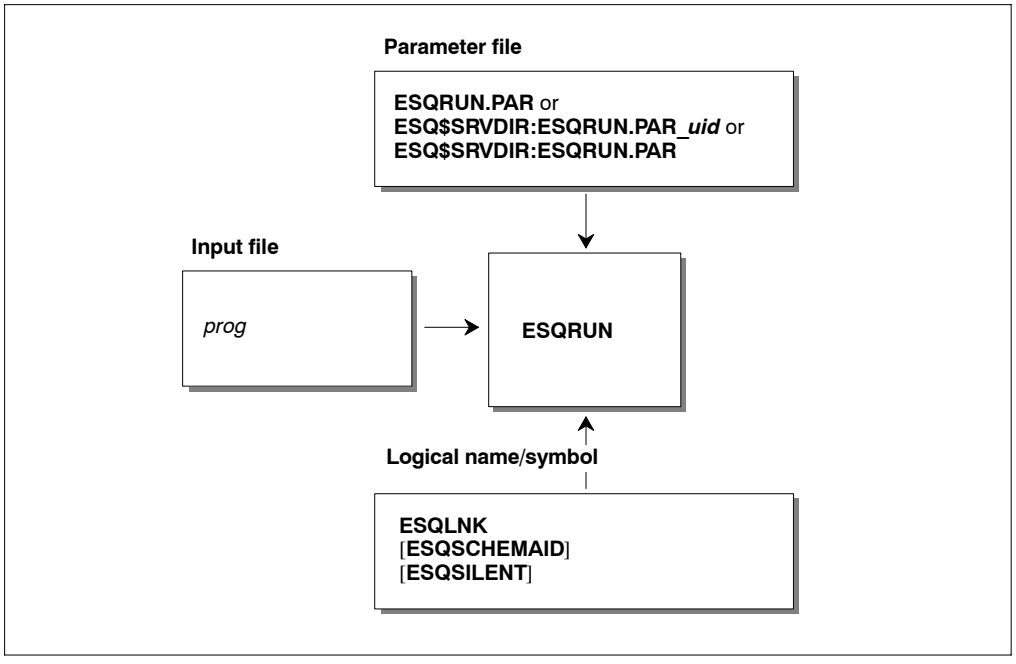


Figure 2-4: Runtime data flow chart

Debug

To build an application in debug mode, there are two different procedures depending on whether the application is being a single or multi-module application.

For single-module applications, enter:

```
$ esqmak      prog  "" /debug /debug
```

For multi-module applications, enter:

```
$ esqpc       prog
$ esqcc       prog     /debug
$ esqpc       module_1
$ esqcc       module_1 /debug
......
$ esqpc       module_x
$ esqcc       module_x /debug
$ esqlink     prog,    module_1,... module_x /debug
```

To start the application, enter:

```
$ esqdebug :=/debug
$ esqrun  prog
```

The result will be that a debugger session will be started for your application.

Driving an ADABAS SQL Server

The ADABAS SQL Server provides a set of commands and an Operator Utility to operate and maintain servers. A server environment is generated via the ESQGEN command, as described in the chapter **Installing the ADABAS SQL Server System**. The server environment is mainly characterized by the ADABAS file which holds the SQL catalog for that server. The server itself, however, is referenced by its name only.

Set the Default Server

The command that sets the current default server within a user session is:

```
$ esqset  servername
```

Server names are generally treated in uppercase. If a lowercase server name must be specified, for example, when refering to a remote server on a UNIX system, the server name must be enclosed in double quotes.

Any following statement (including precompiler and SQL application execution) will take this server as default server if no other server name is specified.

Verify the Default Server

The command to verify the current default server settings is:

```
$ esqshow [/all]
```

The /all option shows also all known ADABAS SQL Servers on the current node.

Start a Server

A server process has to be started, if you want to perform one of the following computing methods:

- local (client and server on same node) or
- remote (client and server on different nodes) client/server computing.

Server-specific environments may be defined in a server-specific startup procedure (ESQ$SRVDIR:STARTUP.COM) which will - if present - be executed automatically before server startup (controlled by the ESQSTART command). The procedure may contain, for example, the setting of logging variables to enable logging during the time the server is active.

Before the server can be started, the client/server communication interface (CSCI) has to be activated. Refer to the Appendix **The ADABAS SQL Server and other Software AG Products** in the manual *ADABAS SQL Server Programmer's Guide* or the *ENTIRE BROKER: CSCI Manual* for details.

For remote client/server computing NET-WORK for OpenVMS must also be active and parameterized properly on both sides. Refer to the chapter **Client/Server Topics** in the *ADABAS SQL Server Programmer's Guide.*

To start the ADABAS SQL Server, use the following command:

```
$ esqstart [servername][user-id]
```

The optional specification user_id is the identification of the user under which the server is to be started as a detached VMS process. The default user_id is the identification of the user entering the ESQSTART command.

The detached VMS process needs write access to the server-specific directory ESQ$SRVDIR for its log file. If this write access has not been granted for the specific user, this VMS process can not even be initiated. Note that in this case the server log file of the most recent server start will be displayed.

Note:
If the server is not active after the execution of the ESQSTART command and the log file displayed is that of the recent server start, it is most likely that no write access has been granted for that specific user. To verify this condition, use the OpenVMS Accounting Utility.

Note:
For information regarding quotas and privileges, refer to the section ***Quotas and Privileges for the ADABAS SQL Server*** *in chapter* ***Installing the ADABAS SQL Server System*** *earlier in this manual.*

Note:
The login command procedure of the user user_id must not modify its process name if in detached mode.

The server processes are started using the parameter file ESQ$SRVDIR:ESQSRV.PAR, which among others defines the number of thread processes to be started for this server.

Note:
One server thread can handle only one client session at a time, so the number of threads determines the maximum number of concurrent client sessions at a given time.

The ESQSTART command prints out the beginning of the server's log file on the requesting terminal after a few seconds so that you can verify if the server startup was performed successfully.

Terminate a Server

There are 4 ways to terminate an ADABAS SQL Server ranging from a normal shutdown to a fast kill.

- **SHUTDOWN**

  The normal server shutdown is done via the ADABAS SQL Server Operator Utility (ESQOPR) with the SHUTDOWN command:

  ```
  $ esqopr [servername]
  esqopr: shutdown
  ```

- **STOP**

  An equivalent command but without being prompted for confirmation and shutdown reasons is the ESQSTOP command:

  ```
  $ esqstop [servername]
  ```

  After execution of these two commands any new client sessions will be rejected by the server. All server threads that currently have no active client session will shut down within one minute. The other server threads remain active until the clients disconnect from the server or the session being timed-out by the server. Only if there are no more active sessions will the server shut down completely.

  As this method may take some time, it is advisable to use the ADABAS SQL Server Operator Utility to watch the server activities (active session, requests etc.)

- **ABORT**

  The next degree in stopping a server is done via the ADABAS SQL Server Operator Utility with the ABORT command.

  ```
  $ esqopr [servername]
  esqopr: abort
  ```

  After execution of this command, any new client sessions and requests will be rejected by the server. All server threads continue with their current client requests if any are active, and will shutdown within one minute regardless of any active client sessions, i.e. the sessions are terminated.

- **KILL**

  This command terminates the server at once, regardless of any active client sessions or requests. Nevertheless, a proper clean-up will be performed.

  ```
  $ esqkill servername
  ```

Display the Server Log File

To display the server log file at any time enter the following command:

```
$ esqlog [servername] [/tail]
```

If only the latest incoming log lines are to be displayed dynamically use the command with the /tail-option. Control-C has to be pressed to terminate the display.

There are also logical names for the current server log file (ESQ$SRVLOG) and for the log file of the previous server startup (ESQ$SRVOLDLOG), which can be used in conjunction with any DCL commands, e.g.

```
$ edit/read ESQ$SRVLOG
```

When an ADABAS SQL Server is started, a log file will be created in the directory pointed to by the logical ESQ$SRVDIR. Each time the server is started, a new log file will be created. For the ADABAS SQL Server to create a new version of the log file, either the server must be started under the same user name as the ownership on the previous log file OR the user name starting the server must have the BYPASS default privileges.

Display Shared Memories used by the ADABAS SQL Server:

For reasons described in the original OpenVMS Install Utility Manual, shared memory is used for the ADABAS SQL Server. The information displayed as a result of the ESQMEM command is the output of the OpenVMS Install Utility with the /full option having the same effect. To display the global sections installed for the ADABAS SQL Server enter the following command:

```
$ esqmem [/full]
```

Display Error Text:

The following command displays the corresponding error text to the specified *error number*. The error text is documented in the file ESQ$VERSION:ESQERR.DAT.

```
$ esqerr error_number
```

Remove a Server Environment

The ESQREMOVE command removes an existing server environment which was generated using the ESQGEN command. Default functionality removes the server-specific files from the operating system directory. If the option /all is specified, additionally, the catalog file and the error file in ADABAS will be removed.

```
$ esqremove servername [/all]
```

The ADABAS SQL Server Operator Utility

The ADABAS SQL Server Operator Utility provides the functionality to obtain actual and statistical information about a specific ADABAS SQL Server running on the local node, and commands to terminate a server. These last commds are described in detail under the section: Terminating a Server, a few pages before.

Start the Utility

The utility is invoked by typing:

```
$ esqopr [servername]
```

The following sample sessions shows the usage and functionality of the ADABAS SQL Server Operator Utility. The commands entered by the user are printed in bold.

Example:

```
$ esqopr SAGTOURS
$ESQOPR-I-STARTED,  23-JUN-1996 15:17:28, Version 1.4/2, (VAX/OpenVMS)
$ESQOPR-I-ONLINE,   Server SAGTOURS attached online on node ENT
esqopr: help

ESQ operator utility online help display

 help      - Online help display (this display)
 dis=act   - Display active servers on local node
 dis=scb   - Display Server Control Block (SCB)
 rep=n     - Repeat display every n seconds (VTxxx only)
 server=SS - Attach to server named SS
 .         - Redisplay most recent display
 shutdown  - Shutdown server (wait for disconnects)
 abort     - Abort server    (disconnect sessions)
 quit      - Leave esqopr

esqopr: dis=scb

ESQ Server Control Block (SCB) Information

 Server name        = SAGTOURS on node ALF1
 Startup time       = 19-AUG-1996 16:45:20
 Version            = 1.4/2
 Server type        = CSCI
 Catalog Files      = DB 101, Files 30-32
 Thread processes   = 5
 Sessions / Thread  = 1
 Last active thread = none
```

```
Server Users:          Client/Server    Linked-in  Precompiler        Total

 Active sessions    =             0            0            0            0
 Active requests    =             0            0            0            0
 Active threads     =             5            0            0            5
 Total sessions     =             0            0            0            0
 Total requests     =             0            0            0            0

19-AUG 17:03:36.68 Server SAGTOURS Thread 5 pid 559940102 up and running


esqopr: dis=act

Active ADABAS SQL servers on node ALF1:

   Name     Catalog     Last Activity    act.ses  act.req  tot.ses  tot.req

 SAGTOURS   101/030   19-AUG-1996 16:45        0        0        0        0
 TEST       101/040   19-AUG-1996 12:28        3        1      193     3456
 NIST       101/050   19-AUG-1996 17:01        2        2     2001    40561

esqopr: shutdown
Really SHUTDOWN server SAGTOURS ? y
Enter reason for shutdown: System Maintenance
%ESQOPR-I-SHUTDWN, Server shutdown initiated at 19-AUG-1996 17:03:40.53
%ESQOPR-I-DETACH,  Server SAGTOURS now detached
esqopr: quit
%ESQOPR-I-TERMINATED, 19-AUG-1996 17:04:54
```

Parameter Processing

The execution of the 3 major components of the ADABAS SQL Server (Server, Precompiler, and Runtime System) can be influenced by parameter settings via PPL (Parameter Processing Language).

When generating a server environment, the following three parameter files are created in the server-specific directory $ESQSRVDIR:

- **ESQ$SRVDIR:ESQSRV.PAR (server parameter file)**

  This parameter file is read in at server startup time. For example, one purpose could be to adjust the number of threads started by the server, depending on the number of concurrent sessions.

  Among other things, it defines parameters, such as: server name, server catalog DBID/FNR, communication type (ENTIRE CSCI, ENTIRE BROKER), number of server threads, default client parameters, request/reply buffer lengths, optional: server trace/log control statements.

- **ESQ$SRVDIR:ESQPC.PAR (precompiler parameter file)**

  This parameter file is read in at precompiler startup time.

  Among others, it defines parameters, such as: default schema identifier, maximal number of compile errors, server catalog DBID/FNR (for single-user linked-in mode), optional: precompiler trace/log control statements.

- **ESQ$SRVDIR:ESQRUN.PAR (client parameter file)**

  This optional parameter file is read in at client application startup time.

  Among other things, it defines parameters, such as: default schema identifier, server session timeout value, maximum number of cursors, server catalog DBID/FNR (for single-user linked-in mode), optional: client trace/log control statements.

These files can be customized to meet server-specific needs.

Parameter Processing and Global Symbols

The standard parameter files contain various values which reference global symbols. The various commands, e.g., ESQRUN attempt to find these references in order to replace them with the values contained in the global symbols. This is achieved via a simple text substitution. Should the original references have been previously hard-coded, no substitution can take place and, therefore, the values of the global symbols are effectively ignored.

The following values are subject to substitution:

- ESQDBID
- ESQCATF
- ESQERRF
- ESQSCHEMAID
- ESQLIBRARY
- ESQPROG

Example of Substitution:

If the current parameter file contains the following line:

```
DEFAULT SCHEMA IDENTIFIER = "$ESQSCHEMAID"
```

and the global symbol ESQSCHEMAID has been set to 'RAINER', then the command ESQRUN will perform the substitution resulting in the value 'RAINER' being the default schema identifier.

Example of Non-substitution:

If the current parameter file contains the following line:

```
DEFAULT SCHEMA IDENTIFIER = PETER
```

regardless of the value of the global symbol ESQSCHEMAID, the command ESQRUN will not perform any substitution and the resulting value for the default schema identifier will be 'PETER'.

For additional information refer to **Appendix B: The Parameter Processing Language (PPL)** later in this manual.

User Exits

Overview

A user exit is a user-written routine that enables the user to participate in the processing performed by the ADABAS SQL Server. The user-written routine is dynamically loaded at the startup of the server or application, and is called at predefined stages in the processing of either.

The routines may be written in any programming language that conforms to the OpenVMS calling standard.

The following user exits are available:

User Exit	Use
Client User Exit 1	user processing on an ESQLNK call before it is processed by the server.
Client User Exit 2	user processing on an ESQLNK call after it is processed by the server.
Server User Exit 5	user processing on an ADABAS call. The function is called before the ADABAS call is executed.
Server User Exit 6	user processing on an ADABAS call. The function is called after the ADABAS call is executed.

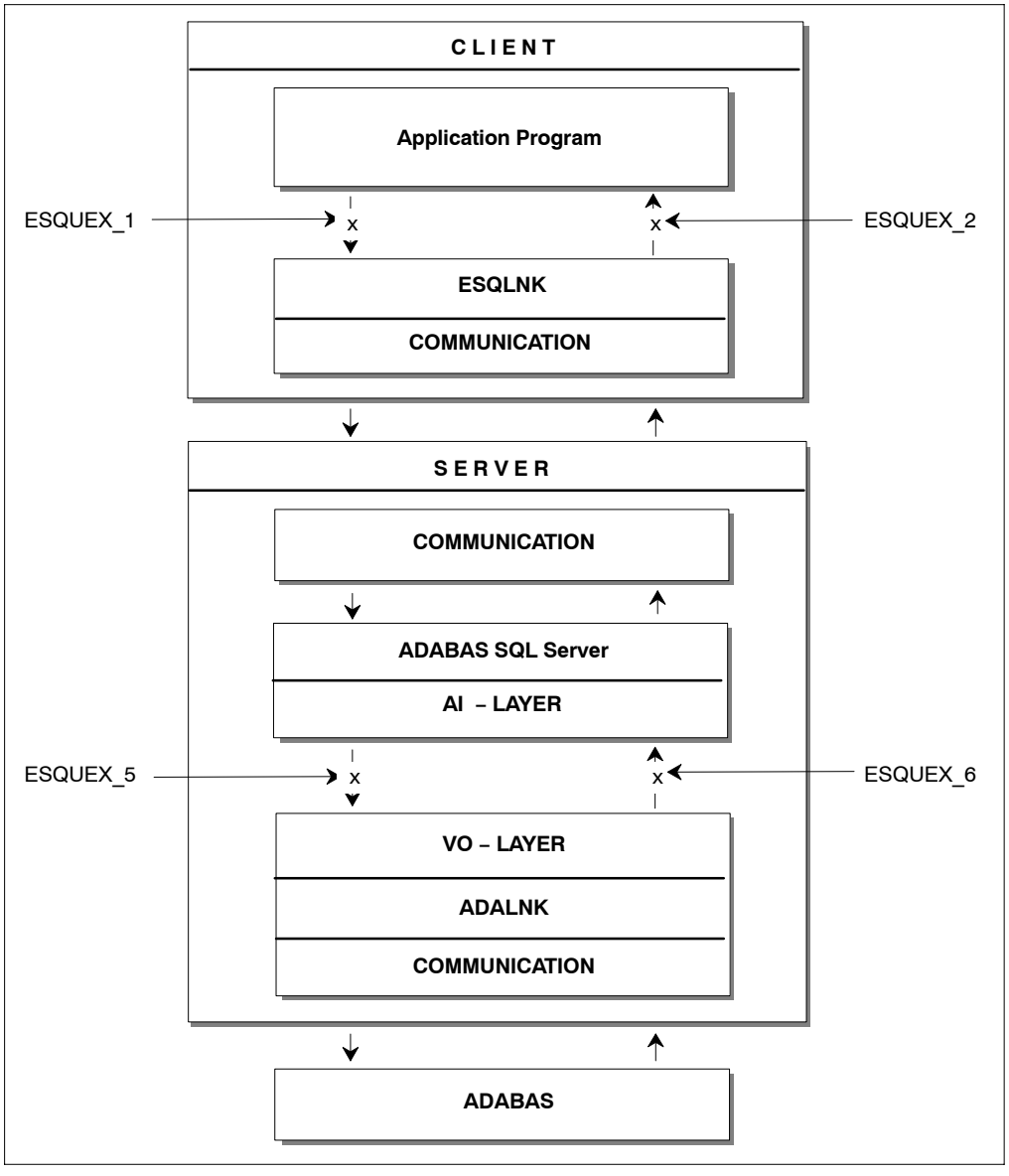


Figure 2-5: Locations of user exits within the ADABAS SQL Server

Client User Exit 1

Description

ADABAS SQL Server User Exit 1 is a user exit that performs user processing on an ESQLNK call. The routine is called when the processing of a statement begins.

Input Parameters

```
Format: UEX_1 ( )
```

Client User Exit 2

Description

ADABAS SQL Server User Exit 2 performs user processing on an ESQLNK call. The routine is called when the processing of a statement ends. The input parameter that is specified enables the user exit to change the response code of the ESQLNK call.

Input Parameters

```
Format: UEX_2 (SQLCA)
```

SQLCA	usage:	SQLCA
	type:	SQLCA (refer to esqca.h)
	access:	read/write
	mechanism:	by reference

Server User Exit 5

Description

ADABAS SQL Server User Exit 5 is a function that performs user processing on an ADABAS call. The user exit is called when the processing of an SQL request causes an ADABAS call and before this ADABAS call is executed.

Input Parameters

Format: UEX_5 (CB, p_acc, FB, RB, SB, VB, IB)

CB
usage: CB
type: unsigned char *
access: read/write
mechanism: by reference

ESQACC
usage: p_acc
type: ESQACC (refer to esqacc.h)
access: read/write_
mechanism: by reference

FB
usage: FB
type: unsigned char *
access: read/write
mechanism: by reference

RB
usage: RB
type: unsigned char *
access: read/write
mechanism: by reference

SB
usage: SB
type: unsigned char *
access: read/write
mechanism: by reference

VB	usage:	VB
	type:	unsigned char *
	access:	read/write
	mechanism:	by reference

IB	usage:	IB
	type:	unsigned char *
	access:	read/write
	mechanism:	by reference

Server User Exit 6

Description

ADABAS SQL Server User Exit 6 is a function that performs user processing on an ADABAS call. The user exit is called after an ADABAS call has been executed.

Input Parameters

Format: UEX_6 (CB, p_acc, FB, RB, SB, VB, IB)

SQLCA	usage:	FB
	type:	unsigned char
	access:	read/write
	mechanism:	by reference
CB	usage:	CB
	type:	unsigned char *
	access:	read/write
	mechanism:	by reference
ESQACC	usage:	p_acc
	type:	ESQACC (refer to esqacc.h)
	access:	read/write_
	mechanism:	by reference
FB	usage:	FB
	type:	unsigned char *
	access:	read/write
	mechanism:	by reference
RB	usage:	RB
	type:	unsigned char *
	access:	read/write
	mechanism:	by reference

SB	usage:	SB
	type:	unsigned char *
	access:	read/write
	mechanism:	by reference
VB	usage:	VB
	type:	unsigned char *
	access:	read/write
	mechanism:	by reference
IB	usage:	IB
	type:	unsigned char *
	access:	read/write
	mechanism:	by reference

Common Rules for Server User Exits 1 and 2

Parameters

p_sqlca	provides access to the SQLCA
p_inp	provides reads access to the input data (user ID, password, etc.)
p_out	provides write acesss to the output data (user ID, password, etc.)

Examples

The user exit calls the function my_security() for a CONNECT statement. If this function fails, value −6701 is returned via sqlcode field of SQLCA. The parameters of the function my_security() are user-ID and password. This information is passed on to the user exit via the parameter struct esq_uex1_inp.

```
#include <stdio.h>
#include <esqacc.h>
#include <esqerr.h>
#include <esquex1.h>
void uex_1(
     struct sqlca * p_sqlca,
     struct esq_uex1_inp * p_inp,
     struct esq_uex1_out * p_out )
{

     if (p_inp->sql_command == ESQ_CONNECT)
     {
          if (my_security(
               p_inp->l_name,    p_inp->c_name,
               p_inp->l_pwd,     p_inp->c_pwd     ))
            p_sqlca->sqlcode = -6701;
     }
     return;
}
```

The following routine changes the user ID to JOHN for a CONNECT statement.

```
#include <stdio.h>
#include <esqacc.h>
#include <esqerr.h>
#include <esquex1.h>

void uex_1(
     struct sqlca * p_sqlca,
     struct esq_uex1_inp * p_inp,
     struct esq_uex1_out * p_out )


{
     if (p_inp->sql_command == ESQ_CONNECT)
     {
          p_out->1_name = strlen( ”JOHN” );
          strcpy( p_out->c_name, ”JOHN” );
     }
     return;
}
```

Common Rules for Server User Exits 5 and 6

Parameters

The following explains the parameters of the function's prototype for the ADABAS SQL Server User Exit 5 and 6:

- CB provides access to the ADABAS control block, as specified in the ADABAS Command Reference Manual.
- ESQACC provides access to the ADABAS SQL Server session context.
- FB provides access to the ADABAS format buffer.
- RB provides access to the ADABAS record buffer.
- SB provides access to the ADABAS search buffer.
- VB provides access to the ADABAS value buffer.
- IB provides access to the ADABAS ISN buffer.

Note:
Ensure that the user exit is prepared to handle NULL pointer values for some of the parameters.

ADABAS SQL Server Session Contexts

The ADABAS SQL Server session context is provided in the include file <esqacc.h>, and is defined as follows:

```
#define  ESQRTS_CONTEXT     (-1)
#define  L_CLIENTS_NAME     (18)
#define  L_CLIENTS_PWD      (32)
#define  L_CLIENTS_NODE     (8)

typedef struct esqacc
 {
  long    c_sid;    /* session id           */
  long    c_con;    /* current context      */
  unsigned char c_node [ L_CLIENTS_NODE ];
  unsigned char c_name [ L_CLIENTS_NAME ];
  unsigned char c_pwd  [ L_CLIENTS_PWD ];
 } ESQACC;
```

As stated in the *ADABAS SQL Server Programmer's Guide*, **Appendix D**, one active client may occupy up to two ADABAS session contexts (user queue elements). To enable distinction between these contexts, the following rules have been established:

RULE_0:	c_pwd will be set to binary zeroes and is, therefore, not visible.
RULE_1:	c_sid is the session identifier for the currently active client session. The value is assigned by the ADABAS SQL Server.
RULE_2:	c_con is the currently active operation context for the current ADABAS call.
RULE_3:	If c_con is equal to c_sid, then the current ADABAS call is executed on behalf of the client, for example, to retrieve data.
RULE_4:	If c_con is equal to ESQRTS_CONTEXT, then the current ADABAS call is executed on behalf of the ADABAS SQL Server, for example, to store a meta program.

Embedding Server User Exits 5 and 6

The user exit processing has been realized within the ADABAS SQL Server as follows (pseudo language example):

```
.
.
if (ESQUEX5 defined)
     BEGIN
     rc = ESQUEX5 (CB,ESQACC,FB,RB,SB,VB,IB);
     if (rc <> 0)
          BEGIN
          CB -> response_code = rc;
          goto do_not_call_ADABAS;
          END
     END

ADABAS (CB, ......);

if (ESQUEX6 defined)
     BEGIN
     rc = ESQUEX6 (CB,ESQACC,FB,RB,SB,VB,IB);
     if (rc <> 0)
          BEGIN
          CB -> response_code = rc;
          END
     END
do_not_call_ADABAS:
.
.
```

User Exist Response Codes

Both user exits are specified to return an integer value via the ‘return’ statement. The returned value is interpreted within the ADABAS SQL Server as an ADABAS response code.

User Exit 5

- If User Exit 5 returns the value zero, the current ADABAS call is executed.
- If the returned value is non-zero, the value is placed in the provided ADABAS control block and the ADABAS call is not executed.

User Exit 6

- If User Exit 6 returns the value zero, the processing continues as if no user exit was called.
- If the returned value is non-zero, the value is placed in the provided ADABAS control block as if ADABAS had returned this value.

Based on the above, any returned non-zero value may be visible to the ADABAS SQL Server client (application program). It is, therefore, advisable to use appropriate values, such as reponse code 200, to signal security violations. Refer to the *ADABAS MESSAGES AND CODES Manual* for further response codes.

The value passed on must be within the following range: 0 <= value <= 255

Note:
Some original ADABAS response codes trigger special exception handling routines. For the original ADABAS response code 9, an ADABAS SQL Server internal ‘reopen’ logic for the ESQRTS context is started.

Creation and Definition

User exits can be written in any high-level language that conforms to the OpenVMS calling standard. The directory ESQ$EXAMPLES contains examples of user exits written in C.

Create and define your user exit by executing the following steps:

1 Write a user exit in any high-level language that conforms to the OpenVMS calling standard. The exits should be written with default function names. The convention is: “uex_” followed by the number of the user exit.

2 Compile the source file of the user exit.

For example, to compile the example supplied for User Exit 1 type:

```
$ CC esquex1.c
```

3 Link the user exit.

For VAX platforms, to link the example supplied for User Exit 1 type:

```
$ LINK esquex1, SYS$INPUT/OPTIONS/SHARE
UNIVERSAL=UEX_1
<CTRL/Z>
```

For Alpha AXP platforms, to link the example supplied for User Exit 1 type:

```
$ LINK esquex1, SYS$INPUT/OPTIONS/SHARE
SYMBOL_VECTOR=UEX_1
<CTRL/Z>
```

The /SHARE command qualifier must be specified with LINK since the UNIVERSAL/ SYMBOL_VECTOR option may only be specified in the creation of a shareable image. The UNIVERSAL option directs the linker to include the user exit’s entry point name into the image’s global symbol directory. This entry is required by the ADABAS SQL Server when activating the user exit.

The /SHARE command qualifier must also be specified with LINK if references to relocatable data have to be resolved by the image activator when the ADABAS SQL Server activates this image.

4 Make the user exit available to the ADABAS SQL Server by the following definition:

```
$ DEFINE ESQUEX_number filespec
```

where;
number is the user exit number,
filespec specifies the name of the image to be loaded (the file type defaults to “.EXE“).

User exits are dynamically loaded at the startup of the ADABAS SQL Server application.

Sort Buffer Size Setting

Using the ORDER BY or the GROUP BY clause may sometimes force the ADABAS SQL Server to perform internal sorting of data.

If the default size (16 KB) of the internal buffer which holds the data to be sorted is exceeded, the overflow will be written to a temporary file. To balance this behavior, the buffer size can be set externally via the symbol ESQSRTBUF.

A detailed analysis of the resource usage during the sort can be obtained by setting the global symbol ESQLG to ‘sort’. An example with the use of the default buffer size (16KB) can be found in the chapter **Logging Facilities**, section **Sort Logging** in the *ADABAS SQL Server Programmer’s Guide.*

Exceptions/Exit Handling

Handling Exceptions

Using the ADABAS SQL Server, special attention must be paid when making use of OpenVMS exception handling.

General Exception Handling

Parts of the ADABAS SQL Server (ESQTHS, ADALNK, CPI, etc) handle exceptions by use of the functions:

```
LIB$ESTABLISH()   (Assembler) or
VAXC$ESTABLISH()  (C)
```

The ADABAS SQL Server temporarily establishes a condition handler to run clean-up routines to avoid inconsistencies in the database(s). These condition handlers do not have to be coded in your application anymore.

Explicitly Handled Exceptions

In contrast to the general exception handling as explained above, the signal() function is used to catch the SIGINT (CTRL-C) exceptions for the Linked-in mode. This constitutes the exception handling concept of the UNIX implementation of the ADABAS SQL Server.

This exception is temporarily delayed and subsequently re-raised using an internal condition handling function. A delay will be initiated in two explicit cases to avoid inconsistencies:

- if an ADABAS utility was spawned to execute ADABAS SQL Server DDL statements.
- if an ADABAS call was performed by the ADABAS SQL Server.

The previously marked signal handling routines for these exceptions are re-established after return.

An application which establishes its own condition handler implicitly forces this self-defined condition handler to be active even inside ADABAS SQL Server. In the following examples, you will see how to use existing condition handlers in your application.

How to Use the VAXC$ESTABLISH() Function

If your application is designed to run in OpenVMS environments only, this function is the preferred function to be used. The following example shows how to use VAXC$ESTABLISH in your ADABAS SQL Server application.

```
#include <ssdef.h>
int sig_handler( )
{
  printf("my sig-handler ...\n");
  /* do whatever needs to be done                                */
  return( SS$_RESIGNAL );
}
main()
{
  int a;
  /* ------------------------------------------------------------ */
  /* establish 'sig_handler'                                      */
  /* ------------------------------------------------------------ */

  LIB$ESTABLISH( sig_handler );

  /* ------------------------------------------------------------ */
  /* execute CONNECT                                              */
  /* ------------------------------------------------------------ */

  exec sql connect to default;

  /* do whatever needs to be done                                */

  /* ------------------------------------------------------------ */
  /* force exception                                              */
  /* ------------------------------------------------------------ */
  a=a/0;
  /* do whatever needs to be done                                */
}
```

In the above example, only the exception handler sig_handler is executed because the exception is generated by the application. If any exception during the CONNECT statement occurred, first the ADABAS SQL Server exception handler would be executed and then sig_handler.

How to Use the signal() Function

If your application is to be run in environments other than OpenVMS as well, the signal() function is the exception handling function to be used. The following example shows how to use the signal() function in your ADABAS SQL Server application.

```
#include <signal.h>
#include <ssdef.h>
void (*prev)() = SIG_DFL;
void sig_handler( int sig )
{
  printf("my sig-handler ... \n");
  /* do whatever needs to be done                                */
  if (prev != SIG_DFL) (*prev)(sig);
  exit(-1);
}
main()
{
  int a;
  /* ------------------------------------------------------------ */
  /* establish 'sig_handler'                                      */
  /* ------------------------------------------------------------ */

  prev = signal( SIGFPE, sig_handler );

  /* ------------------------------------------------------------ */
  /* execute CONNECT                                              */
  /* ------------------------------------------------------------ */

  exec sql connect to default;

  /* do whatever needs to be done                                */

  /* ------------------------------------------------------------ */
  /* force exception                                              */
  /* ------------------------------------------------------------ */
  a=a/0;
  /* do whatever needs to be done                                */
}
```

Handling Exits

When entering ESQLNK for the first time, the ADABAS SQL Server system layer tries to register a function which is to be called at program termination.

This is done to prevent that the application program terminates silently without issuing a DISCONNECT statement.

This functionality is implemented using the OpenVMS system call: `SYS$DCLEXH()`

How to Use the SYS$DCLEXH()/atexit() Functions

If you want to set up your own exit handler for your application the SYS$DCLEXH() or atexit() functions can be used depending on the environments in which your application is to be run. Example:

```
void exi_handler( )
{
  printf("my exi-handler ... \n");
  /* do whatever needs to be done                                */
}
main()
{
  /* ------------------------------------------------------------ */
  /* establish 'exi_handler'                                      */
  /* ------------------------------------------------------------ */

  atexit( exi_handler );

  /* ------------------------------------------------------------ */
  /* execute CONNECT                                              */
  /* ------------------------------------------------------------ */

  exec sql connect to default;

  /* do whatever needs to be done                                */
}
```

In this example, the ADABAS SQL Server exit handler, which is activated during the CONNECT statement, will be executed first. Then "exi_handler" will be called by OpenVMS.

Note:
If you place your exit handler behind the first ESQLNK call (first EXEC SQL in your program), then the order of the exit handler calls is reversed.

3

OPERATING THE ADABAS SQL SERVER UTILITIES

This chapter contains a detailed description of how to work with the three ADABAS SQL Server Utilities.

1 BASIC Interactive Facilities (BASIC)

The BASIC Interactive Facilities consist of two elements which can be used on any platform.

- BASIC Interactive SQL allows the user to enter SQL statements interactively and see results displayed on the screen immediately.
- BASIC Catalog Retrieval allows the user to retrieve catalog information via predefined views.

2 ADABAS SQL Server Migration Utility

This utility serves as a migration tool especially designed for the upgrade from the ADABAS SQL Server Version 1.3.x to the ADABAS SQL Server Version 1.4.2.

3 Generate Table Description Utility

If existing ADABAS files must be introduced to the ADABAS SQL Server via the CREATE TABLE/CLUSTER DESCRIPTION statements, the drafting of such statements can be a complex operation, as well as a laborious one. By using this utility, you can easily draft up CREATE TABLE/CLUSTER DESCRIPTION statements. They may then be edited by hand and be submitted to the server via the BASIC Interactive SQL Utility.

BASIC Interactive Facilities

Target Users

BASIC has been designed for users who require quick and easy access to:

- interactively entered SQL statements and immediate results.
- catalog information, for example, databases, tables, views.

Overview

The BASIC Interactive Facilities are an interactive SQL query tool which supports basic functionality and is fast and effective. The BASIC Interactive Facilities consist of two elements:

- BASIC Interactive SQL and
- BASIC Directory Retrieval

Statements containing cursor names, host variables or parameter markers cannot be executed interactively. Statements marked for embedded or dynamic mode in the *ADABAS SQL Server Reference Manual*, for example, the CLOSE, DECLARE or CONNECT statements cannot be used in this context.

Invoking the BASIC Interactive Facilities

To invoke the BASIC Interactive Facilities, type the following:

```
$ esqint [[servername] [communication] [destination]]
```

The optional parameters must be specified according to the same rules as in the Server Routing File. For details see the *ADABAS SQL Server Programmer's Guide*, chapter: **Client/Server Topics.**

Setting Server Name, Communication and Destination

Examples:

To connect to a local server named LOCESQ, enter:

```
$ esqint LOCESQ
```

To connect to the remote server named VAXESQ via CSCI on node SAGVAX11, enter:

```
$ esqint VAXESQ CSCI SAGVAX1
```

To connect to the remote server named IBMESQ via the ENTIRE BROKER with the broker ID IBMBROKER11, enter:

```
$ esqint IBMESQ BROKER IBMBROKER11
```

Setting User ID and Password

BASIC will then issue prompts asking for the user ID and password which will be used to perform a CONNECT statement to the server. When the input is redirected to a file, the user ID and password must be in the first two lines of that file.

Alternatively, the symbol ESQUSER can be defined with user ID and password. The prompting will be suppressed in this case. In a system with the security feature, the symbol ESQUSER must contain both the user and the password, separated by a comma: ‘userid,password’. In a non-security system, the environment variable needs to contain the user ID only.

If, in a non-security system, an empty string is used for the user ID, a default CONNECT will be executed. The user ID is then identical to the operating system user ID.

```
$ esqint

ESQ User:
Password:

esqint:
```

A system prompt will appear and ad-hoc SQL statements may be entered at this point. For details on how to work with BASIC, refer to the platform-independent **Appendix A** to this manual.

BASIC Interactive SQL

BASIC Interactive SQL allows the user to enter SQL statements interactively and see results displayed on the screen. Conceptually it consists of 3 elements:

- Input Mode with a set of Input Mode Commands,
- Output Mode with a set of Output Mode Commands,
- Help Mode with Online Help Texts.

In addition to these three elements, there is a set of Global Commands for general navigation purposes.

Global Commands

Command	Description
exit	returns to the higher level.
help	activates the help function.
quit	returns to the higher level.
spawn *command*	spawns the specified DCL command.
$ *command*	spawns the specified DCL command (abbreviated form).

Input Mode (esqint:)

When you receive the prompt 'esqint:' BASIC is automatically in the input mode and a line editor is provided. SQL statements can now be entered or edited. An SQL statement can consist of several lines. Each statement must end with a semicolon. If ENTER or RETURN is pressed and no semicolon is found, the line editor provides a new line with a descending line number. If a semicolon is found the statement is executed immediately after pressing ENTER or RETURN. Executing a statement automatically activates the output mode where the results are displayed. Only one statement can be active at a time and this so-called currently active statement can be run or edited again. In the case of syntax errors or other reasons for re-editing the statement: leave the output mode, list and edit the statement and run it again. Already saved statements have to be read into the input session to take the place of the currently active statement. If the previously active statement has not been saved, it is no longer available. The compressed maximum length of a statement is 63000 characters, whereby the term compressed means that multiple white spaces are replaced one blank.

Executing Interactive SQL Statements

After entering a new valid SQL statement on the blank input screen, properly end it with a semicolon and press ENTER or RETURN to immediately execute the statement.

A currently active and valid statement can be executed via the ‘run’ command.

A saved and valid statement must be read into the input area and executed via the ‘run’ command.

Input Mode Commands

Command	Description
continuation '*x*'	defines the character 'x' as a line continuation marker.
edit *nn*	edit line *nn* of the current SQL statement.
list	lists the current SQL statement.
read *xx*	reads the file *xx* into the input session.
run	executes/runs the current SQL statement.
save *xx*	saves the SQL statement with the name *xx*.
server	displays server information.

Note:
On mainframe platforms for the read and save commands the file xx is the name of the DD-card.

Output Mode (output (col *l-r* of *t*):)

The output mode is invoked by executing a valid SQL statement and is indicated by the prompt 'output (col *l-r* of *t*):'. The numbers in brackets indicate the offset in spaces when executing the output mode command 'right'.

Where *(l)* and *(r)* mark the left and right margin of the output window in reference to the size of the total output window (*t*) which depends on the number of columns requested.

The other output mode commands available are 'left' which moves the display window to the left and 'home' which returns to column 1. Scrolling down page by page is achieved by pressing ENTER or RETURN. To return to the input mode, the global commands 'quit' or 'exit' are used.

Output Mode Commands

Command	Description
home	scrolls back to the first column.
left	scrolls output window to the left.
right	scrolls output window to the right.

Online Help

Typing ‘help’ after the prompt will display the online help text informing the user about the commands available in BASIC.

Summary: Executing Statements

Working with the Currently Active Statement:

1. At the prompt ‘esqint:’ type in the statement considering the syntax regulations. The statement may exceed one line,
2. delimit each statement with a semicolon (;),
3. to immediately execute the statement, press ENTER or RETURN. You are now in the output mode. If you want to edit the statement (it is still active) or simply want to return to the input mode,
4. enter ‘quit’ at the prompt output(col *l-r* of *t*):,
5. enter ‘list’ which will list the entire statement and note the line number(s) to be edited (only one line can be edited per edit command),
6. enter ‘edit *n’* where *n* is the number of the line to be edited. After editing, you can either execute the statement again by entering ‘run’ or save it with a name by entering ‘save *name’*.

Working with a Previously Saved Statement:

1 A saved statement has to be read into the input session to gain the status of an currently active statement. Enter ‘read *name’,*

2 now you may list, edit, run or save the statement as described above.

BASIC Catalog Retrieval

BASIC Catalog Retrieval allows the user to retrieve information stored in the catalog, like tables, columns, etc.

How to Retrieve Information from the Catalog

The following views are available: databases, views, indexes, metaprograms, tables and columns. After starting the BASIC Interactive Facilities, the following statements may be entered:

To display all databases, type the following:

```
esqint: select * from information_schema.databases;
```

To display all tables, type the following:

```
esqint: select * from information_schema.tables;
```

To display all tables, type the following:

```
esqint: select * from information_schema.tablespaces;
```

To display all columns of table SAILOR, type the following:

```
esqint: select * from information_schema.columns where table_name =
”SAGTOURS.SAILOR”;
```

To display all views, type the following:

```
esqint: select * from information_schema.views;
```

To display all indexes, type the following:

```
esqint: select * from information_schema.indexes;
```

Note:
The results of the SELECT statements are displayed in output mode in the same manner as other SELECT statements entered in BASIC Interactive SQL. The same set of commands applies.

The Migration Utility

Target Users

The Migration Utility is to be used as an aid migrating the contents of an ADABAS SQL Server Version 1.3 directory to an SQL catalog as supported by the ADABAS SQL Server Version 1.4.

Overview

Despite the extensive increase in functionality offered by the 1.4 version of the ADABAS SQL Server, in general, upward compatibility is assured. However, certain steps are necessary in order to ensure a smooth migration. These steps are supported by the provision of a Migration Utility. Migration primarily concerns itself with the problems posed by a radically new catalog structure. The new catalog complies with the 1992 ANSI SQL Standard and is not compatible with the directory structure contained in the ADABAS SQL Server Version 1.3.

The Migration Utility takes the the 1.3 directory contents as input and produces a file containing the generated DDL statements (in the following referred to as ‘generated file’). The Migration Utility then starts the BASIC Interactive Facilities to process the contents of this generated file.

Although the catalog must be established completely afresh, the underlying user data, contained in the target ADABAS files, can remain undisturbed during the process of migration. We recommend that a backup of all data is made prior to migration.

It may, however, not be necessary to use the services of the Migration Utility. If the contents of the existing directory are well documented and all create table description statements and all create view statements have been saved, then this is effectively equivalent to the output of the Migration Utility. However, the three concepts of users, schemas and security must still be considered (see below). In addition, as CREATE TABLE DESCRIPTION statements are now actively checked for accuracy, it may be, in isolated cases, that what was previously permitted is now rejected.

Once migration has been completed, the old 1.3 server must be removed. It is not recommended to run a 1.4 server and a 1.3 server in parallel, serving the same set of ADABAS target files.

Prerequisites

- A prerequisite for the use of the Migration Utility is an installed ADABAS SQL Server Version 1.4. It is strongly recommended that the new catalog is empty prior to migration to avoid conflicting situations.
- Another prerequisite is that the 1.3 directory file and 1.4 target catalog files reside in the same ADABAS database.

New Concepts

Due to the various new concepts, it may be necessary to provide manual input to the generated file. In particular, the following issues must be considered:

- A user concept. It is now necessary to introduce the concept of authenticated users. All users of the ADABAS SQL Server must be made known via the CREATE USER statement.
- A schema concept. Data structures are “contained” in schemata. A schema implies ownership.
- A security concept. If the server has been installed with the security option, then privileges must be granted to users as required.

Special Considerations

In general, pre-compiled applications will not require modification or indeed re-building in any way. However, non-upwardly compatible features will result in the need for modification:

- Existing identifiers now equivalent to a new keyword. New keywords have been introduced. If an existing identifier is the same as one of these keywords, then the identifier must be modified.
- DDL and DML statements mixed within the same transaction. Such statements must be in separate transactions.
- Certain DDL statements may not be upwardly compatible. These are:
  CREATE DATABASE (if host variable was used)
  CREATE TABLE DESCRIPTION

Invoking the Migration Utility

This utility serves as a migration tool especially designed for the upgrade from version 1.3 to version 1.4.

To invoke the Migration Utility type the following:

```
$ esqmigrate  server_name  dir_fnr   [-v] [-t] [-n] [-s] [-h]
```

server name	name of the destination server for the migration.
dir_fnr	ADABAS file number of the version 1.3 server directory.
−v	no generation of VIEWs.
−t	generation of CREATE TABLE and CREATE TABLESPACE statements.
−n	no generation of CREATE TABLE DESCRIPTION or CREATE TABLE statements.
−s	no generation of CREATE TABLESPACE statements. Only valid in conjunction with the −t option.
−h	preparation for semi-automatic migration. Do not specify in the case of automatic, non-security migration.

The default statement leads to the generation of TABLE DESCRIPTIONs and VIEWs. The Migration Utility requires an ADABAS SQL Server Version 1.4 environment.

During invocation, a file named ESQ$SRVDIR:ESQMIG_*dir_fnr*.SQL will be generated, where *dir_fnr* is the version 1.3 directory file number. The file will be referred to as the ‘generated file’ throughout this section.

Automatic Migration

The generation and the loading of the objects takes place in one step. This type of migration makes no provision for the use of the new concepts of the Version 1.4, which means, no security features, no user concept, and no schema concept.

The owner of all objects in this case is the user DBA.

For each specified database a CREATE DATABASE statement is generated. Should there be more than one database identifier for a particular database number in the original directory, then the Migration Utility will still only generate one CREATE DATABASE statement.

The BASIC Interactive Facilities (esqint command) will not be performed if errors or warnings occur. In this case:

- check the error messages issued by the Migration Utility,
- correct the generated file,
- start the BASIC Interactive Facilities manually using the esqint command.

Note:
For more information on servers generated without security features refer to ***Step 4*** *in the chapter* ***Installing the ADABAS SQL Server System*** *earlier in this manual.*

Semi-Automatic Migration

If the new concepts of ADABAS SQL Server Version 1.4 are to be used, it is necessary to postprocess the generated file ESQ$SRVDIR:ESQMIG_*dir_fnr*.SQL in a second step before submitting it to the BASIC Interactive Facilities for execution.

For each user, a CREATE USER statement is to be generated and for the schemas to be created, the appropriate authorization is to be specified.

If further databases, other than the default database, are already defined in the catalog, it may also be necessary to postprocess the generated file. This is particularly true, if table descriptions existed in the version 1.3 specifying a database that is already defined in ADABAS SQL Server Version 1.4. In this case, the generated CREATE DATABASE statements are to be adapted by hand.

2-Steps Semi-automatic Migration:

1 Invoke the Migration Utility to establish the generated file.

2 Adapt the file to reflect the required data structures. Adaptations are necessary for the following three cases:

Case I: If you want to use the owner concept for schemas, perform the command esqmigrate with the −h option and a file is generated with the following structure:

```
# !!!!!!!!!!!! START PART1 !!!!!!!!!!!!!
DBA
# ******************************************
# * REPLACE THE FOLLOWING ??? BY A USER NAME  *
# * AND A PASSWORD SUCCESSIVELY!              *
# ******************************************
# CREATE USER ??? PASSWORD ???;
# .
# .
# CREATE USER ??? PASSWORD ???;
# !!!!!!!!!!!! END   PART1 !!!!!!!!!!!!!
# !!!!!!!!!!!! START PART2 !!!!!!!!!!!!!
# DBA
# ****************************************
# * REPLACE THE FOLLOWING ??? BY THE USER *
# * WHO WILL BE THE OWNER OF THE SCHEMA!  *
# ****************************************
# CREATE SCHEMA schema1 AUTHORIZATION ???;
CREATE SCHEMA schema1;
# !!!!!!!!!!!! END   PART2 !!!!!!!!!!!!!
# !!!!!!!!!!!! START PART3 !!!!!!!!!!!!!
# ************************************
# * REPLACE THE FOLLOWING ??? BY THE    *
# * OWNER OF THE SCHEMA schema1!        *
# ************************************
#
CREATE DATABASE...;
CREATE TABLE DESCRIPTION schema1.table1...;
CREATE TABLE DESCRIPTION schema1.tablen...;
# !!!!!!!!!!!! END   PART3 !!!!!!!!!!!!!
# !!!!!!!!!!!! START PART4 !!!!!!!!!!!!!
DBA
# ****************************************
# * REPLACE THE FOLLOWING ??? BY THE USER *
# * WHO WILL BE THE OWNER OF THE SCHEMA!  *
# ****************************************
# CREATE SCHEMA schema2 AUTHORIZATION ???;
  CREATE SCHEMA schema2;
# !!!!!!!!!!!! END   PART4 !!!!!!!!!!!!!
# !!!!!!!!!!!! START PART5 !!!!!!!!!!!!!
# ************************************
# * REPLACE THE FOLLOWING ??? BY THE    *
# * OWNER OF THE SCHEMA schema2!        *
```

```
  # ************************************
  CREATE TABLE DESCRIPTION schema2.table1...;
  ...
CREATE TABLE DESCRIPTION schema2.tablen...;
  # !!!!!!!!!!!! END   PART5 !!!!!!!!!!!!!
```

Edit the file as follows:

- copy each part (all between START PARTx – END PARTx) into a file (for example: filex),
- change ”???” to user name or password as indicated in the comment,
- delete the comment signs and CREATE SCHEMA statements where needed.
- call the BASIC Interactive Facilities (esqint) with filex as input.

Case II: If you want to to use database names other than those generated:

- generating an output file using the command esqmigrate with the −h option.
- change the database names in the generated CREATE DATABASE statements from old name to new name.
- replace each old name by a new name in the CREATE TABLE DESCRIPTION statements.
- calling the BASIC Interactive Facilities (esqint) with file1 as input.

Case III: There are table and/or column names in the version 1.3 directory, which are keywords in the version 1.4. Delimited identifiers are not supported in version 1.4, therefore, the following steps have to be taken:

- look into the error file esqmig.err and search for warnings: ESQMIG−W−KEYWORD: ”Column/Table name int is also a keyword”
- if there are warnings of this kind, replace the names in question by new names that are not keywords in ADABAS SQL Server Version 1.4
- Call the BASIC Interactive Facilities (esqint) using the generated file as input.

Note:
Make sure to change your applications to reflect the change of identifiers.

Generate Table Description Utility

Target User

Anyone who wants to introduce an existing ADABAS file to the ADABAS SQL Server

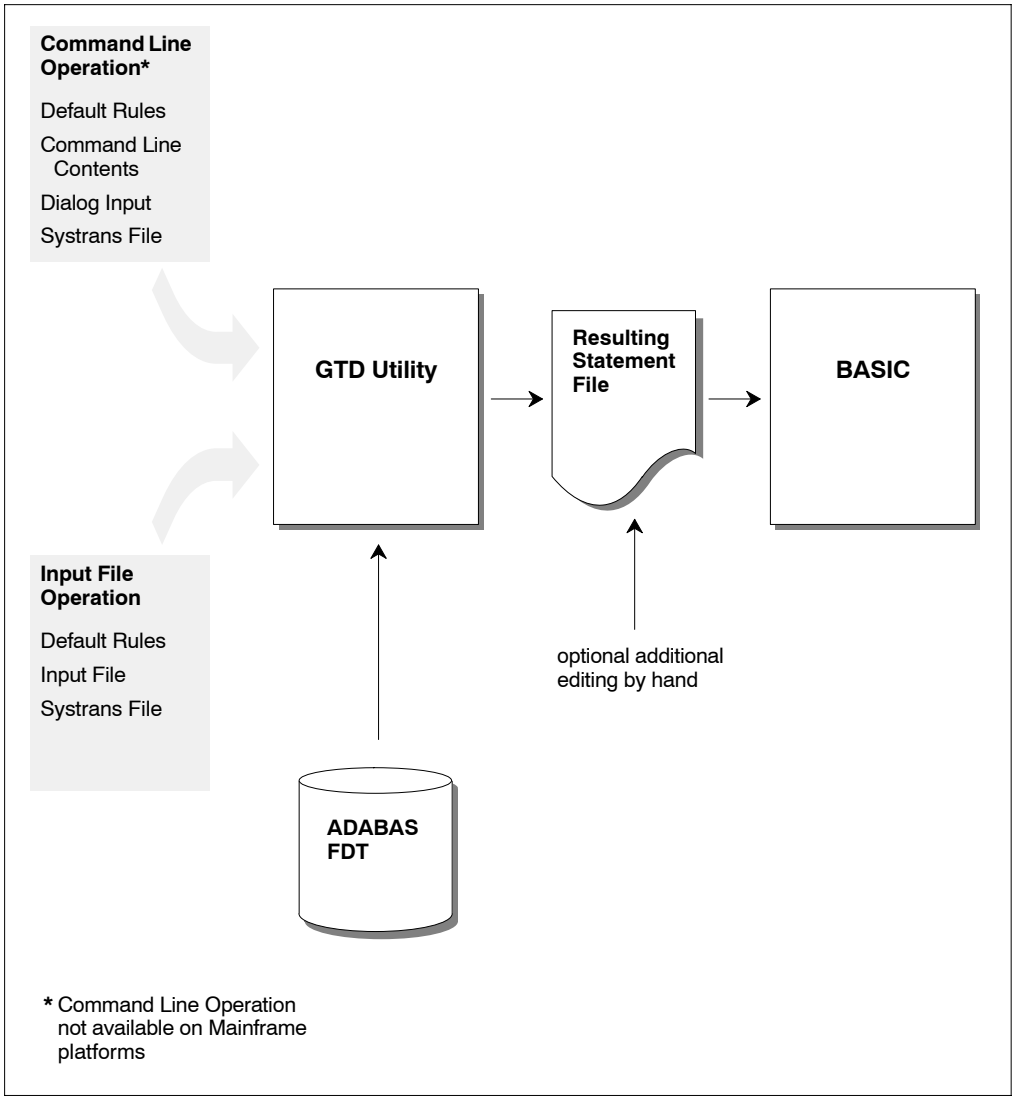
Overview

The Generate Table Description Utility (in the following called GTD Utility) is used to generate either CREATE TABLE DESCRIPTION statements or CREATE CLUSTER DESCRIPTION statements for existing ADABAS files. To make existing ADABAS files accessible, they must be introduced to the ADABAS SQL Server by means of these statements. The drafting by hand of such statements can be a complex operation, as well as a laborious one. Therefore, the GTD Utility is used to automatically generate a text file containing such statements. This text file must then be submitted to the Basic Interactive Utility for execution, thus introducing the existing ADABAS file to the ADABAS SQL Server.

The GTD Utility can be operated in two distinct operative modes:

- Command Line Operation
  Command line operation is best suited for quick and easy use. From command line operation the dialog mode can be activated as an option. The dialog mode provides a large subset of the full functionality of the utility.
- Input File Operation
  The full functionality of the utility is only available in input file operation. Fine tuning of the resulting generated statements is possible by modifying and resubmitting the input file.

It can be seen from the following diagram, that the GTD Utility obtains information from various sources, depending upon the operative mode. The resultant generated statement is dependent upon the information which the GTD Utility will obtain from one or more of these sources:



Note:
Some of the above information sources are mutually exclusive. Please refer to the section: Informations Sources Overview later in this chapter.

The resultant statement output file is not automatically presented to the ESQINT Basic Interactive Utility. There is therefore always the opportunity to review the generated output and indeed edit it further by hand.

Because the GTD Utility must obtain information from the relevant ADABAS FDT, during its execution, it is therefore a pre-requisite that the appropriate ADABAS database is online and accessible.

Invoking the GTD Utility

As mentioned in the Overview, there are two distinct operative modes. These modes are mutually exclusive.

Command Line Operation

In command line operation mode, the GTD Utility is invoked as follows:

```
$ define/user sys$output output_file
$ esqgtd [option] database_name database_number file_number [ddm_name]
```

where:

output_file	must indicate a valid file path name. The indicated file will be generated containing the resultant statement.
option	none, one or more of the following options may be specified in any order:
–f	dialog mode is suppressed: default rules therefore apply.
–d	the generation of the column's data type is forced in the statement. Normally, the data type is only generated for longalpha character fields, or numeric or decimal fields where a scale is specified in a NATURAL DDM SYSTRANS file.
–t *systrans_file*	this option has an additional parameter. The option indicates that a NATURAL DDM SYSTRANS file is to be used as an additional information source. The parameter *systrans_file* must indicate a valid file path name, where the file contains the SYSTRANS DDM information to be considered.
–s *schema_name*	this option has an additional parameter. The option indicates that an explicit schema identifier is to be generated in the resultant statement. The parameter *schema_name* specifies this identifier.

−a	in dialog mode, a prompt is issued for all fields. Normally, only MU/PE fields are prompted.
−o	in dialog mode, a prompt is issued for the column and subtable identifiers.
−h	invokes on-line help only.
database_name	specifies the name of the ADABAS SQL Server target database. This name will appear in the final statement.
database_number	specifies the number of the ADABAS source database. The ADABAS file to be described must reside in this database.
file_number	specifies the number of the ADABAS source file to be described.
ddm_name	this optional parameter specifies the particular SYSTRANS DDM in the *systrans_file* which is to be used as additional information. If the *ddm_name* contains a period '.', the part before the period is the NATURAL library name and the part after the period is the real DDM name. If *the ddm_name* is omitted, then the DBID and file number are used for the purposes of identification.

For example:

```
$ define/user sys$output esqgtd.out
$ esqgtd −f −t systrans.file DBNAME 202 181
```

Note:
The files esqgtd.out and systrans.file are to be found in the local directory.

Input File Operation

In input file operation mode, the GTD Utility is invoked as follows:

```
$ define/user sys$output output_file
$ define/user sys$input input_file
$ esqgtd
```

where:

input_file	must indicate a valid file path name. The indicated file must contain a valid input file specification. For details refer to the section: **Input File Language Syntax** later in this chapter.
output_file	must indicate a valid file path name. The indicated file will be generated containing the resultant statement.

For example:

```
$ define/user sys$output esqgtd.out
$ define/user sys$input esqgtd.in
$ esqgtd
```

Note:
The files esqgtd.out and esqgtd.in are to be found in the local directory.

Information Sources Overview

The interpretation that can be placed upon an ADABAS file in terms of SQL table mapping is numerous. The GTD Utility can generate differing statements based upon the same ADABAS FDT depending on the information that is presented to it. The information sources and the information presented within these sources, control the nature of the resultant statement.

Please refer to the diagram earlier in this chapter.

The ADABAS FDT

The GTD Utility has the task of producing a description of an ADABAS file in terms of SQL. It therefore takes, as its starting point, the ADABAS FDT (Field Description Table) of the file to be described. The appropriate ADABAS database must be online and accessible to the GTD Utility. The information derived from this source is:

- The ADABAS file structure.
- The ADABAS field short names.
- The ADABAS field data types.

Command Line Information

When in Command Line Operation mode, the GTD Utility obtains vital information via the command line input. The presence of command line input, automatically suppresses Input File Operative mode. Some of the information is mandatory. Some of the information also indicates if further information sources are available. The information derived from this source is:

- The DBID and file number of the ADABAS file to be described (mandatory).
- An indication if dialog mode is to be used as an information source (optional).
- An indication if a NATURAL DDM SYSTRANS file is to be used as an information source (optional).

Should additional information sources not be available, i.e. the dialog mode has been suppressed (the –f option) and no SYSTRANS file has been specified, then the resultant statement is generated according to the default rules.

Dialog Input

If in the command line information, dialog input has not been explicitly suppressed, then a dialog will be initiated by the GTD Utility with the user. The user dialog is intended as a comfortable interface to the GTD Utility, which is quick and easy to use and will cover most cases. The user always has the option to manually edit the resultant file. Via the dialog, the interpretation of the ADABAS fields themselves can be controlled and the following can be influenced:

- PE groups can be interpreted as a subtable.
- PE groups can be suppressed, in which case:
  - PE fields can be interpreted as a rotated field.
  - PE fields can be interpreted as long-alpha (if of data type character).
  - MU fields within the PE group are automatically suppressed.
- MU fields can be interpreted as a subtable.
- MU fields can be suppressed.
- MU fields can be interpreted as a rotated field.
- MU fields can be interpreted as long-alpha (if of data type character).

If the GTD Utility is additionally invoked with the '–o' option, then the user is prompted for the column and subtable identifiers. Otherwise the default rules for identifier generation are employed.

If the GTD Utility is invoked with the –a option, then a prompt is issued for all fields, thus enabling the suppression of a non MU or PE field. Otherwise only fields where a decision is required would be presented within the dialog i.e. MU or PE fields.

Default Rules

Default rules apply when no other information is available to the GTD Utility. This may be that no information is available at all for the specified ADABAS file, in which case the default rules apply to all aspects of the statement generation. Alternatively, no additional information is available for a particular field, in which case the default rules apply only to that particular field.

By default:

- Any MU field will be interpreted as a subtable.
- Any PE group will be interpreted as a subtable.
- If the file contains no MU or PE fields, then a CREATE TABLE DESCRIPTION statement is generated.
- If the file contains MU or PE fields, then a CREATE CLUSTER DESCRIPTION statement is generated.
- All available SEQNO columns will be generated into the table descriptions.
- Any foreign key relationship will always be based upon appropriate SEQNO column(s).
- The cluster identifier will be 'CLUSTER_XXX', where 'XXX' is the file number.
- The master table identifier will be 'TABLE_XXX', where 'XXX' is the file number.
- The table identifier of a subtable derived from a PE group will be 'TABLE_XXX_YY', where 'XXX' is the file number and 'YY' is the PE group short name.
- The table identifier of a subtable derived from an MU field will be 'TABLE_XXX_YY', where 'XXX' is the file number and 'YY' is the field short name.
- The table identifier of a subtable derived from numerous MU fields in parallel will be 'TABLE_XXX_YY', where 'XXX' is the file number and 'YY' is the field short name of the first MU field. Note this interpretation of numerous MU fields is itself not by default.
- The column identifier will be 'COL_YY', where 'YY' is the field short name.

The pure default operation is intended for quick spontaneous use. It does, however, lead to very cryptic column and table names and may also lead to cluster structures which are not what the user requires. This approach is intended as a starting point. The user always has the option of either editing the generated statement by hand or introducing additional information sources.

The NATURAL DDM SYSTRANS File

A NATURAL DDM SYSTRANS file can be introduced to the GTD Utility, primarily for the purpose of supplying long names for columns and indeed, in some cases, for the tables themselves. The information that it contains is merged with the basic information of the ADABAS FDT.

Note:
If introducing the SYSTRANS file to the GTD Utility results in the error message: % ESQGTD−E−INVREC, Invalid record structure found in SYSTRANS file <name>, refer to the description of the NATURAL DDM SYSTRANS File Conversion Tool below.

The following information is extracted from a NATURAL DDM SYSTRANS file:

- Long names for master tables.
- Long names for periodic group-based (PE) subtables.
- Long names for multiple-value fields (MU) based subtables.
- Long names for column identifiers (except SEQNO columns).
- The precision and scale of a column of data type numeric or decimal.
- The explicit data type of natural date or natural time columns.

Names of master tables, or tables which do not appear in a cluster description, are derived from the actual DDM name itself. If this name is qualified with a library name, then this is ignored for the purposes of table identifier generation.

Names of subtables based upon a PE group are derived from the concatenation of the generated table identifier (itself derived from the DDM name) with the PE group long name.

Names of subtables based upon an MU are derived from the concatenation of the generated table identifier (itself derived from the DDM name) with the MU field long name.

Names of subtables based upon numerous MU fields in parallel are derived from the concatenation of the generated table identifier (itself derived from the DDM name) with the first MU field long name.

Column names are simply derived from the long name associated with the appropriate ADABAS field. If the column is derived from a rotated field, then a numeric suffix is concatenated.

Should any resultant identifiers not correspond to a legal SQL identifier, then it will be appropriately amended. Such an action will result in a warning being generated. Examples of common conversions are as follows:

- A delimiter character has been used in the long name e.g. 'yearly–bonus' would become 'yearly_bonus'
- An SQL keyword has been used for a long name e.g. 'group' would become 'group_'
- The long name does not start with an alphabetic character e.g. '1stbonus' would become 'col_1stbonus'.

Should any resultant long name contain more than the permitted 32 characters, then a warning is generated. It is then necessary to amend such identifiers by hand in the output file. Failure to do so will result in an error when attempting to execute the generated statement.

Any information extracted from a DDM file is subordinate to any contradictory information presented to the GTD Utility from other sources.

It is sometimes the case that the DDM in a SYSTRANS file refers to a DBID and a file number that does not correspond with the physical DBID and file number of the file concerned. In such a case, an explicit DDM name must be supplied, in order to identify the desired DDM. In such a case the DBID and the file number given in the DDM are ignored.

It is sometimes the case that the SYSTRANS DDM file actually contains more than one DDM representation. Only one DDM may be considered by the GTD Utility during execution. If the SYSTRANS DDM file does contain more than one DDM representation and the desired DDM is not explicitly specified, then the DBID and file number are used for the purposes of identification. A DBID of '0' in the SYSTRANS DDM file, will be recognized as a match. The first DDM to be found will be the one used during execution, should there be more than one DDM in the file which match the search criterion. Alternatively if an explicit name was given, then the first instance of the specified DDM will be used.

For information on how to create a NATURAL SYSTRANS DDM file, refer to the appropriate NATURAL documentation.

NATURAL DDM SYSTRANS File Conversion Tool

The formatting of the SYSTRANS files generated by several NATURAL versions differs on the various platforms.

Should the GTD Utility reject one of these formats, the following error message will be issued: *% ESQGTD−E−INVREC, Invalid record structure found in SYSTRANS file <name>*. In this case, the NATURAL DDM SYSTRANS File Conversion Tool must be applied on the file, which will convert the original file into a GTD-acceptable format. The tool is started via the ESQCON command:

```
$ esqcon systrans_file_name [/del]
```

If the optional parameter /del is specified the original SYSTRANS file will be deleted automatically. If not specified, a copy of the original file will remain and the extension '_orig' will be added to the original file name.

The Input File

The input file can only be presented to the GTD Utility in Input File Operation Mode.

All aspects of the statement generation can be controlled via the input file. In addition to this information source, the information sources NATURAL SYSTRANS DDM file and default information can also be employed. However, information contained in the input file has precedence over any other information source.

In addition to the control which is available in dialog mode, the following functionality is available when using an input file:

- Suppression of SEQNO columns.
- Long names for SEQNO columns.
- Non-SEQNO primary keys in master tables.
- Long names for foreign key columns.
- Specific names for columns derived from rotated field members.
- Subtables consisting of more than one MU field.

Should a particular optional piece of information be omitted from the input file, then the default rules will apply in this particular regard.

How to .....

As already mentioned above, the whole purpose of the GTD Utility is to produce a CREATE TABLE/CLUSTER DESCRIPTION statement, which maps an existing ADABAS file to an SQL data structure. In doing so, choices between various alternatives must be made. A clear understanding of what is to be achieved is required. This section sets out what particular actions are possible and how to invoke them.

Generate a Cluster Description

The GTD Utility can generate a CREATE CLUSTER DESCRIPTION statement or alternatively a CREATE TABLE DESCRIPTION based upon a single ADABAS file.

Default Rules

If the ADABAS file contains MU or PE fields, then a CREATE CLUSTER DESCRIPTION statement is generated by default. Otherwise a CREATE TABLE DESCRIPTION statement is generated.

Dialog Mode

If the ADABAS file does not contain MU or PE fields, then a CREATE TABLE DESCRIPTION statement is always generated. If however, the ADABAS file does contain MU or PE fields and if these fields are either suppressed or rotated or (if applicable) determined to be long alpha columns, then a CREATE TABLE DESCRIPTION statement is also generated. Otherwise a CREATE CLUSTER DESCRIPTION statement is generated

Input File

If the ADABAS file does not contain MU or PE fields, then a CREATE TABLE DESCRIPTION statement is always generated. If however, the ADABAS file does contain MU or PE fields and if these fields are either suppressed or rotated or (if applicable) determined to be long alpha columns, then a CREATE TABLE DESCRIPTION statement is also generated. Otherwise a CREATE CLUSTER DESCRIPTION statement is generated

Generate a Table Description

The GTD Utility can generate a CREATE TABLE DESCRIPTION statement or alternatively a CREATE CLUSTER DESCRIPTION based upon a single ADABAS file.

Default Rules

If the ADABAS file does not contain MU or PE fields, then a CREATE TABLE DESCRIPTION statement is always generated.

Dialog Mode

If the ADABAS file does not contain MU or PE fields, then a CREATE TABLE DESCRIPTION statement is always generated. If however, the ADABAS file does contain MU or PE fields and if these fields are either suppressed or rotated or (if applicable) determined to be long alpha columns, then a CREATE TABLE DESCRIPTION statement is also generated. Otherwise a CREATE CLUSTER DESCRIPTION statement is generated.

Input File

If the ADABAS file does not contain MU or PE fields, then a CREATE TABLE DESCRIPTION statement is always generated. If however, the ADABAS file does contain MU or PE fields and if these fields are either suppressed or rotated or (if applicable) determined to be long alpha columns, then a CREATE TABLE DESCRIPTION statement is also generated. Otherwise a CREATE CLUSTER DESCRIPTION statement is generated.

Generate a Database Identifier

A database identifier is required as part of the syntax of a CREATE TABLE/CLUSTER DESCRIPTION statement.

Default Rules

No default available.

Command Line

The database identifier is specified as one of the command line parameters.

Dialog Mode

Not available.

Input File

The database identifier is specified via a database directive.

For example:

```
DATABASE 214 DB_NAME
```

Generate a File Number

A file number is required as part of the syntax of a CREATE TABLE/CLUSTER DESCRIPTION statement.

Default Rules

No default available.

Command Line

The file number is specified as one of the command line parameters.

Dialog Mode

Not available.

Input File

The file number is specified via the file specification directive.

For example:

```
FILE 181
```

Generate a Cluster Identifier

A cluster identifier is required as part of the syntax of a CREATE CLUSTER DESCRIPTION statement.

Default Rules

The cluster identifier is generated as 'CLUSTER_XXX' where 'XXX' is the source ADABAS file number.

Dialog Mode

Not available

Input File

The cluster identifier can be specified via the file specification directive.

For example:

```
FILE 181 CLUSTER_ID
```

Generate a PE Group or an MU Field as a Subtable

Default Rules

By default, a PE group or an individual MU field are always interpreted as a subtable.

Dialog Mode

When a PE group or an MU field is encountered, a prompt will be issued. At this point, the option 'T' should be entered.

Input File

By specifying a subtable directive, a PE group or an MU field can be interpreted as a subtable. The following directive would map the field AB to a subtable.

For example:

```
AB SUBTABLE
```

Generate Multiple MU Fields Subtable

Multiple MU fields can be brought together in one subtable. The fields are then in 'parallel'.

Default Rules

Not possible.

Dialog Mode

Not possible

Input File

This can be specified by use of the SUBTABLE directive. The following directive would map the MU fields to a single subtable containing the fields AB and AC plus the various sequence number columns (total 3 in this example).

For example:

```
AB, AC SUBTABLE
```

Generate a Rotated MU Field

An MU field can also be interpreted as a rotated field.

Default Rules

Not possible

Dialog Mode

When an MU field is encountered, a prompt will be issued. At this point, the option 'R' should be entered. A further prompt will be issued which requires the depth of the rotated field.

Input File

An MU field can be interpreted as a rotated field. The following directive would map the 12 occurrences of the field AB to 12 individual SQL columns.

For example:

```
AB ROTATE 12
```

Generate a Rotated PE Field

An individual PE group member field can also be interpreted as a rotated field.

Default Rules

Not possible

Dialog Mode

Initially, when a PE group is encountered the prompt is issued in order to generate a subtable. However, if at this point the table is suppressed (enter 'S'), then further prompts can be issued which refer to the individual PE group member fields. These can be rotated by entering 'R' in response to the prompt. A further prompt will be issued which requires the depth of the rotated field.

Input File

In order to rotate fields within a PE group, the group itself must be suppressed so that it does not form a subtable. Thereafter, PE group member fields can be rotated. The following directives would generate 12 individual columns based upon the field BA and 20 individual columns based upon the field BB rather than one subtable based upon the PE group BO.

For example:

```
SUPPRESS BO

BA ROTATE 12
BB ROTATE 20
```

Generate a Long-Alpha MU Field

An MU field can also be interpreted as a long-alpha field, provided that the field is of type character.

Default Rules

Not possible

Dialog Mode

When an MU field is encountered, a prompt will be issued. At this point, the option 'L' should be entered. A further prompt will be issued which requires the size of the character column.

Input File

An MU field can be interpreted as a long-alpha field. The following directive would map the field AB to an SQL column of 1024 characters.

For example:

```
AB LONGALPHA 1024
```

Generate Table Identifiers

The generation of table identifiers can be specified.

Default Rules

The master table identifier will be generated as 'TABLE_XXX', where 'XXX' is the file number.

- The table identifier of a subtable derived from a PE group will be 'TABLE_XXX_YY', where 'XXX' is the file number and 'YY' is the PE group short name.
- The table identifier of a subtable derived from an MU field will be 'TABLE_XXX_YY', where 'XXX' is the file number and 'YY' is the field short name.
- The table identifier of a subtable derived from numerous MU fields in parallel will be 'TABLE_XXX_YY', where 'XXX' is the file number and 'YY' is the field short name of the first MU field. Note this interpretation of numerous MU fields is itself not by default

Dialog Mode

In general, the provision of explicit table identifiers in dialog mode is not possible. However, with the specification of the command line option '–o', where appropriate, a prompt is issued requesting a suitable table identifier.

Note:
the specification of a table identifier in dialog mode has precedence over any specification in a NATURAL DDM SYSTRANS file.

Input File

The master table identifier can be specified within the file directive.

For example:

```
FILE 181 Cluster_name City
```

A subtable identifier is specified via the following subtable directive.

For example:

```
B0 SUBTABLE Cities
```

Note:
The specification of a table identifier in the input file has precedence over any specification in a NATURAL DDM SYSTRANS file.

NATURAL DDM SYSTRANS File

- The master table identifier will be generated from the DDM name.
- The table identifier of a subtable derived from a PE group will be concatenation of the DDM name and the PE group name (if available) or the PE shortname.
- The table identifier of a subtable derived from an MU field of level 1 will be the concatenation of the DDM name and the MU field name (if available) or the MU shortname.
- The table identifier of a subtable derived from an MU field of level 2 will be the concatenation of the DDM name, the PE name and the MU field name (if available) or the MU shortname.

Generate Column Identifiers

The generation of column identifiers can be specified.

Default Rules
A column identifier will be generated as 'COL_YY', where 'YY' is the field short name.

Dialog Mode
The specification of a column identifier in dialog mode has precedence over any specification in a NATURAL DDM SYSTRANS file.

Input File
A column identifier is explicitly specified via the name directive. This specification would cause the field AB to be generated with the name state_name.

For example:

```
NAME AB state_name
```

Note:
The specification of a column identifier in dialog mode has precedence over any specification in a NATURAL DDM SYSTRANS file.

NATURAL DDM SYSTRANS File
Any name associated with a particular field will be used as the column's identifier.

Generate Columns Identifiers for Rotated Field Members

Each occurrence of a rotated field must be uniquely identified as a column within the table.

Default Rules

Although rotated fields themselves are not by default generated, identifiers for them are. The default column identifier for a rotated field is 'COL_YY_Z' where 'YY' is the field shortname of the MU field or the PE group member field and 'Z' is the occurrence number of the particular column.

Dialog Mode

In general, the provision of explicit column identifiers in dialog mode is not possible. However, with the specification of the command line option '–o', where appropriate, a prompt is issued requesting a suitable column identifier prefix. The generated column identifier is the concatenation of this prefix plus the occurrence number of the particular column.

Input File

A rotated field column identifier is explicitly specified via the name directive:

For example:

```
NAME MA 3 BONUS_MARCH
```

NATURAL DDM SYSTRANS File

Any name associated with a particular field will be used as the column's identifier prefix.

Generate SEQNO Column Identifiers

Any generated SEQNO column will have a column identifier. The contents of a NATURAL DDM SYSTRANS file has no influence on the generation of column identifiers for SEQNO columns

Default Rules

A SEQNO column identifier is by default 'COL_SEQNO_0' for level 0 sequence number column, 'COL_SEQNO_1' for level 1 and 'COL_SEQNO_2' for level 2.

Dialog Mode

No action possible.

Input File

A SEQNO column identifier is explicitly specified via the name directive. The shortname is given as '*0' for a level 0 sequence number, '*1' for a level 1 and '*2' for a level 2:

For example:

```
NAME *0 STATE_ID
```

Generate a Primary/Foreign Key Specification

It is possible to generate primary/foreign key relationships for the various master and subtables, other than what is available by default.

Default Rules

- The primary key for a master table is the column COL_SEQ_0.
- The primary key for a level 1 subtable is the columns COL_SEQ_0 and COL_SEQ_1.
- The primary key for a level 2 subtable is the columns COL_SEQ_0, COL_SEQ_1 and COL_SEQ_2.

Dialog Mode

No action possible.

Input File

If a primary key is required, based on columns other than just the SEQNO column then this is specified via the primary directive.

For example:

```
PRIMARY AA abbreviation
```

Similarly, if a foreign key is required, based on columns other than just the SEQNO column then this is specified via the FOREIGN directive.

For example:

```
FOREIGN AA stat_abrv
```

Suppression of Columns

It may be that certain fields are not required in the generated statement.

Default Rules

All fields are included in the generated statement.

Dialog Mode

Fields that would 'naturally' induce a prompt can be suppressed via the input 'S'. These are fields which are either members of a PE group or are MU fields.

In general in dialog mode, 'normal' fields do not induce a prompt. However, with the specification of the command line option '−a', a prompt is issued for all fields. All fields can therefore be suppressed via the input 'S'.

Input File

Fields can be explicitly suppressed via the SUPPRESS directive.

For example:

```
SUPPRESS BA
```

Input File Language Syntax

The input file is used to provide information to the GTD Utility. All aspects of the generation can be controlled via the provision of an input file.

The file consists of directives. Each directive is started by a new line and is terminated by an end of line. The order of the directives is of no significance.

The GTD Utility is only able to process one ADABAS file at a time.

Comments

Comments are delimited by the appearance of a # character in the first column and by a new line.

SYSTRANS DDM File Specification

Should a SYSTRANS DDM file be required it must be specified by using the following directive:

```
SYSTRANS systrans_file
```

where:

systrans_file	is a valid file name concerned and indicates the SYSTRANS DDM file.

For example:

```
SYSTRANS systrans.ddm
```

Database Specification

The appropriate ADABAS database must be specified by using the following directive:

```
DATABASE db_nr database_name
```

where:

db_nr	is the target ADABAS database number
database_name	is the associated database name which will be used in the generated description statements.

For successful eventual execution of the generated statement, a CREATE DATABASE statement must already have been executed associating this database name with the given database number.

For example:

```
DATABASE 202 TEST_DB
```

Schema Identifier

In order to specify the schema identifier, the following directive is used:

```
SCHEMA schema_name
```

For example:

```
SCHEMA production_schema
```

File Specification

The source ADABAS file is specified by the following syntax:

FILE *file_nr* [DDM=*ddm_name*] [[*cluster_name*] *table_name*]

where:

file_nr	specifies the ADABAS file from which the description statement will be generated.
DDM=*ddm_name*	optionally specifies the exact DDM representation in the SYSTRANS DDM file which is to be used. If the *ddm_name* contains a period '.', the part before the period is the NATURAL library name and the part after the period is the real DDM name.
table_name	optionally specifies the table name which will be used either for the master table or for the single table if there is no cluster.
cluster_name	optionally specifies the cluster name which will be used in the generated statement. If both a cluster name and a table name are present, then this forces the generation of a CREATE CLUSTER DESCRIPTION statement, even if such a statement is not strictly necessary.

For example:

```
FILE 32  DDM=employees  cluster_employees employees
```

Subtable Specification

A subtable can be built around a PE group, a single MU field or a number of MU fields which act in parallel. The subtable directive is able to express either of these possibilities and to enable the optional specification of an identifier for the resultant subtable. In the first two cases, only a single field is required. In the third case, numerous MU fields can be specified in the form of a list separated by commas. The syntax is as follows:

```
short_name [, short_name]... SUBTABLE [table_name]
```

where:

short_name	is as it suggests.
table_name	represents the intended table identifier for the subtable.

For example:

```
AB SUBTABLE
```

would map the field AB to a subtable and would generate a table name accordingly.

```
AC SUBTABLE Employee_number
```

would map the field AC to a subtable but would use the name as indicated.

```
AE, AF SUBTABLE Employee_status
```

would map the two MU fields to the single subtable Employee_status.

Rotated Field Interpretation

If a field is to be interpreted as a rotated field, then the following syntax applies:

```
short_name ROTATE value
```

where:

short_name	is as it suggests.
value	is the fixed number of occurrences to be rotated e.g. 12 in our example above.

For example:

```
MA ROTATE 12
```

would map the twelve occurrences of the field MA to twelve individual SQL columns.

Longalpha Field Interpretation

If an MU field of type character is to be interpreted as a longalpha field, then the following syntax applies:

```
short_name LONGALPHA <value>
```

where:

short_name	is as it suggests.
value	is the number of bytes which are to be considered in the character field.

For example:

```
LT LONGALPHA 512
```

This would map the field LT to a character column of 512 characters.

Long name Specification

A column identifier for a particular field can be specified as follows:

```
NAME short_name [index] column_name
```

where:

short_name	is as it suggests.
index	optionally refers to the occurrence number of an MU if the field is rotated.
column_name	is a valid SQL identifier representing the intended column identifier.

For example:

```
NAME MA 3 BONUS_MARCH
```

Any name directive overrides any long name for the column derived from a SYSTRANS DDM file.

Field Suppression

If a field is not to be included in a description, then its suppression is specified as follows:

```
SUPPRESS short_name
```

where:

short_name	is as it suggests.

For example:

```
SUPPRESS BA
```

Foreign Key Specification

The GTD Utility, by default forms the primary key/foreign key relationship, based upon the SEQNO columns. If a different basis for this relationship is required, then the FOREIGN directive is necessary. This directive (or directives) must immediately follow the SUBTABLE directive. A foreign key may consist of one or more columns and this is represented simply by a repetition of the directive. The syntax is as follows:

```
FOREIGN short_name [column_name]
```

where:

short_name	is as it suggests.
column_name	is a valid SQL identifier representing the intended column identifier. This is therefore equivalent to the FOREIGN directive without a column name and an appropriate NAME directive.

For example:

```
FOREIGN AA
```

Primary Key Specification

By default, the utility forms the primary key of a table based upon the SEQNO column. By using the PRIMARY KEY directive, a different basis can be taken. A primary key may consist of one or more columns and this is represented simply by a repetition of the directive. The syntax is as follows:

```
PRIMARY short_name [column_name]
```

where:

short_name	is as it suggests.
column_name	is a valid SQL identifier representing the intended column identifier. This is therefore equivalent to the PRIMARY directive without a column name and an appropriate NAME directive

For example:

```
PRIMARY AA
```

Data Type Generation

In general, the GTD Utility does not generate the explicit data type declaration for a column in the resultant generated statement. The explicit data type declaration is however not strictly necessary as the ADABAS SQL Server can operate on a minimum create table/cluster statement, i.e. the server will fill in any missing information automatically. In such a case, the resultant generated statement can be significantly longer. The forced generation of the column's data type is helpful for documentation purposes. The syntax is as follows:

```
DATATYPE
```

Error Handling

Should the GTD Utility encounter an error condition during execution, then it will generally terminate with an exit code and in addition an error message.

The possible exit codes are:

 0 Successful completion / or warnings.
−1 Error detected.

The error text can be found in the standard error file.

Examples

The examples presented here are all based upon the following FDT:

Field Definition Table:

Level	Name	Length	Format	Options	Flags
1	AA	2	A	DE,UQ	SP
1	AB	20	A	DE,UQ	
1	AC	20	A	DE,NC	
1	AD	4	F	NC	
1	B0			PE	
2	BA	20	A	DE,NU	SP
2	BB	4	F	NU	
2	CA	20	A	NU,MU	
2	CB	2	F	NU,MU	
2	DA	20	A	NU,MU	

It is assumed that the file is to be found in database 202, file number 181.

Example 1: Default Command Line Invocation

This example shows the simplest way of invoking the GTD Utility, namely via the command line input without any extra information sources other that the default rules.

Invoking the GTD Utility as shown below:

```
$ esqgtd WS2V14 202 181
```

will generate the following output, based upon no dialog and the default rules.

```
########################################################################
#  ESQGTD Version      : 1.4D.2
#  Date/Time           : 12-SEP-1996 17:17:49.49
#  File                : 202:181
########################################################################
continuation '\'
create cluster description CLUSTER_181 database WS2V14 file number 181
(
    create table description TABLE_181
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,COL_AA                           shortname 'AA'
      ,COL_AB                           shortname 'AB'
      ,COL_AC                           shortname 'AC'
      ,COL_AD                           shortname 'AD'
      ,primary key ( COL_SEQNO_0)
# Number of columns for this table:    5.
   )
   ,create table description TABLE_181_B0
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,COL_SEQNO_1                      seqno(1)  not null
      ,COL_BA                           shortname 'BA'
      ,COL_BB                           shortname 'BB'
      ,foreign key ( COL_SEQNO_0) references TABLE_181
      ,primary key ( COL_SEQNO_0, COL_SEQNO_1)
# Number of columns for this table:    4.
   )
   ,create table description TABLE_181_CA
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,COL_SEQNO_1                      seqno(1)  not null
      ,COL_SEQNO_2                      seqno(2)  not null
      ,COL_CA                           shortname 'CA'
      ,foreign key ( COL_SEQNO_0, COL_SEQNO_1) references TABLE_181_B0
```

```
      ,primary key ( COL_SEQNO_0, COL_SEQNO_1, COL_SEQNO_2)
# Number of columns for this table:    4.
   )
   ,create table description TABLE_181_CB
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,COL_SEQNO_1                      seqno(1)  not null
      ,COL_SEQNO_2                      seqno(2)  not null
      ,COL_CB                           shortname 'CB'
      ,foreign key ( COL_SEQNO_0, COL_SEQNO_1) references TABLE_181_B0
      ,primary key ( COL_SEQNO_0, COL_SEQNO_1, COL_SEQNO_2)
# Number of columns for this table:    4.
   )
   ,create table description TABLE_181_DA
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,COL_SEQNO_1                      seqno(1)  not null
      ,COL_SEQNO_2                      seqno(2)  not null
      ,COL_DA                           shortname 'DA'
      ,foreign key ( COL_SEQNO_0, COL_SEQNO_1) references TABLE_181_B0
      ,primary key ( COL_SEQNO_0, COL_SEQNO_1, COL_SEQNO_2)
# Number of columns for this table:    4.
   )
);
```

Example 2: Command Line Invocation With a Systrans File

```
$ esqgtd -t systrans.file WS2V14 202 181
```

The following NATURAL DDM SYSTRANS file was used in this example:

```
*H**ANAT22011997022511271270VMS/AXP         1
*C**                                SYSTEM  CITY_GUIDE                      V S
*D01NAT2201V SYSTEM  CITY_GUIDE                      KJO                     S
*D02            19970202154500019951025154500000000001061
*D03OVMS/AXP
*D040020200181
*S** 1AASTATES_ABBREVIATION             A0020 D
*S** 1ABSTATES_STATE_NAME               A0200 D
*S** 1ACSTATES_CAPITAL                  A0200 D
*S** 1ADSTATES_POPULATION               I0040
*S**P1B0CITIES                           0000
*S** 2BACITIES_CITY_NAME                A0200ND
*S** 2BBCITIES_POPULATION               I0040N
*S**M2CABUILDINGS_BUILDING_NAME         A0200N
*S**M2CBBUILDINGS_HEIGHT                I0020N
*S**M2DAPLACES_PLACE_NAME               A0200N
*S**P1X1X1-1                            A0220 S
*S***      -------- SOURCE FIELD(S) -------
*S***      CITIES_CITY_NAME(1-20)
*S***      STATES_ABBREVIATION(1-2)
*S********DDM OUTPUT TERMINATED******
*E
```

Invoking the GTD Utility as follows:

```
$ define/user sys$output output_file
$ esqgtd -f -t systrans.file WS2V14 202 181
```

will produce the following output:

```
##############################################################################
#  ESQGTD Version      : 1.4D.2
#  Date/Time           : 12-SEP-1996 17:18:13.58
#  File                : 202:181
##############################################################################
continuation '\'
create cluster description CLUSTER_181 database WS2V14 file number 181
(
  create table description CITY_GUIDE
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,STATES_ABBREVIATION              shortname 'AA'
      ,STATES_STATE_NAME                shortname 'AB'
      ,STATES_CAPITAL                   shortname 'AC'
      ,STATES_POPULATION                shortname 'AD'
      ,primary key ( COL_SEQNO_0)
# Number of columns for this table:    5.
   )
   ,create table description CITY_GUIDE_CITIES
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,COL_SEQNO_1                      seqno(1)  not null
      ,CITIES_CITY_NAME                 shortname 'BA'
      ,CITIES_POPULATION                shortname 'BB'
      ,foreign key ( COL_SEQNO_0) references CITY_GUIDE
      ,primary key ( COL_SEQNO_0, COL_SEQNO_1)
# Number of columns for this table:    4.
   )
   ,create table description CITY_GUIDE_BUILDINGS_BUILDING_NAME
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,COL_SEQNO_1                      seqno(1)  not null
      ,COL_SEQNO_2                      seqno(2)  not null
      ,BUILDINGS_BUILDING_NAME          shortname 'CA'
      ,foreign key ( COL_SEQNO_0, COL_SEQNO_1) references CITY_GUIDE_CI\
TIES
      ,primary key ( COL_SEQNO_0, COL_SEQNO_1, COL_SEQNO_2)
# Number of columns for this table:    4.
   )
   ,create table description CITY_GUIDE_BUILDINGS_HEIGHT
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,COL_SEQNO_1                      seqno(1)  not null
      ,COL_SEQNO_2                      seqno(2)  not null
      ,BUILDINGS_HEIGHT                 shortname 'CB'
      ,foreign key ( COL_SEQNO_0, COL_SEQNO_1) references CITY_GUIDE_CI\
```

```
TIES
      ,primary key ( COL_SEQNO_0, COL_SEQNO_1, COL_SEQNO_2)
# Number of columns for this table:    4.
   )
   ,create table description CITY_GUIDE_PLACES_PLACE_NAME
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,COL_SEQNO_1                      seqno(1)  not null
      ,COL_SEQNO_2                      seqno(2)  not null
      ,PLACES_PLACE_NAME                shortname 'DA'
      ,foreign key ( COL_SEQNO_0, COL_SEQNO_1) references CITY_GUIDE_CI\
TIES
      ,primary key ( COL_SEQNO_0, COL_SEQNO_1, COL_SEQNO_2)
# Number of columns for this table:    4.
   )
);
```

Example 3: Input File Usage (Default Rules)

The following input file was used in this example:

```
DATABASE 202 WS2V14
FILE 181
```

Invoking the GTD Utility as follows:

```
$ define/user sys$output output_file
$ define/user sys$input input_file
$ esqgtd
```

will produce in this case, the same output as example #1.

Example 4: Input file with Natural Systrans DDM File

In this example, the following input file was used. It references the same NATURAL SYSTRANS DDM file as above:

```
DATABASE 202 WS2V14
FILE 181
SYSTRANS systrans.file
```

Invoking the GTD Utility as follows:

```
$ define/user sys$output output_file
$ define/user sys$input input_file
$ esqgtd
```

will produce in this case, the same output as example #2.

Example 5: Input File With Directives

In this example the following input file was used:

```
DATABASE 202 WS2V14
FILE 181
B0 SUBTABLE LEVEL_1_TABLE
CA,CB SUBTABLE LEVEL_2_TABLE
SUPPRESS DA
```

Invoking the GTD Utility as follows:

```
$ define/user sys$output output_file
$ define/user sys$input input_file
$ esqgtd
```

will produce in this case, the following output:

```
#########################################################################
#  ESQGTD Version      : 1.4D.2
#  Date/Time           : 12-SEP-1996 17:18:24.85
#  File                : 202:181
#########################################################################
continuation '\'
create cluster description CLUSTER_181 database WS2V14 file number 181
(
    create table description TABLE_181
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,COL_AA                           shortname 'AA'
      ,COL_AB                           shortname 'AB'
      ,COL_AC                           shortname 'AC'
      ,COL_AD                           shortname 'AD'
      ,primary key ( COL_SEQNO_0)
# Number of columns for this table:    5.
   )
   ,create table description LEVEL_1_TABLE
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,COL_SEQNO_1                      seqno(1)  not null
      ,COL_BA                           shortname 'BA'
      ,COL_BB                           shortname 'BB'
      ,foreign key ( COL_SEQNO_0) references TABLE_181
      ,primary key ( COL_SEQNO_0, COL_SEQNO_1)
# Number of columns for this table:    4.
   )
   ,create table description LEVEL_2_TABLE
   (
       COL_SEQNO_0                      seqno(0)  not null
      ,COL_SEQNO_1                      seqno(1)  not null
      ,COL_SEQNO_2                      seqno(2)  not null
      ,COL_CA                           shortname 'CA'
      ,COL_CB                           shortname 'CB'
      ,foreign key ( COL_SEQNO_0, COL_SEQNO_1) references LEVEL_1_TABLE
      ,primary key ( COL_SEQNO_0, COL_SEQNO_1, COL_SEQNO_2)
# Number of columns for this table:    5.
   )
);
```

4

LOGGING FACILITIES

Introduction

Numerous actions and operations performed by the ADABAS SQL Server can be logged. The act of logging records that a particular event has occurred and also records additional information associated with the event. Events are recorded chronologically. This record of events can then be examined at a later date. Different types of events can be recorded simultaneously and the types of events to be logged can be specified as required.

Logging can be activated before executing the client application or before starting the server. Logging for the server can be activated without having to interrupt the server by activating the logging on the client side. The client stub (ESQLNK) informs the server about the additional logging parameters. These additional logging parameters are valid for the current client session only.

Logging Facilities are always available, even in the non-trace version. However, their use will affect the overall system performance. In addition, certain logging facilities are extremely detailed and liable to produce very large amounts of information.

Logging can provide information necessary for performance analysis, problem diagnosis and system access monitoring.

For example, the Explain Logging Facility provides detailed information about the access paths used when executing DML statements. Using this information, it may be possible to construct statements which are more efficient, or to introduce appropriate indices to improve performance. The Client/Server Communication Logging can be used to investigate any problems experienced in a particular client/server link. Finally, the Security Logging can be used to monitor changing privileges.

Logging is activated by setting the symbol ESQLG appropriately. By setting this symbol, a particular type of logging with options, if applicable, can be activated. Furthermore, combinations of types of logging can be specified. The character “*” activates the complete logging.

Logging Facilities Overview

The following tables outline the types of logging available and the string setting for ESQLG required to activate it:

Logging Facilities Activated by the Symbol (ESQLG)

Client/Server Characteristics Logging	MODE
Brief SQL Command Logging	
Client and Server side	CMD
Client side only	CCMD
Server side only	SCMD
Static SQL Command Logging	
80-Character Block Format	STATEMENT
Continuous Stream Format	STATEMENT=STREAM
Dynamic SQL Command Logging	
80-Character Block Format	DYNHVS
Continuous Stream Format	DYNHVS=STREAM
Client/Server Communication Logging	CSC
Server Session Logging	SES
Schema Identifier Logging	SCHEMAIDENT
User Exit Logging	USEREXIT
Elapsed Time Logging	
without component time	TIME
with component time	TIME = ALL
Sort Logging	SORT
ADABAS Utility Logging	ADU
Security Logging	
User authentication check logging	AUTH
Simple statement logging	SEC
ALL resultant actions logged	SEC=ALL
ESQLNK Call Logging	ENTRY
Explain Logging	EXPL

Logging Facilities Activated by Parameter File Specification

ADABAS Command Logging

How to activate Logging

With the exception of the ADABAS Command Logging (which is controlled via the ADABAS SQL Parameter file and is discussed later in this chapter), all other logging is controlled via the ESQLG symbol.

Logging is activated by assigning a string to ESQLG containing one or more keywords, specifying the entity for logging.

Logging may be activated on both the **client side** and the **server side**. To activate logging on the **client side**, the symbol ESQLG must be defined in the local symbol table.

Example:

```
$ ESQLG := CMD, CSC            turns on the brief SQL command logging,
                               and the client/server communications logging.
```

To activate logging on the **server side**, the symbol ESQLG must be defined as a global symbol in a file called startup.com located in the server directory (ESQ$SRVDIR).

Example: The contents of ESQ$SRVDIR:startup.com

```
$ ESQLG:==  CMD, CSC
```

Logging Output

The Logging Facility logs information from the either the client or the server standpoint, or both, depending on the value assigned to ESQLG. Therefore, the destination of the logging information will depend on who is performing the work:

- the client or
- the server

and the client/server configuration being used:

- LINKED-IN Mode
  When logging is used in LINKED-IN mode, the work performed by both the client and the server is logged to the standard output of the client.
- Client/Server Mode
  When logging is used in Client/Server mode, the work performed by the client is logged to the client's standard output and the work performed by the server is logged to the server log file ESQ$SRVDIR/esqsrv.log.

Logging Facilities Reference

The following sections describe all types of logging. You will find information about:

- actions and information to be logged
- examples of how to activate and run logging

Most of the following examples were executed in LINKED-IN mode. This has the advantage that the server logging also appears on the client side. For examples executed in Client/Server mode, the symbol ESQLG has to be used to get the server's logging output.

Client/Server Characteristics Logging

This logging gives information about:

- Server, client stub, precompiler version
- Trace availability
- Thread mode (single/multi user linked-in)
- Application was linked/loaded with esq.o, ESQLNK or ESQTHS

Example:

```
$ esqlg:=mode
$ show symbol esqlg
  ESQLG = "MODE"
$ esqint
%ESQINT-I-STARTED, 22-NOV-1996 09:49:44, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

%ESQRTS-I-ESQINFO, ESQLG is enabled for 'MODE'
%ESQRTS-I-ESQINFO, MT/TR trace is NOT available
%ESQRTS-I-ESQINFO, ESQ application was linked/loaded with ESQLNK V1.4B.2.1
%ESQRTS-I-ESQINFO, session/process memory is reliable between requests
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select * from information_schema.tables;

 TABLE_SCHEMA                    TABLE_NAME                        TABLE_TYPE

 INFORMATION_SCHEMA              SERVER_INFO                       VIEW
 INFORMATION_SCHEMA              INFORMATION_SCHEMA_CATALOG_NAME   VIEW
 INFORMATION_SCHEMA              SCHEMATA                          VIEW
 INFORMATION_SCHEMA              CLUSTERS                          VIEW
 INFORMATION_SCHEMA              TABLES                            VIEW
 INFORMATION_SCHEMA              BASE_TABLES                       VIEW
...
esqint: q
%ESQINT-I-TERMINATED, 22-NOV-1996 09:50:59
```

Brief SQL Command Logging

The brief SQL Command Logging is available on both, client and server side. The log lines are produced at the end of each SQL request so that the response code is visible.

- Following logging lines are produced on client side:

```
%ESQRTS-I-ESQINFO, Clt: PREPARE   cnt=    7/   3    rsp=0
                        |                 |    |        |
                        |                 |    |        |
                        |                 |    |        response code
                        |                 |    |
                        |                 |    C/S communication count
                        |                 |
                        |                 SQL request count
                        |
                        SQL statement
```

- Following logging lines are produced on server side:

```
30-NOV 14:00:07.58 Th0: PREPARE  -cnt=  7/    2 usr=wsesq7:JOHN   rsp=0
                        |        |      |     |     |      |          |
                        |        |      |     |     |      |          |
                        |        |      |     |     |      |          response
                        |        |      |     |     |      |          code
                        |        |      |     |     |      |
                        |        |      |     |     |      CONNECT client
                        |        |      |     |     |      user ID
                        |        |      |     |     |
                        |        |      |     |     Client node name
                        |        |      |     |
                        |        |      |     C/S communication count
                        |        |      |
                        |        |      SQL request count
                        |        |
                        |        + : Meta program found in the catalog buffer
                        |        - : Meta program loaded into the catalog buff.
                        |        = : Meta program generated and loaded
                        |
                        SQL statement
```

- Especially for the SQL statement FETCH (with MULTIFETCH) following logging lines are produced:

```
30-NOV 14:00:08.37 Th0: mFETCH (rf=16) +cnt=  12/   7 usr=wsesq7:JOHN rsp=0
```

MULTIFETCH was executed on server side. 16 records were fetched and transported to the client. ”rf” stands for ”records fetched”.

```
%ESQRTS-I-ESQINFO, Clt: mFETCH (rf=16)  cnt=  12/   8 rsp=0
```

MULTIFETCH was executed on client side. 16 records were received by the client. First record is offered to the application.

```
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=2)   cnt=  13/   8 rsp=0
```

Single FETCH was executed on client side only. The second record on client side is offered to the application. ”rp” stands for ”record position”.

Example:

```
$ esqlg:=cmd
$ show symbol esqlg
  ESQLG = ”CMD”
$ esqint
%ESQINT-I-STARTED, 22-NOV-1996 09:55:48, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

%ESQRTS-I-ESQINFO, ESQLG is enabled for ’CMD’
%ESQRTS-I-ESQINFO, Clt: CONNECT         cnt=   1/   2 rsp=0
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select * from information_schema.tables;
%ESQRTS-I-ESQINFO, Clt: PREPARE         cnt=   7/   3 rsp=0
%ESQRTS-I-ESQINFO, Clt: DESCRIBE        cnt=   8/   4 rsp=0
%ESQRTS-I-ESQINFO, Clt: DESCRIBE        cnt=   9/   5 rsp=0
%ESQRTS-I-ESQINFO, Clt: DYN DECL CURSO  cnt=  10/   6 rsp=0
%ESQRTS-I-ESQINFO, Clt: OPEN CURSOR     cnt=  11/   7 rsp=0
%ESQRTS-I-ESQINFO, Clt: mFETCH (rf=16)  cnt=  12/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=2)   cnt=  13/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=3)   cnt=  14/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=4)   cnt=  15/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=5)   cnt=  16/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=6)   cnt=  17/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=7)   cnt=  18/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=8)   cnt=  19/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=9)   cnt=  20/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=10)  cnt=  21/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=11)  cnt=  22/   8 rsp=0
```

```
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=12)  cnt=  23/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=13)  cnt=  24/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=14)  cnt=  25/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=15)  cnt=  26/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=16)  cnt=  27/   8 rsp=0
%ESQRTS-I-ESQINFO, Clt: mFETCH (rf=12)  cnt=  28/   9 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=2)   cnt=  29/   9 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=3)   cnt=  30/   9 rsp=0
%ESQRTS-I-ESQINFO, Clt: sFETCH (rp=4)   cnt=  31/   9 rsp=0

 TABLE_SCHEMA                    TABLE_NAME                       TABLE_TYPE

 INFORMATION_SCHEMA              SERVER_INFO                      VIEW
 INFORMATION_SCHEMA              INFORMATION_SCHEMA_CATALOG_NAME  VIEW
 INFORMATION_SCHEMA              SCHEMATA                         VIEW
 INFORMATION_SCHEMA              CLUSTERS                         VIEW
 INFORMATION_SCHEMA              TABLES                           VIEW
 ...
esqint: q
%ESQRTS-I-ESQINFO, Clt: CLOSE CURSOR    cnt=  32/  10 rsp=0
%ESQRTS-I-ESQINFO, Clt: DISCONNECT      cnt=  33/  11 rsp=0
%ESQINT-I-TERMINATED, 22-NOV-1996 09:56:12
%ESQRTS-I-ESQINFO, Clt: DISCONNECT ALL  cnt=  34/  11 rsp=0
```

Static SQL Command-String Logging

The Static SQL Command String Logging produces log lines containing the complete SQL statements. This logging type is available for runtime and for precompilation time. Two logging formats are supported:

- 80-character Block Format (ESQLG:=STATEMENT)
- Continuous Stream Format (ESQLG:=STATEMENT=STREAM)

Example:

```
$ esqlg:=statement
$ show symbol esqlg
  ESQLG = "STATEMENT"
$ esqint
%ESQINT-I-STARTED, 22-NOV-1996 10:10:07, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

%ESQRTS-I-ESQINFO, ESQLG is enabled for 'STATEMENT'
%ESQRTS-I-ESQINFO, CONNECT TO DEFAULT USER : "&sql_user[0]" CHARACTER(,32,0,0,8
_01 ,33)
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select * from information_schema.tables;
%ESQRTS-I-ESQINFO, PREPARE : "&sql_id[0]" CHARACTER(,32,0,0,46,33) FROM : "&sma
_01 ll_sql_statement[0]" CHARACTER(,128,0,0,47,129)
%ESQRTS-I-ESQINFO, DESCRIBE : "&sql_id[0]" CHARACTER(,32,0,0,55,33) INTO INPUT
_01 : "if_sql_input_sqlda" ( "if_sql_input_sqlda" POINTER ( ,0,0,0,0))
%ESQRTS-I-ESQINFO, DESCRIBE : "&sql_id[0]" CHARACTER(,32,0,0,56,33) INTO OUTPUT
_01  : "if_sql_output_sqlda" ( "if_sql_output_sqlda" POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, DECLARE : "&sql_cursor[0]" CHARACTER(,18,0,0,87,19) CURSOR F
_01 OR : "&sql_id[0]" CHARACTER(,32,0,0,88,33)
%ESQRTS-I-ESQINFO, OPEN : "&sql_cursor[0]" CHARACTER(,18,0,0,89,19)
%ESQRTS-I-ESQINFO, FETCH : "&sql_cursor[0]" CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : "if_sql_output_sqlda" ( "if_sql_output_sqlda" POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : "&sql_cursor[0]" CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : "if_sql_output_sqlda" ( "if_sql_output_sqlda" POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : "&sql_cursor[0]" CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : "if_sql_output_sqlda" ( "if_sql_output_sqlda" POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : "&sql_cursor[0]" CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : "if_sql_output_sqlda" ( "if_sql_output_sqlda" POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : "&sql_cursor[0]" CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : "if_sql_output_sqlda" ( "if_sql_output_sqlda" POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : "&sql_cursor[0]" CHARACTER(,18,0,0,91,19) USING DESC
```

```
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))
%ESQRTS-I-ESQINFO, FETCH : ”&sql_cursor[0]” CHARACTER(,18,0,0,91,19) USING DESC
_01 RIPTOR : ”if_sql_output_sqlda” ( ”if_sql_output_sqlda” POINTER ( ,0,0,0,1))

 TABLE_SCHEMA                    TABLE_NAME                       TABLE_TYPE

 INFORMATION_SCHEMA              SERVER_INFO                      VIEW
 INFORMATION_SCHEMA              INFORMATION_SCHEMA_CATALOG_NAME  VIEW
 INFORMATION_SCHEMA              SCHEMATA                         VIEW
 INFORMATION_SCHEMA              CLUSTERS                         VIEW
 INFORMATION_SCHEMA              TABLES                           VIEW
...
esqint: q
%ESQRTS-I-ESQINFO, CLOSE : ”&sql_cursor[0]” CHARACTER(,18,0,0,90,19)
%ESQRTS-I-ESQINFO, DISCONNECT CURRENT
%ESQINT-I-TERMINATED, 22-NOV-1996 10:10:20
%ESQRTS-I-ESQINFO, DISCONNECT ALL
```

Dynamic SQL Command-String Logging

The Static SQL Command String Logging produces log lines containing the dynamic SQL statements passed on to a PREPARE or EXECUTE IMMEDIATE statement. Two logging formats are supported:

- 80-character Block Format (ESQLG:=DYNHVS)
- Continuous Stream Format (ESQLG:=DYNHVS=STREAM)

Example:

```
$ esqlg:=dynhvs
$ show symbol esqlg
  ESQLG = "DYNHVS"
$ esqint
%ESQINT-I-STARTED, 22-NOV-1996 10:03:46, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

%ESQRTS-I-ESQINFO, ESQLG is enabled for 'DYNHVS'
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select * from information_schema.tables;
%ESQRTS-I-ESQINFO, HV01: sql_i
%ESQRTS-I-ESQINFO, HV02:  select * from information_schema.table

 TABLE_SCHEMA                    TABLE_NAME                        TABLE_TYPE

 INFORMATION_SCHEMA              SERVER_INFO                       VIEW
 INFORMATION_SCHEMA              INFORMATION_SCHEMA_CATALOG_NAME   VIEW
 INFORMATION_SCHEMA              SCHEMATA                          VIEW
 INFORMATION_SCHEMA              CLUSTERS                          VIEW
 INFORMATION_SCHEMA              TABLES                            VIEW
 ...
esqint: q
%ESQINT-I-TERMINATED, 22-NOV-1996 10:04:03
```

Client/Server Communication Logging

The Client/Server Communication Logging gives information about messages sent between the ADABAS SQL Server client and the ADABAS SQL Server. It can be enabled on client and/or on server side and applies only to real client/server mode (not in linked-in mode):

- Following logging lines are produced on client side for one SQL request within one session:

```
%ESQRTS-I-ESQINFO, Clt: sending an OPEN to WSESQ7:ESQV14 ...
%ESQRTS-I-ESQINFO, Clt: ONS cid=0 sndl=0 rcvl=0 retl=0
%ESQRTS-I-ESQINFO, Clt: ONS cid=400842E8 sndl=0 rcvl=0 retl=0
%ESQRTS-I-ESQINFO, Clt: ... OPEN was successful
%ESQRTS-I-ESQINFO, Clt: sending a CSCI request ...
%ESQRTS-I-ESQINFO, Clt: QNS cid=400842E8 sndl=1024 rcvl=338 retl=0
%ESQRTS-I-ESQINFO, Clt: QNS cid=400842E8 sndl=1024 rcvl=338 retl=338
%ESQRTS-I-ESQINFO, Clt: ... received CSCI reply
%ESQRTS-I-ESQINFO, Clt: sending a CLOSE ...
%ESQRTS-I-ESQINFO, Clt: CNS cid=400842E8 sndl=0 rcvl=0 retl=310
%ESQRTS-I-ESQINFO, Clt: CNS cid=400842E8 sndl=0 rcvl=0 retl=310
%ESQRTS-I-ESQINFO, Clt: ... CLOSE was successful

%ESQRTS-I-ESQINFO, Clt: QNS cid=400842E8 sndl=1024 rcvl=338 retl=0
                        |||     |             |         |        |
                        |||     |             |         |        returned
                        |||     |             |         |        length
                        |||     |             |         |
                        |||     |             |         receive-buffer length
                        |||     |             |
                        |||     |             send-buffer length
                        |||     |
                        |||     connection ID
                        |||
                        ||Mode      S : synchronous
                        ||
                        |Type       N : native
                        |
                        Function    O : open
                                    Q : request
                                    C : close
```

- Following logging lines are produced on server side for one SQL request within one session:

```
01-DEC 10:47:47.51 Th2: f=ION c=1,        ,         l=0,0,0
01-DEC 10:47:52.30 Th2: waiting for a request ...
01-DEC 10:50:41.60 Th2: f=LOS c=0,grg     ,WSESQ7   l=0,8240,0
01-DEC 10:50:41.60 Th2: accept OPEN
01-DEC 10:50:41.61 Th2: f=AOS c=40030A08,grg     ,WSESQ7   l=0,8240,0
01-DEC 10:50:41.61 Th2: waiting for a request ...
01-DEC 10:50:41.63 Th2: f=LQS c=40030A08,grg     ,WSESQ7   l=0,8240,1024
01-DEC 10:50:45.89 Th2: sending the reply
01-DEC 10:50:45.90 Th2: f=PQS c=40030A08,grg     ,WSESQ7   l=338,0,0
01-DEC 10:50:45.90 Th2: waiting for a request ...
01-DEC 10:51:01.33 Th2: f=LCS c=40030A08,grg     ,WSESQ7   l=0,8240,0
01-DEC 10:51:01.33 Th2: accept CLOSE
01-DEC 10:51:01.34 Th2: f=PCS c=40030A08,grg     ,WSESQ7   l=0,8240,0

01-DEC 10:50:41.63 Th2: f=LQS c=40030A08,grg     ,WSESQ7   l=0,8240,1024
                         |||   |        |        |          | |    |
                         |||   |        |        |          | |    returned
                         |||   |        |        |          | |    length
                         |||   |        |        |          | |
                         |||   |        |        |          | receive-buffer
                         |||   |        |        |          | length
                         |||   |        |        |          |
                         |||   |        |        |          send-buffer length
                         |||   |        |        |
                         |||   |        |        client node name
                         |||   |        |
                         |||   |        OS client user-ID
                         |||   |
                         |||   connection ID
                         |||
                         ||Mode     S : synchronous
                         ||
                         |Type      O : open
                         |          Q : requet
                         |          C : close
                         |
                         Function   I : initialise
                                    L : listen
                                    A : accept open
                                    P : reply / close
```

Example:

```
$ esqlg:=csci
$ show symbol esqlg
  ESQLG == "CSCI"
$ esqset ESQ140AT
$ type esq$srvdir:startup.com
          $ esqlg:== csci
$ esqstart
%RUN-S-PROC_ID, identification of created process is 00000E45
%ESQSTART-I-STARTDET, Server ESQ140AT started as detached process
$ esqint
%ESQRUN-I-PAR, Using parameter file ESQRUN.PAR
%ESQINT-I-STARTED, 14-JAN-1997 13:41:28, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:
%ESQRTS-I-ESQINFO, ESQLG is enabled for 'CSCI'
%ESQRTS-I-ESQINFO, Clt: CSCI ident 1/1
%ESQRTS-I-ESQINFO, Clt: sending an OPEN to SCALOS:ESQ140AT ...
%ESQRTS-I-ESQINFO, Clt: ONN cid=0 sndl=0 rcvl=0 retl=0
%ESQRTS-I-ESQINFO, Clt: ONN cid=2FEB48 sndl=0 rcvl=0 retl=0
%ESQRTS-I-ESQINFO, Clt: ... OPEN was successful
%ESQRTS-I-ESQINFO, Clt: sending a CSCI request ...
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=1122 rcvl=338 retl=0
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=1122 rcvl=338 retl=338
%ESQRTS-I-ESQINFO, Clt: ... received CSCI reply
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select * from information_schema.tables;
%ESQRTS-I-ESQINFO, Clt: sending a CSCI request ...
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=432 rcvl=342 retl=338
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=432 rcvl=342 retl=342
%ESQRTS-I-ESQINFO, Clt: ... received CSCI reply
%ESQRTS-I-ESQINFO, Clt: sending a CSCI request ...
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=306 rcvl=1532 retl=342
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=306 rcvl=1532 retl=1532
%ESQRTS-I-ESQINFO, Clt: ... received CSCI reply
%ESQRTS-I-ESQINFO, Clt: sending a CSCI request ...
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=306 rcvl=1532 retl=1532
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=306 rcvl=1532 retl=1532
%ESQRTS-I-ESQINFO, Clt: ... received CSCI reply
%ESQRTS-I-ESQINFO, Clt: sending a CSCI request ...
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=304 rcvl=310 retl=1532
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=304 rcvl=310 retl=310
%ESQRTS-I-ESQINFO, Clt: ... received CSCI reply
%ESQRTS-I-ESQINFO, Clt: sending a CSCI request ...
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=264 rcvl=310 retl=310
```

```
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=264 rcvl=310 retl=310
%ESQRTS-I-ESQINFO, Clt: ... received CSCI reply
%ESQRTS-I-ESQINFO, Clt: sending a CSCI request ...
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=324 rcvl=2118 retl=310
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=324 rcvl=2118 retl=2118
%ESQRTS-I-ESQINFO, Clt: ... received CSCI reply
%ESQRTS-I-ESQINFO, Clt: sending a CSCI request ...
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=324 rcvl=2118 retl=2118
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=324 rcvl=2118 retl=2118
%ESQRTS-I-ESQINFO, Clt: ... received CSCI reply

 TABLE_SCHEMA                    TABLE_NAME                      TABLE_TYPE

 INFORMATION_SCHEMA              SERVER_INFO                     VIEW
 INFORMATION_SCHEMA              INFORMATION_SCHEMA_CATALOG_NAME VIEW
 INFORMATION_SCHEMA              SCHEMATA                        VIEW
 INFORMATION_SCHEMA              CLUSTERS                        VIEW
 INFORMATION_SCHEMA              TABLES                          VIEW
...
esqint: quit
%ESQRTS-I-ESQINFO, Clt: sending a CSCI request ...
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=264 rcvl=310 retl=2118
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=264 rcvl=310 retl=310
%ESQRTS-I-ESQINFO, Clt: ... received CSCI reply
%ESQRTS-I-ESQINFO, Clt: sending a CSCI request ...
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=264 rcvl=310 retl=310
%ESQRTS-I-ESQINFO, Clt: QNS cid=2FEB48 sndl=264 rcvl=310 retl=310
%ESQRTS-I-ESQINFO, Clt: ... received CSCI reply
%ESQRTS-I-ESQINFO, Clt: sending a CLOSE ...
%ESQRTS-I-ESQINFO, Clt: CNS cid=2FEB48 sndl=0 rcvl=0 retl=310
%ESQRTS-I-ESQINFO, Clt: CNS cid=2FEB48 sndl=0 rcvl=0 retl=310
%ESQRTS-I-ESQINFO, Clt: ... CLOSE was successful
%ESQINT-I-TERMINATED, 14-JAN-1997 14:09:00
$ esqlog
Log file of ESQ server ESQ140AT:
%ESQSRV-I-EXECUTE, Executing SAG$ROOT:[ESQ.ESQ140AT]STARTUP.COM;1 ...
%ESQINI-I-ESQINFO, ESQLG is enabled for 'CSCI'
%ESQINI-I-ESQINFO, Software AG ADABAS SQL Server starting up on ALPHA
AXP/OpenVMS
%ESQINI-I-ESQINFO, Version:         1.4B.2.1
%ESQINI-I-ESQINFO, Startup time:    14-JAN-1997 14:07:37
%ESQINI-I-ESQINFO, Server name:     ESQ140AT on node SCALOS
%ESQINI-I-ESQINFO, Server type:     CSCI
%ESQINI-I-ESQINFO, Catalog files:   DB 222, Files 30-32
%ESQINI-I-ESQINFO, Server pid:      3669
%ESQINI-I-ESQINFO, Server image:    SAG$ROOT:[ESQ.V142]ESQINI.EXE;1
%ESQINI-I-ESQINFO, Thread image:    SAG$ROOT:[ESQ.V142]ESQSRV.EXE;1
```

```
%ESQINI-I-ESQINFO, Thread library:  SAG$ROOT:[ESQ.V142]ESQTHS142.EXE
%ESQINI-I-ESQINFO, Threads:         3
%ESQINI-I-ESQINFO, Sessions/Thread: 1
%ESQSRV-I-ESQINFO, ESQLG is enabled for 'CSCI'
14-JAN 14:07:38.56 Th1: initializing C/S Interface
%ESQSRV-I-ESQINFO, Th1: compile-time CSC_VERSION=1, run-time csc.ident=1
14-JAN 14:07:39.09 Th1: f=ION c=1,        ,         l=0,0,0
14-JAN 14:07:39.09 Server ESQ140AT Thread 1 pid 3671 up and running
14-JAN 14:07:39.09 Th1: waiting for a request ...
%ESQSRV-I-ESQINFO, ESQLG is enabled for 'CSCI'
14-JAN 14:07:41.40 Th2: initializing C/S Interface
%ESQSRV-I-ESQINFO, Th2: compile-time CSC_VERSION=1, run-time csc.ident=1
14-JAN 14:07:41.51 Th2: f=ION c=1,        ,         l=0,0,0
14-JAN 14:07:41.51 Server ESQ140AT Thread 2 pid 3672 up and running
14-JAN 14:07:41.51 Th2: waiting for a request ...
%ESQSRV-I-ESQINFO, ESQLG is enabled for 'CSCI'
14-JAN 14:07:42.49 Th3: initializing C/S Interface
%ESQSRV-I-ESQINFO, Th3: compile-time CSC_VERSION=1, run-time csc.ident=1
14-JAN 14:07:42.60 Th3: f=ION c=1,        ,         l=0,0,0
14-JAN 14:07:42.60 Server ESQ140AT Thread 3 pid 3673 up and running
14-JAN 14:07:42.60 Th3: waiting for a request ...
14-JAN 14:07:48.34 Server ESQ140AT up and running
14-JAN 14:08:48.71 Th3: f=LOS c=0,SAGDBA  ,SCALOS   l=0,8240,0
14-JAN 14:08:48.72 Th3: accept OPEN
14-JAN 14:08:48.72 Th3: f=AOS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,0
14-JAN 14:08:48.72 Th3: waiting for a request ...
14-JAN 14:08:48.73 Th3: f=LQS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,1122
%ESQSRV-I-ESQINFO, ESQLG is enabled for 'CSCI'
14-JAN 14:08:49.28 Th3: sending the reply
14-JAN 14:08:49.29 Th3: f=PQS c=4005E0,SAGDBA  ,SCALOS   l=338,0,0
14-JAN 14:08:49.29 Th3: waiting for a request ...
14-JAN 14:08:56.84 Th3: f=LQS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,432
14-JAN 14:08:57.46 Th3: sending the reply
14-JAN 14:08:57.46 Th3: f=PQS c=4005E0,SAGDBA  ,SCALOS   l=342,0,0
14-JAN 14:08:57.46 Th3: waiting for a request ...
14-JAN 14:08:57.47 Th3: f=LQS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,306
14-JAN 14:08:57.48 Th3: sending the reply
14-JAN 14:08:57.48 Th3: f=PQS c=4005E0,SAGDBA  ,SCALOS   l=1532,0,0
14-JAN 14:08:57.48 Th3: waiting for a request ...
14-JAN 14:08:57.49 Th3: f=LQS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,306
14-JAN 14:08:57.50 Th3: sending the reply
14-JAN 14:08:57.50 Th3: f=PQS c=4005E0,SAGDBA  ,SCALOS   l=1532,0,0
14-JAN 14:08:57.50 Th3: waiting for a request ...
14-JAN 14:08:57.51 Th3: f=LQS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,304
14-JAN 14:08:57.52 Th3: sending the reply
14-JAN 14:08:57.52 Th3: f=PQS c=4005E0,SAGDBA  ,SCALOS   l=310,0,0
14-JAN 14:08:57.52 Th3: waiting for a request ...
```

```
14-JAN 14:08:57.53 Th3: f=LQS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,264
14-JAN 14:08:57.55 Th3: sending the reply
14-JAN 14:08:57.55 Th3: f=PQS c=4005E0,SAGDBA  ,SCALOS   l=310,0,0
14-JAN 14:08:57.55 Th3: waiting for a request ...
14-JAN 14:08:57.56 Th3: f=LQS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,324
14-JAN 14:08:58.15 Th3: sending the reply
14-JAN 14:08:58.15 Th3: f=PQS c=4005E0,SAGDBA  ,SCALOS   l=2118,0,0
14-JAN 14:08:58.15 Th3: waiting for a request ...
14-JAN 14:08:58.18 Th3: f=LQS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,324
14-JAN 14:08:58.28 Th3: sending the reply
14-JAN 14:08:58.28 Th3: f=PQS c=4005E0,SAGDBA  ,SCALOS   l=2118,0,0
14-JAN 14:08:58.28 Th3: waiting for a request ...
14-JAN 14:09:00.08 Th3: f=LQS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,264
14-JAN 14:09:00.08 Th3: sending the reply
14-JAN 14:09:00.08 Th3: f=PQS c=4005E0,SAGDBA  ,SCALOS   l=310,0,0
14-JAN 14:09:00.08 Th3: waiting for a request ...
14-JAN 14:09:00.09 Th3: f=LQS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,264
14-JAN 14:09:00.12 Th3: sending the reply
14-JAN 14:09:00.12 Th3: f=PQS c=4005E0,SAGDBA  ,SCALOS   l=310,0,0
14-JAN 14:09:00.12 Th3: waiting for a request ...
14-JAN 14:09:00.13 Th3: f=LCS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,0
14-JAN 14:09:00.13 Th3: accept CLOSE
14-JAN 14:09:00.13 Th3: f=PCS c=4005E0,SAGDBA  ,SCALOS   l=0,8240,0
14-JAN 14:09:00.13 Th3: waiting for a request ...
```

Session Logging on Server Side

Session Logging only works on the server side and, only if CSCI is used, for C/S communication. It gives information about:

- Opening of a client session
- Closing of a client session
- Time-out of a client session

This type of logging is also switched on with the SQL command logging (CMD).

Example:

```
$ esqlg:=ses
$ sh sym esqlg
  ESQLG = "SES"
$ esqset ESQ140AT
$ type esq$srvdir:startup.com
$ esqlg:== ses
$ esqstart
%RUN-S-PROC_ID, identification of created process is 00000C64
%ESQSTART-I-STARTDET, Server ESQ140AT started as detached process
Log file of ESQ server ESQ140AT:
%ESQSRV-I-EXECUTE, Executing SAG$ROOT:[ESQ.ESQ140AT]STARTUP.COM;2 ...
%ESQINI-I-ESQINFO, ESQLG is enabled for 'SES'
%ESQINI-I-ESQINFO, Software AG ADABAS SQL Server starting up on ALPHA
AXP/OpenVMS
%ESQINI-I-ESQINFO, Version:         1.4B.2.1
%ESQINI-I-ESQINFO, Startup time:    14-JAN-1997 14:40:06
%ESQINI-I-ESQINFO, Server name:     ESQ140AT on node SCALOS
%ESQINI-I-ESQINFO, Server type:     CSCI
%ESQINI-I-ESQINFO, Catalog files:   DB 222, Files 30-32
%ESQINI-I-ESQINFO, Server pid:      3172
%ESQINI-I-ESQINFO, Server image:    SAG$ROOT:[ESQ.V142]ESQINI.EXE;1
%ESQINI-I-ESQINFO, Thread image:    SAG$ROOT:[ESQ.V142]ESQSRV.EXE;1
%ESQINI-I-ESQINFO, Thread library:  SAG$ROOT:[ESQ.V142]ESQTHS142.EXE
%ESQINI-I-ESQINFO, Threads:         3
%ESQINI-I-ESQINFO, Sessions/Thread: 1
%ESQSRV-I-ESQINFO, ESQLG is enabled for 'SES'
14-JAN 14:40:07.05 Server ESQ140AT Thread 1 pid 3430 up and running
%ESQSRV-I-ESQINFO, ESQLG is enabled for 'SES'
14-JAN 14:40:10.13 Server ESQ140AT Thread 2 pid 3431 up and running
%ESQSRV-I-ESQINFO, ESQLG is enabled for 'SES'
14-JAN 14:40:11.22 Server ESQ140AT Thread 3 pid 3688 up and running
14-JAN 14:40:16.97 Server ESQ140AT up and running
```

```
$ esqint
%ESQRUN-I-PAR, Using parameter file ESQRUN.PAR
%ESQINT-I-STARTED, 14-JAN-1997 14:41:51, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

%ESQRTS-I-ESQINFO, ESQLG is enabled for 'SES'
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select * from information_schema.tables;

 TABLE_SCHEMA                    TABLE_NAME                      TABLE_TYPE

 INFORMATION_SCHEMA              SERVER_INFO                     VIEW
 INFORMATION_SCHEMA              INFORMATION_SCHEMA_CATALOG_NAME VIEW
 INFORMATION_SCHEMA              SCHEMATA                        VIEW
 INFORMATION_SCHEMA              CLUSTERS                        VIEW
 INFORMATION_SCHEMA              TABLES                          VIEW
...
esqint: q
%ESQINT-I-TERMINATED, 14-JAN-1997 14:42:07
$ esqlog
Log file of ESQ server ESQ140AT:
%ESQSRV-I-EXECUTE, Executing SAG$ROOT:[ESQ.ESQ140AT]STARTUP.COM;2 ...
%ESQINI-I-ESQINFO, ESQLG is enabled for 'SES'
%ESQINI-I-ESQINFO, Software AG ADABAS SQL Server starting up on ALPHA
AXP/OpenVMS
%ESQINI-I-ESQINFO, Version:         1.4B.2.1
%ESQINI-I-ESQINFO, Startup time:    14-JAN-1997 14:40:06
%ESQINI-I-ESQINFO, Server name:     ESQ140AT on node SCALOS
%ESQINI-I-ESQINFO, Server type:     CSCI
%ESQINI-I-ESQINFO, Catalog files:   DB 222, Files 30-32
%ESQINI-I-ESQINFO, Server pid:      3172
%ESQINI-I-ESQINFO, Server image:    SAG$ROOT:[ESQ.V142]ESQINI.EXE;1
%ESQINI-I-ESQINFO, Thread image:    SAG$ROOT:[ESQ.V142]ESQSRV.EXE;1
%ESQINI-I-ESQINFO, Thread library:  SAG$ROOT:[ESQ.V142]ESQTHS142.EXE
%ESQINI-I-ESQINFO, Threads:         3
%ESQINI-I-ESQINFO, Sessions/Thread: 1
%ESQSRV-I-ESQINFO, ESQLG is enabled for 'SES'
14-JAN 14:40:07.05 Server ESQ140AT Thread 1 pid 3430 up and running
%ESQSRV-I-ESQINFO, ESQLG is enabled for 'SES'
14-JAN 14:40:10.13 Server ESQ140AT Thread 2 pid 3431 up and running
%ESQSRV-I-ESQINFO, ESQLG is enabled for 'SES'
14-JAN 14:40:11.22 Server ESQ140AT Thread 3 pid 3688 up and running
14-JAN 14:40:16.97 Server ESQ140AT up and running
14-JAN 14:41:53.53 Th3: OPEN SESSION               csusr=SCALOS:SAGDBA
%ESQSRV-I-ESQINFO, ESQLG is enabled for 'SES'
14-JAN 14:42:07.28 Th3: CLOSE SESSION              csusr=SCALOS:SAGDBA
```

Schema Identifier Logging

ADABAS SQL Server schema identifier logging gives information about the schema identifier setting on the client and on the server side.

Example:

```
$ esqlg:=schemaident
$ sh sym esqlg
  ESQLG = "SCHEMAIDENT"
$ esqint
%ESQINT-I-STARTED, 14-JAN-1997 14:58:38, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

%ESQRTS-I-ESQINFO, ESQLG is enabled for 'SCHEMAIDENT'
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select * from information_schema.tables;
14-JAN 14:58:46.80 Th0: schema-ident srv=ESQ clt=ESQ
14-JAN 14:58:46.95 Th0: schema-ident srv=ESQ clt=ESQ
14-JAN 14:58:46.95 Th0: schema-ident srv=ESQ clt=ESQ
14-JAN 14:58:46.95 Th0: schema-ident srv=ESQ clt=ESQ
14-JAN 14:58:46.96 Th0: schema-ident srv=ESQ clt=ESQ
14-JAN 14:58:46.98 Th0: schema-ident srv=ESQ clt=ESQ
14-JAN 14:58:47.58 Th0: schema-ident srv=ESQ clt=ESQ

 TABLE_SCHEMA                    TABLE_NAME                        TABLE_TYPE

 INFORMATION_SCHEMA              SERVER_INFO                       VIEW
 INFORMATION_SCHEMA              INFORMATION_SCHEMA_CATALOG_NAME   VIEW
 INFORMATION_SCHEMA              SCHEMATA                          VIEW
 INFORMATION_SCHEMA              CLUSTERS                          VIEW
 INFORMATION_SCHEMA              TABLES                            VIEW
...
esqint: quit
14-JAN 14:58:56.21 Th0: schema-ident srv=ESQ clt=ESQ
%ESQINT-I-TERMINATED, 14-JAN-1997 14:58:56
```

User Exit Logging

The User Exit Logging gives information about the execution of user exits. Following information is logged:

- call of any user exit
- return of any user exit
- important parameters and return values

Example:

```
$ esqlg:=userexit
$ show symbol esqlg
  ESQLG = "USEREXIT"
$ esqint
%ESQINT-I-STARTED, 14-JAN-1997 15:00:46, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

%ESQRTS-I-ESQINFO, ESQLG is enabled for 'USEREXIT'
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select yacht_id from yacht;

%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:OP)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:LF)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:LF)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:L2)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:S1)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:2158, cc:OP)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:L3)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:S1)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:S1)
```

```
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:S2)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:L1)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:RC)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:2158, cc:LF)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:L3)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:S1)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:L3)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:2158, cc:L2)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
     YACHT_ID

       6230
       6228
       6200
       5001
        157
        156
        155
        154
        153
        152
        151
        150
        149
        148
        147
        146
        145
        144
esqint: q

%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:2158, cc:BT)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:2158, cc:CL)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)
%ESQRTS-I-ESQINFO, Th0: > uex5(sid:2158, con:-1, cc:CL)
%ESQRTS-I-ESQINFO, Th0: < uex5(rc=0)

%ESQINT-I-TERMINATED, 14-JAN-1997 15:01:24
```

Elapsed Time Logging

Elapsed Time Logging gives information about the total elapsed time of each SQL request as well as split up into the elapsed time of each software component involved in milliseconds and as a percentage.

The elapsed time values displayed are:

tot	total elapsed time for one SQL request
clt	elapsed time spent by the ADABAS SQL Server client software
net	elapsed time spent by the communication to the server
srv	elapsed time spent by ADABAS SQL Server server
vo/net	elapsed time spent by the communication to ADABAS
ada	elapsed time spent by the ADABAS nucleus

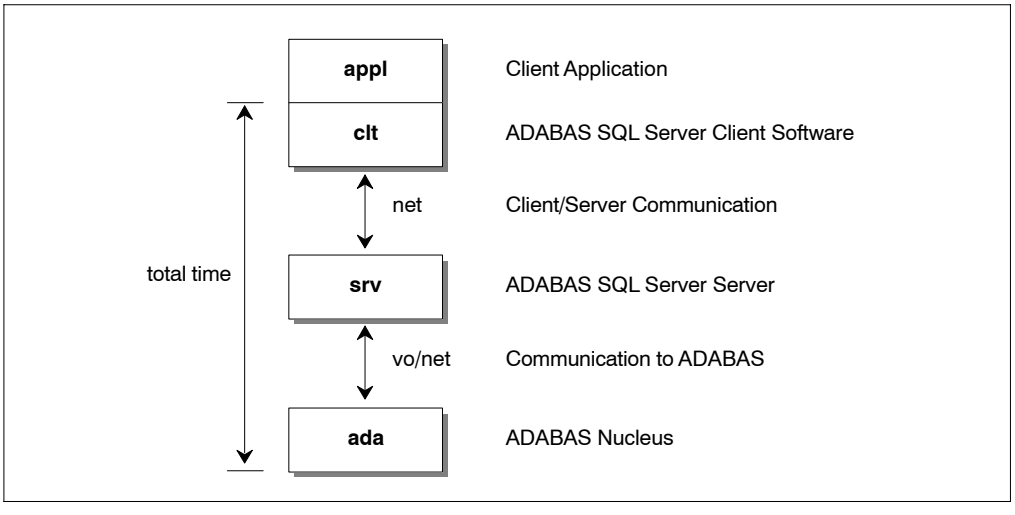


Figure 4-1: Data Flow and Time Portions

Log lines of the following style are displayed with each SQL request:

```
%ESQRTS-I-Clt: ti-quot     tot     clt     net     srv  vo/net     ada
%ESQRTS-I-Clt:      ms    7599       9      42    4013     640    2895
%ESQRTS-I-Clt:      %     100       0       1      53       8      38
```

Note:
If ESQLG is set to "ALLTIME" or "TIME=ALL" then additionally the total times spent on each software layer will be displayed.

Example:

```
$ esqlg:=time
$ show symbol esqlg
  ESQLG = "TIME"
$ esqint
%ESQINT-I-STARTED, 14-JAN-1997 15:05:18, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

%ESQRTS-I-ESQINFO, ESQLG is enabled for 'TIME'
%ESQRTS-I-ESQINFO, Clt: ti-quot     tot     clt     net     srv  vo/net     ada
%ESQRTS-I-ESQINFO, Clt:      ms     163      21       0     110      32       0
%ESQRTS-I-ESQINFO, Clt:      %      100      13       0      67      20       0
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select * from information_schema.tables;
%ESQRTS-I-ESQINFO, Clt: ti-quot     tot     clt     net     srv  vo/net     ada
%ESQRTS-I-ESQINFO, Clt:      ms     153       1       0     152       0       0
%ESQRTS-I-ESQINFO, Clt:      %      100       1       0      99       0       0
%ESQRTS-I-ESQINFO, Clt: ti-quot     tot     clt     net     srv  vo/net     ada
%ESQRTS-I-ESQINFO, Clt:      ms       3       1       0       2       0       0
%ESQRTS-I-ESQINFO, Clt:      %      100      33       0      67       0       0
%ESQRTS-I-ESQINFO, Clt: ti-quot     tot     clt     net     srv  vo/net     ada
%ESQRTS-I-ESQINFO, Clt:      ms       3       0       0       3       0       0
%ESQRTS-I-ESQINFO, Clt:      %      100       0       0     100       0       0
%ESQRTS-I-ESQINFO, Clt: ti-quot     tot     clt     net     srv  vo/net     ada
%ESQRTS-I-ESQINFO, Clt:      ms       2       0       0       2       0       0
%ESQRTS-I-ESQINFO, Clt:      %      100       0       0     100       0       0
%ESQRTS-I-ESQINFO, Clt: ti-quot     tot     clt     net     srv  vo/net     ada
%ESQRTS-I-ESQINFO, Clt:      ms      19       2       0       2       4      11
%ESQRTS-I-ESQINFO, Clt:      %      100      11       0      11      21      58
...
 TABLE_SCHEMA                    TABLE_NAME                      TABLE_TYPE

 INFORMATION_SCHEMA              SERVER_INFO                     VIEW
```

```
 INFORMATION_SCHEMA               INFORMATION_SCHEMA_CATALOG_NAME  VIEW
 INFORMATION_SCHEMA               SCHEMATA                         VIEW
 INFORMATION_SCHEMA               CLUSTERS                         VIEW
 INFORMATION_SCHEMA               TABLES                           VIEW
 ...
esqint: quit
%ESQRTS-I-ESQINFO, Clt: ti-quot     tot     clt     net     srv  vo/net     ada
%ESQRTS-I-ESQINFO, Clt:      ms       0       0       0       0       0       0
%ESQRTS-I-ESQINFO, Clt:      %     100       0       0       0       0       0
%ESQRTS-I-ESQINFO, Clt: ti-quot     tot     clt     net     srv  vo/net     ada
%ESQRTS-I-ESQINFO, Clt:      ms      37       1       0       5       8      23
%ESQRTS-I-ESQINFO, Clt:      %     100       3       0      14      22      62
%ESQRTS-I-ESQINFO, Clt: elapsed time for ESQ session    1066 ms
%ESQINT-I-TERMINATED, 14-JAN-1997 15:05:48
%ESQRTS-I-ESQINFO, Clt: ti-quot     tot     clt     net     srv  vo/net     ada
%ESQRTS-I-ESQINFO, Clt:      ms       0       0       0       0       0       0
%ESQRTS-I-ESQINFO, Clt:      %     100       0       0       0       0       0
%ESQRTS-I-ESQINFO, Clt: elapsed time for ESQ session       0 ms
```

Sort Logging

When using ORDER BY or GROUP BY clauses, ADABAS SQL Server is in some cases forced to perform an explicit internal sorting of data. The sorting itself is done by the VO layer of the ADABAS SQL Server. This implies that on different platforms different sorting algorithms are used. This sorting may become a bottleneck during processing. To allow monitoring and analysis of the sort step, a special sort logging is offered by ADABAS SQL Server.

If the symbol ESQLG is set to “SORT”, the sort step will log statistical information about the resource usage. These statistics can be analyzed. Based on the analysis, default sort buffer sizes may be newly assigned, thus improving the performance.

Example:

```
$ esqlg:=sort
$ show symbol esqlg
  ESQLG = ”SORT”
$ esqint
%ESQINT-I-STARTED, 14-JAN-1997 15:11:23, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

%ESQRTS-I-ESQINFO, ESQLG is enabled for ’SORT’
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select sailor_name,sailor_id from sagtours.sailor order by 1,2;
%SAGVO-I-SORTBYTES, sum  16452, core  16380, file     72, core_size 16KB
%SAGVO-I-SORTRECS,  sum    457, core    455, file      2, ratio 99:1
%SAGVO-I-SORTTIME,  sum     10, load      0, sort     10, unload      0

 SAILOR_NAME                      SAILOR_ID

 ADAM DERFLER                        216503
 ADAM KING                           215602
 AGNES GRGOIRE                       209602
 ALAIN                               207303
...
esqint: quit
%ESQINT-I-TERMINATED, 14-JAN-1997 15:11:44
```

Discussion:

```
SORTBYTES:
Total volume sort data   : 16452 Bytes
In buffer                : 16380 Bytes
In temporary file        :    72 Bytes
Current core area        :    16 KB     (DEFAULT)
SORTRECS:
Total number of records  :   457
In buffer                :   455
In file                  :     2
Ratio buffer/temp. file  :  99:1
SORTTIME:
Load time                :    0 Milliseconds
Sort time                :   10 Milliseconds
Unload time              :    0 Millisecond
Total sort time          :   10 Milliseconds
```

Note:
Loading/Unloading of records is performed in 16 KB chunks.

If the default sort-buffer size is set to 48 (unit = KB), the sort logging will change as follows.

```
$ esqsrtbuf:=48
$ show symbol esqsrtbuf
  ESQSRTBUF = ”48”
$ esqint
%ESQINT-I-STARTED, 14-JAN-1997 15:17:34, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

%ESQRTS-I-ESQINFO, ESQLG is enabled for ’SORT’
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select sailor_name,sailor_id from sagtours.sailor order by 1,2;
%SAGVO-I-SORTBYTES, sum  16452, core  16452, file      0, core_size 48KB
%SAGVO-I-SORTRECS,  sum    457, core    457, file      0, ratio 100:0
%SAGVO-I-SORTTIME,  sum     10, load      0, sort     10, unload      0

 SAILOR_NAME                    SAILOR_ID

 ADAM DERFLER                      216503
 ADAM KING                         215602
 AGNES GRGOIRE                     209602
...
esqint: quit
%ESQINT-I-TERMINATED, 14-JAN-1997 15:18:09
```

ADABAS Utility Logging

ADABAS SQL Server ADABAS Utility logging gives information about calls from the ADABAS SQL Server run-time system to a ADABAS utility. ADABAS utilities are used by ADABAS SQL Server to execute SQL DDL statements like CREATE TABLE.

Example:

```
$ esqlg:=adu
$ show symbol esqlg
  ESQLG = "ADU"
$ esqint
%ESQINT-I-STARTED, 14-JAN-1997 15:21:42, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: dba
Password:

%ESQRTS-I-ESQINFO, ESQLG is enabled for 'ADU'
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: create schema john;
%ESQINT-I-SUCCESS, SQL statement completed successfully
esqint: create table john (col1 int);
14-JAN 15:21:59.24 Th0: ADU > adafdu
%ESQRTS-I-ESQINFO, Th0: ADU - DBID=222
%ESQRTS-I-ESQINFO, Th0: ADU - FILE=243
%ESQRTS-I-ESQINFO, Th0: ADU - NAME=DBA.JOHN
%ESQRTS-I-ESQINFO, Th0: ADU - DSSIZE=10B
%ESQRTS-I-ESQINFO, Th0: ADU - NISIZE=10B
%ESQRTS-I-ESQINFO, Th0: ADU - UISIZE=10B
%ESQRTS-I-ESQINFO, Th0: ADU - MAXISN=300
%ESQRTS-I-ESQINFO, Th0: ADU - REUSE=(DS)
%ESQRTS-I-ESQINFO, Th0: ADU - FIELDS
%ESQRTS-I-ESQINFO, Th0: ADU - 01,AA,4,F,NC
;COL
1
%ESQRTS-I-ESQINFO, Th0: ADU - END_OF_FIELDS
14-JAN 15:22:02.09 Th0: ADU < rsp=0/0/0
%ESQINT-I-SUCCESS, SQL statement completed successfully
esqint: quit
%ESQINT-I-TERMINATED, 14-JAN-1997 15:22:06
```

Security Logging

The logging mechanism of an ADABAS SQL Server also enables logging of GRANT and REVOKE statements and user system access via CONNECT statements. Logging for this can be enabled via the symbol ESQLG.

Examples:

Example 1:

```
$ esqlg:=sec
```

The GRANT/REVOKE statement will be logged. If any errors occur, the corresponding response codes are also displayed.

```
13-APR 13:41:19.93 Th0: VDK: GRANT SELECT, INSERT ON VDK.TABLE1 TO BRU
==> Finished (Rsp: 0)
```

The grant statement has successfully finished.

```
13-APR 13:41:59.79 Th0: BRU: GRANT UPDATE ON VDK.TABLE1 TO TRO
==> Finished (Rsp: -6702)
```

The grant statement has finished with an error, in this case with a security violation, i.e. the grantor is not allowed to grant this privilege.

```
13-APR 13:43:42.37 Th0: VDK: GRANT ALL ON VDK.TABLE1 TO FS WITH GRANT OPTION
==> Finished (Rsp: 0)
```

The grant statement has successfully finished.

The CONNECT statement results in a password verification which will also be logged if ESQLG is set to SEC.

```
13-APR 13:49:54.88 Th0: User VDK has connected successfully.
13-APR 13:51:28.96 Th0: Authentication error during CONNECT for user BRU.
```

Example 2:

```
$ esqlg:=sec=all
```

In this case, additionally, every single GRANT/REVOKE step resulting from a GRANT ALL/ REVOKE ALL is logged.

```
13-APR 13:56:23.41 Th0: VDK: REVOKE ALL ON VDK.TABLE1 FROM AE ==> Started
```

A REVOKE ALL statement has started.

```
13-APR 13:56:23.52 Th0: VDK: REVOKE UPDATE(COL1) ON VDK.TABLE1 FROM AE
==> (generated from ALL)
13-APR 13:56:24.04 Th0: VDK: REVOKE UPDATE(COL2) ON VDK.TABLE1 FROM AE
==> (generated from ALL)
13-APR 13:56:24.11 Th0: VDK: REVOKE UPDATE(COL4) ON VDK.TABLE1 FROM AE
==> (generated from ALL)
13-APR 13:56:24.22 Th0: VDK: REVOKE SELECT ON VDK.TABLE1 FROM AE
==> (generated from ALL)
13-APR 13:56:24.30 Th0: VDK: REVOKE INSERT ON VDK.TABLE1 FROM AE
==> (generated from ALL)
```

The individual REVOKE steps result from the above REVOKE ALL statement.

```
13-APR 13:56:24.38 Th0: VDK: REVOKE ALL ON VDK.TABLE1 FROM AE ==> Finished
(Rsp: 0)
```

A REVOKE ALL statement has sucessfully finished.

Example 3:

```
$ esqlg:=-sec
```

No GRANT/REVOKE statements will be logged. The same is true if nothing is specified.

ESQLNK Call Logging

ESQLNK Call Logging gives information about:

- the point in time where the application is calling ESQLNK, esqerr or esqint and
- the point in time when control is returned to the application

Example:

```
$ esqlg:=entry
$ show symbol esqlg
  ESQLG = "ENTRY"
$ esqint
%ESQINT-I-STARTED, 14-JAN-1997 15:26:56, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

%ESQRTS-I-ESQINFO, ESQLG is enabled for 'ENTRY'
%ESQRTS-I-ESQINFO, Clt: > ESQLNK()        cnt=   1
%ESQRTS-I-ESQINFO, Clt: < ESQLNK()        cnt=   1      rsp=0
%ESQRTS-I-ESQINFO, Clt: > esqinf()        cnt=   2
%ESQRTS-I-ESQINFO, Clt: < esqinf()        cnt=   2      rsp=0
%ESQRTS-I-ESQINFO, Clt: > esqinf()        cnt=   3
%ESQRTS-I-ESQINFO, Clt: < esqinf()        cnt=   3      rsp=0
%ESQRTS-I-ESQINFO, Clt: > esqinf()        cnt=   4
%ESQRTS-I-ESQINFO, Clt: < esqinf()        cnt=   4      rsp=0
%ESQRTS-I-ESQINFO, Clt: > esqinf()        cnt=   5
%ESQRTS-I-ESQINFO, Clt: < esqinf()        cnt=   5      rsp=0
%ESQRTS-I-ESQINFO, Clt: > esqinf()        cnt=   6
%ESQRTS-I-ESQINFO, Clt: < esqinf()        cnt=   6      rsp=0
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select * from information_schema.tables;
%ESQRTS-I-ESQINFO, Clt: > ESQLNK()        cnt=   7
%ESQRTS-I-ESQINFO, Clt: < ESQLNK()        cnt=   7      rsp=0
%ESQRTS-I-ESQINFO, Clt: > ESQLNK()        cnt=   8
%ESQRTS-I-ESQINFO, Clt: < ESQLNK()        cnt=   8      rsp=0
%ESQRTS-I-ESQINFO, Clt: > ESQLNK()        cnt=   9
%ESQRTS-I-ESQINFO, Clt: < ESQLNK()        cnt=   9      rsp=0
%ESQRTS-I-ESQINFO, Clt: > ESQLNK()        cnt=  10
%ESQRTS-I-ESQINFO, Clt: < ESQLNK()        cnt=  10      rsp=0
%ESQRTS-I-ESQINFO, Clt: > ESQLNK()        cnt=  11
%ESQRTS-I-ESQINFO, Clt: < ESQLNK()        cnt=  11      rsp=0
..
 TABLE_SCHEMA                    TABLE_NAME                     TABLE_TYPE
INFORMATION_SCHEMA              SERVER_INFO                    VIEW
```

```
 INFORMATION_SCHEMA              INFORMATION_SCHEMA_CATALOG_NAME  VIEW
 INFORMATION_SCHEMA              SCHEMATA                         VIEW
 INFORMATION_SCHEMA              CLUSTERS                         VIEW
 INFORMATION_SCHEMA              TABLES                           VIEW
...
esqint: quit
%ESQRTS-I-ESQINFO, Clt: > ESQLNK()        cnt=  32
%ESQRTS-I-ESQINFO, Clt: < ESQLNK()        cnt=  32      rsp=0
%ESQRTS-I-ESQINFO, Clt: > ESQLNK()        cnt=  33
%ESQRTS-I-ESQINFO, Clt: < ESQLNK()        cnt=  33      rsp=0
%ESQINT-I-TERMINATED, 14-JAN-1997 15:27:15
%ESQRTS-I-ESQINFO, Clt: > ESQLNK()        cnt=  34
%ESQRTS-I-ESQINFO, Clt: < ESQLNK()        cnt=  34      rsp=0
%ESQRTS-I-ESQINFO, Clt: > esqtrm()        cnt=  35
%ESQRTS-I-ESQINFO, Clt: < esqtrm()        cnt=  35      rsp=0
```

Explain Logging

This logging facility provides information about the execution of an SQL statement within the ADABAS SQL Server. It provides a pseudo-code representation of the execution path of the statement. Within the pseudo-code the following information is available:

- ADABAS access commands
- Descriptors used for searching the tables
- Restriction evaluation after fetch has occurred
- Group by and order by information
- Predicates and subqueries

Detailed knowledge of ADABAS and SQL is needed in order to be able to fully interpret the resultant logging information.

To turn the logging facility on, the ESQLG symbol must be set to EXPL as follows:.

$ esqlg:=expl

To produce an output, a DML statement must be executed. If the meta-program already exists, then no output will be produced, but the statement needs to be one of the following types:

SELECT	Explains the statements execution fully
DELETE	Explains the searching phase of the delete statement
UPDATE	Explains the searching phase of the update statement
INSERT	Explains the searching phase on nested tables

Note:
If a view is used within the statement, then the statement is 'view merged'. View merging is where all references to the view are replaced by appropriate references to the base table/tables upon which the view is based. Hence the explain logging output will not contain any view references, rather table references.

Example:

- The created view:

```
create view view_1 as
     select start_harbor, destination_harbor
          from cruise
          where cruise_price < 1000;
```

- The executed statement:

```
select destination_harbor
     from view_1
          where start_harbor = 'MIAMI';
```

- The explained statement:

```
select destination_harbor
     from cruise
          where start_harbor = 'MIAMI'
          and cruise_price < 1000;
```

  The result of the explained statement :

```
          ********   Start of EXPLAIN logging   ********
          for (L2 ;SAGTOURS.CRUISE-0; L2 )  /* GET NEXT */
          {
            (((SAGTOURS.CRUISE.START_HARBOR = 'MIAMI' )
              (SAGTOURS.CRUISE.CRUISE_PRICE < 1000 )))
          }
          ********    End  of EXPLAIN logging    ********
```

ADABAS Access Commands

In an SQL query statement, there is generally some access to the underlying database. The SQL is translated to ADABAS database access commands. The ADABAS commands that are used in the explained statement are displayed as the actual ADABAS command names, for example, S1, L3, L2. For details on the ADABAS commands, refer to the *ADABAS Command Reference Manual*.

The Explain logging for a fetch is similar to a 'FOR' construct in 'C'. It has two parts for the ADABAS commands: one for get first record, and another for get next record. The S1 command cannot be used in the get next section; one of the other commands, normally an L1(N) is used. S1s and L3s are displayed before the 'for' loop header, along with their search buffers, rather than as part of the loop header. Also, there is a section which shows the table name, the schema name and the table level, whether it is LEVEL 0, 1 or 2. With an L1I, additional information of upper and lower searching bounds is displayed before the 'for' loop header as with the S1s and L3s.

Examples:

Example 1: An example fetch header for a simple table access.

```
select cruise_id from cruise;
********   Start of EXPLAIN logging   ********
S1 (AA > 00AA EOF )
for (   ;SAGTOURS.CRUISE-0; L1N)  /* GET NEXT */
{
}
********    End of EXPLAIN logging    ********

or
select * from contract;
********   Start of EXPLAIN logging   ********

for (L2;SAGTOURS.CONTRACT-0;L2 ) /* GET NEXT */
{
}
********    End of EXPLAIN logging    ********
```

If a query on a clustered table is logged, then a different output is used so that the access on the tables with lower table levels than the requested table can be documented.

Example 2: The table BUILDINGS is a level 2 table.

```
select building_name, height
from buildings;
********   Start of EXPLAIN logging   ********

for (L2 ;SAGTOURS.BUILDINGS-0; L2 )  /* GET NEXT */
{
  for (every occurrence SAGTOURS.BUILDINGS-1; )
  {
    if (buffer exhausted)
        L1  refill
           for (every occurrence SAGTOURS.BUILDINGS-2; )
           {
             if (buffer exhausted)
               L1  refill
            }
   }
 }
 ********    End of EXPLAIN logging    ********
```

Here an access is made for every subtable below the requested table if the buffer for that subtable access is exhausted. The L1 refill should only occur if the buffering factor is incorrectly set. This factor can be altered via a DDL statement.

Descriptor Searching

Descriptor searching occurs when ADABAS S1 or L3 commands are issued. It shows what descriptor/super descriptors and operands are used for the search criteria. The search buffer is displayed as the ADABAS SQL Server internal format. Prefixing the search buffer is the ADABAS command that is associated with the buffer. So a typical example of a search buffer is:

```
S1 (AA = 0078 EOF )
```

- AA is the ADABAS shortname of the descriptor used.
- The '=' is the comparison operator used for filtering unwanted records.
- The 4-digit hexadecimal number is an offset into an internal ADABAS SQL Server storage table which holds the value that is to be compared against.
- EOF is the end-of-buffer marker, not present with L3 search buffers.

With an L3 it is possible to receive a search buffer like the following:

```
L3 (AA <= {temp value} )
```

The {temp value} is a value that has not been calculated yet, i.e. it is not a constant. This may be a value that has be retrieved from a previous table access, and therefore is not know at compile time.

Also an L1I produces an output similar to S1s and L3s. However, they are not based on a descriptor but on the ADABAS ISN column (SEQNO). Below is an example L1I buffer:

```
L1I (ISN >= 2 AND ISN <= 4 )
```

There may be a case where an S1 may be produced without a loop. This is where a set of ISNs needs to be established for processing multiple times or where the resultant set is accessed by another S1. At most, this command will be executed only once for that specific table. These will then be displayed as:

```
select cruise_id from cruise where exists
(select * from yacht where yacht_id > 0);
********   Start of EXPLAIN logging   ********
if (first execution for SAGTOURS.YACHT-0;)
{
  S1 (AA > 006E EOF )
}
  for (R1 ;SAGTOURS.YACHT-0; L1N)  /* GET NEXT */
  {
  }
    for (L2 ;SAGTOURS.CRUISE-0; L2 )  /* GET NEXT */
    {
    }
********    End  of EXPLAIN logging    ********
```

The other case where an S1 may be produced without a loop is where there will only be one result returned from the S1 (a unique descriptor search). Then the display will look like:

```
select * from yacht where yacht_id = 152;
********   Start of EXPLAIN logging   ********

S1 (AA = 0082 EOF ) on SAGTOURS.YACHT-0;
********    End of EXPLAIN logging    ********
```

Restriction Evaluation

Restriction evaluation is the process of removing unwanted records from the set returned by the ADABAS database access. The format of its display is similar to the same predicate in the SQL statement that is being explained. However, the only logical connector that is displayed is the ‘OR’ operator. No ‘AND’ operators will be displayed. Therefore, between each factor, if an ‘OR’ is not displayed, then it should be assumed that the operator is an ‘AND’.

If a ‘NOT’ predicate is used, this will not be shown. The statement that is displayed will be the negated version of the original statement according to the rules of DeMorgan’s theorem. The data type BINARY is displayed as its hexadecimal value.

Example:

SQL Statement	Explain Output
where NOT (column_1 > 65) ;	(((column_1 <= 65)))
where bin_col > Y’10101100’;	(((bin_col > H’AC’)))

The predicates are split up into expressions, terms and factors. Each section has its opening and closing brackets, to mark the start and end of each. Each expression can have many terms and each term can have many factors.

Example:

```
select   yacht_id, yacht_name
from     yacht
where    yacht_id = :host_variable_1
or       yacht_name = :host_variable_2
and yacht_id = 23 * yacht_length;
********   Start of EXPLAIN logging   ********
for (L2 ;SAGTOURS.YACHT-0; L2 )  /* GET NEXT */
{
 (((SAGTOURS.YACHT.YACHT_ID = ? ))
 OR  ((SAGTOURS.YACHT.YACHT_NAME = ? )
      (SAGTOURS.YACHT.YACHT_ID = 23 * SAGTOURS.YACHT.YACHT_LENGTH )))
}
********    End of EXPLAIN logging    ********
```

Note:
The ’?’ is used to indicate a host variable or a parameter marker.

Grouping and Ordering

These actions are caused by the GROUP and ORDER clauses of SQL. An order clause may be obsolete if the equivalent occurs in the group clause or if it can be implemented by an ordered retrieval (L3). In the output, there is an indication of what sorting is done, and then what constraints are done on each of the groups. The constraints are defined in a HAVING clause. Each SQL statement with grouping will, at its most inner point of the query, have a ‘save tuple’ command.

Example:

```
select yacht_type from yacht
group by yacht_type
having min(yacht_name) ='AQUIVA';
********   Start of EXPLAIN logging   ********
for (L2 ;SAGTOURS.YACHT-0; L2 )  /* GET NEXT */
{
      save tuple
}
Sort by:
  SAGTOURS.YACHT.YACHT_TYPE
for (every group)
{
   HAVING:
   ((( MIN()  = 'AQUIVA' )))
}
********    End of EXPLAIN logging    ********
```

Predicates and Subqueries

The predicates BETWEEN and IN are not displayed in their original versions. The BETWEEN is transformed to a range consisting of an upper and a lower bound value. IN predicates are transformed to comparison predicates, which are connected by an OR.

Example:

SQL Statement	Explain Output
where column_1 between 3 and 78;	(((table_name.column_1 >= 3) (table_name.column_1 <= 78)))
where column_1 in (12, 13, 14)	(((table_name.column_1 = 12)) ((table_name.column_1 = 13)) ((table_name.column_1 = 14)))

Other predicates are not transformed.

There are two sorts of subqueries: there are those that can be solved before the main execution, and then there are those that are solved during the main execution. If a subquery contains a reference to a table declared in the outer query then no distinction is made about which table it actual is if the tables have the same name.

In predicates where a subquery is used, what is displayed depends on the type of subquery. Those that have been solved before the main query execution are replaced by ‘pre_evaluated subquery’. Those that are to be executed within the main execution because of the need of values obtained from outside the subquery or because there is more than one record returned are displayed at the point where they would be executed.

Example:

```
select id_customer
from contract
where id_cruise = ALL(select cruise_id from cruise);
********   Start of EXPLAIN logging   ********
for (L2 ;SAGTOURS.CONTRACT-0; L2 )  /* GET NEXT */
{
  (((SAGTOURS.CONTRACT.ID_CRUISE = )))
      S1 (AA <> 0096 EOF )
      for (   ;SAGTOURS.CRUISE-0; L1N)  /* GET NEXT */
      {
      }
}

********    End of EXPLAIN logging    ********
```

If the ‘NOT’ predicate is used with an EXISTS, where the subquery has an outer reference, then any table joins will show the inner table columns first.

Example:

```
select * from person a
where NOT EXISTS ( select * from yacht b
     where a.person_id >= b.id_owner);
```

is explained as:

```
      for (L2 ;SAGTOURS.PERSON-0; L2 )  /* GET NEXT */
      {
    for (L2 ;SAGTOURS.YACHT-0; L2 )  /* GET NEXT */
       {
      (((SAGTOURS.YACHT.ID_OWNER <= SAGTOURS.PERSON.PERSON_ID)))
    }
      }
```

UNIONs have no special representation within Explain. Each part of the UNION is treated as a separate statement, and the representation of the UNIONs execution path is placed one part after another.

Example:

This is an example of a subquery which can be evaluated before the main execution:

```
select cruise_id, destination_harbor
from cruise
where cruise_price < (select min(cruise_price)
               from cruise
               where id_yacht < 145);
********   Start of EXPLAIN logging   ********
for (L2 ;SAGTOURS.CRUISE-0; L2 )  /* GET NEXT */
{
 (((SAGTOURS.CRUISE.ID_YACHT < 145 )))
}
for (L2 ;SAGTOURS.CRUISE-0; L2 )  /* GET NEXT */
{
  (((SAGTOURS.CRUISE.CRUISE_PRICE < {pre-evaluated subquery})))

}
********    End of EXPLAIN logging    ********
```

Example:

This is an example of a level 1 subtable:

```
select city_name, population
from cities;
********   Start of EXPLAIN logging   ********
for (L2 ;SAGTOURS.CITIES-0; L2 )  /* GET NEXT */
{
  for (every occurrence SAGTOURS.CITIES-1; )
  {
    if (buffer exhausted)
      L1  refill
  }
}
********    End of EXPLAIN logging    ********
```

Additional Logging Facilities

ADABAS Command Logging

In terms of ADABAS, the ADABAS SQL Server is an application program which processes SQL requests. To process SQL requests the ADABAS SQL Server makes use of the standard ADABAS call interface. A detailed discussion of the interaction with ADABAS is given in the *ADABAS SQL Server Programmer's Guide*, **Appendix D**.

To allow the monitoring and analysis of an application, the ADABAS SQL Server offers the possibility to log the usage of ADABAS. In addition to this, the ADABAS command logging may be used for trouble shooting.

Due to its internal structure, the ADABAS SQL Server has two software layers which logically interface to ADABAS:

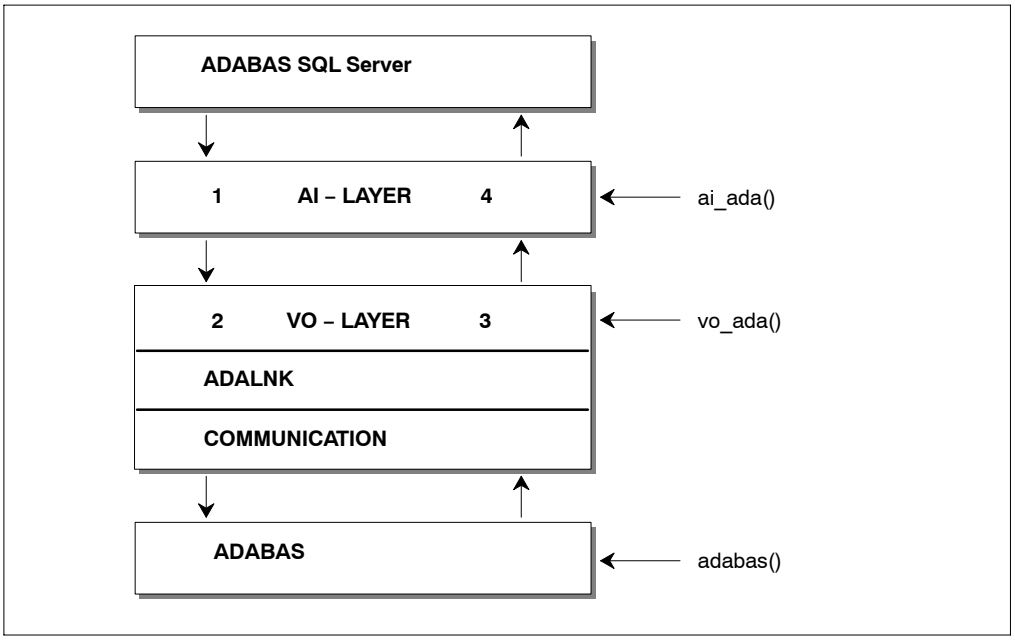
- The AI (ADABAS Interface) layer

  This layer maps CLIENT contexts to the matching ADABAS session contexts ("User queue elements") and performs various optimizations (Multifetch, RC optimization, and so on.)

- The VO (Virtual Operating-system) layer

  This layer performs the real "call ADABAS()" for different contexts.

Based on this, the ADABAS SQL Server offers four logging points:



Log Point(1) Upon entering the AI layer (AI IN)
Log Point(2) Upon entering the VO layer (VO IN)
Log Point(3) Upon leaving the VO layer (VO OUT)
Log Point(4) Upon leaving the AI layer (AI OUT)

All the above log points offer the standard ADABAS call interface, consisting of:

Control Block (CB)
Format Buffer (FB)
Record Buffer (RB)
Search Buffer (SB)
Value Buffer (VB)
ISN Buffer (IB)

Each buffer can be displayed in part or full.

Note:
Remember that, in case of Multifetch, the buffers may be really huge (up to 32KB). So great care should be taken if full buffer logging is chosen.

Additionally, there is also an ADABAS command log (CLOG) style logging available. This logging is done at the log point ”VO OUT” (right after returning from the real “call ADABAS()”) and provides a compact yet informative logging.

Enabling ADABAS SQL Server ADABAS Command Logging

The ADABAS SQL Server ADABAS command logging is enabled and controlled via the ADABAS SQL Server parameter file. Depending on whether logging is to be enabled for the complete server or just a particular client, it is marked in the server’s or in the client’s parameter file in the GLOBAL section.

The active logging is defined as a union of the server’s parameter settings and the client’s parameter settings.

Example: Enabling ADABAS CLOG-style logging on server side

```
/*#############################################################
*
*  File name  : esqsrv.par
*
*  Description: ADABAS SQL Server Server parameter file
*
 #############################################################*/
    GLOBAL
       BEGIN
       ...
       ADABAS LOG (CLOG)
       ...
    END
```

Syntax examples for ADABAS logging specification:

```
ADABAS LOG (CLOG)
```

Does a compact logging at the “VO OUT” log point.

```
ADABAS LOG (CLOG, VO OUT(CB(FULL)))
```

Does a compact logging at the “VO OUT” log point. Additionally, the ADABAS CB is logged at the “VO IN” log point.

```
ADABAS LOG (AI OUT (CB(10,10),SB(FULL),RB(FULL)))
```

At the log point “AI OUT”, the first 10 and the last 10 Bytes of the CB, and the whole SB and RB are logged.

```
ADABAS LOG (AI IN( FULL ))
```

At the log point ”AI IN”, all buffers (CB,FB,RB,SB,IB,VB) are logged in the full length.

```
ADABAS LOG (FULL)
```

At all four log point, all buffers are logged in full length.

Warning:
This may result in a dramatic logging output.

I/O Channel for ADABAS Logging Data

The logging information is written to standard output SYS$OUTPUT.

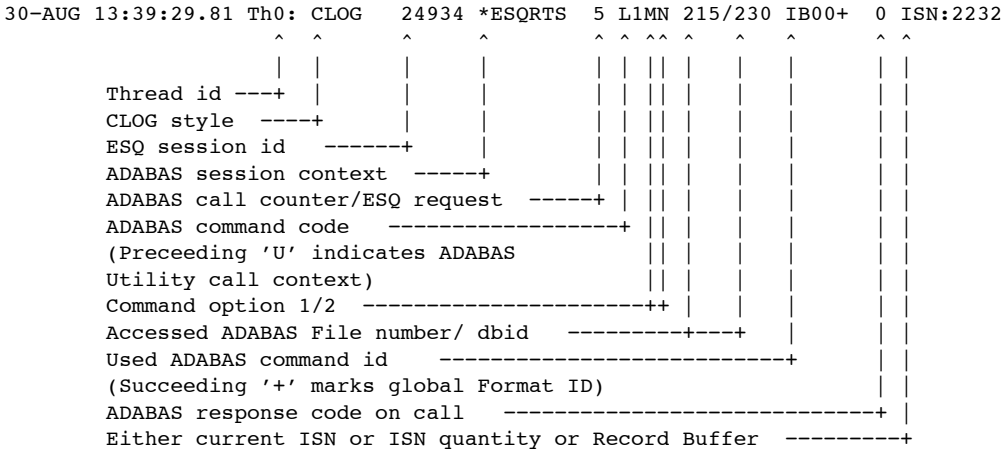
Examples of an ADABAS Command Log

1. ADABAS LOG (CLOG)

```
$ esqint
%ESQRUN-I-PAR, Using parameter file ESQRUN.PAR
%ESQINT-I-STARTED, 14-JAN-1997 15:55:23, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

14-JAN 15:55:24.75 Th0: CLOG   3625 *ESQRTS  1 S1 I 222/ 30 IM00+  0 ISQ:1
14-JAN 15:55:24.77 Th0: CLOG   3625 *ESQ     2 OP.. 222/  0 ....   0 RB=”.”
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select yacht_id from sagtours.yacht;
14-JAN 15:55:42.73 Th0: CLOG   3625 *ESQRTS  1 S1 I 222/ 30 IH00+  0 ISQ:1
14-JAN 15:55:42.74 Th0: CLOG   3625 *ESQ     1 L2M  222/ 38 S001   0 ISN:1
14-JAN 15:55:42.76 Th0: CLOG   3625 *ESQ     1 L2M  222/ 38 S001   0 ISN:17

    YACHT_ID

         144
...
esqint: select count(*) from sagtours.sailor;
14-JAN 15:56:11.11 Th0: CLOG   3625 *ESQRTS  1 S1 I 222/ 30 IH00+  0 ISQ:1
14-JAN 15:56:11.13 Th0: CLOG   3625 *ESQ     2UOP.. 222/  0 ....  22 RB=”UTI.”
14-JAN 15:56:11.20 Th0: CLOG   3625 *ESQ     3 L1MI 222/ 37 S002   0 ISN:1

 NO_COLUMN_NAME

            457
esqint: quit
14-JAN 15:56:17.93 Th0: CLOG   3625 *ESQ     1 BT   222/  0 ....   0 ISN:0
14-JAN 15:56:17.95 Th0: CLOG   3625 *ESQ     2 CL   222/  0 ....   0 ISN:0
14-JAN 15:56:17.96 Th0: CLOG   3625 *ESQRTS  3 CL   222/  0 ....   0 ISN:0
%ESQINT-I-TERMINATED, 14-JAN-1997 15:56:17
```

Discussion of CLOG-style Logging

Each executed ADABAS call is logged with one clog line. The clog line can be read as follows:



2. ADABAS LOG (CLOG, VO IN(FB(FULL)))

```
$ esqint
%ESQRUN-I-PAR, Using parameter file ESQRUN.PAR
%ESQINT-I-STARTED, 14-JAN-1997 16:00:07, Version 1.4B/2.1 (ALPHA AXP/OpenVMS)

ESQ User: esq
Password:

%ESQRTS-I-ESQINFO, Th0: ADALOG 3625 *ESQRTS > VO_ADA FB FULL (LENGTH: 300)
004EFA88 00000000:   53412C33 322C412C 53422C33 322C422E    SA,32,A,SB,32,B.
004EFA98 00000010:   00000000 00000000 00000000 00000000    ................
                16 lines identical to line above
004EFBA8 00000120:   00000000 00000000 00000000             ............
14-JAN 16:00:09.30 Th0: CLOG   3625 *ESQRTS  1 S1 I 222/ 30 IM00+  0 ISQ:1
14-JAN 16:00:09.31 Th0: CLOG   3625 *ESQ     2 OP.. 222/  0 ....   0 RB=”.”
%ESQINT-I-CONNECT, Server ESQ140AT connected successfully
esqint: select yacht_id from sagtours.yacht;
%ESQRTS-I-ESQINFO, Th0: ADALOG 3625 *ESQRTS > VO_ADA FB FULL (LENGTH: 300)
004EF5C8 00000000:   4F412C33 322C412C 4F422C33 322C412C    OA,32,A,OB,32,A,
004EF5D8 00000010:   4F432C33 322C412C 4F442C33 322C412C    OC,32,A,OD,32,A,
004EF5E8 00000020:   4F452C34 302C412C 4F462C31 2C412E00    OE,40,A,OF,1,A..
004EF5F8 00000030:   00000000 00000000 00000000 00000000    ................
                14 lines identical to line above
004EF6E8 00000120:   00000000 00000000 00000000             ............
14-JAN 16:00:28.06 Th0: CLOG   3625 *ESQRTS  1 S1 I 222/ 30 IH00+  0 ISQ:1
%ESQRTS-I-ESQINFO, Th0: ADALOG 3625 *ESQ    > VO_ADA FB FULL (LENGTH: 7)
005F64D0 00000000:   41412C34 2C462E                        AA,4,F.
14-JAN 16:00:28.07 Th0: CLOG   3625 *ESQ     1 L2M  222/ 38 S001   0 ISN:1
%ESQRTS-I-ESQINFO, Th0: ADALOG 3625 *ESQ    > VO_ADA FB FULL (LENGTH: 7)
005F64D0 00000000:   41412C34 2C462E                        AA,4,F.
14-JAN 16:00:28.09 Th0: CLOG   3625 *ESQ     1 L2M  222/ 38 S001   0 ISN:17

    YACHT_ID

         144
...
          6230
esqint: quit
14-JAN 16:00:47.80 Th0: CLOG   3625 *ESQ     1 BT   222/  0 ....   0 ISN:0
14-JAN 16:00:47.81 Th0: CLOG   3625 *ESQ     2 CL   222/  0 ....   0 ISN:0
14-JAN 16:00:47.82 Th0: CLOG   3625 *ESQRTS  3 CL   222/  0 ....   0 ISN:0
%ESQINT-I-TERMINATED, 14-JAN-1997 16:00:47
```

A

APPENDIX A — THE PARAMETER PROCESSING LANGUAGE (PPL)

The Parameter Processing Language (PPL) has been developed to enable suitable settings of individual parts of the ADABAS SQL Server system. The list on the following two pages shows in which mode each parameter can be set and what the effects are.

This version of the PPL is presented in 5 logical sections. These sections are: PRECOMPILER, COMPILER, RUNTIME, GLOBAL and SERVER.

Syntax

PPL works via a text parameter file, which can be of any size (including lines/record lengths) and can contain comments.

The comments can be in the form of 'C' comments, starting with '/*' and ending with '*/'.

Comments can also start with '<<' and end with '>>' or start with '--' and end with '--'.

The file must not contain line numbers. Statements in the file can span multiple lines and there can be multiple statements on a line. If supported by the given operating system, some parameters can be specified directly in the command line.

Each logical unit of parameter settings is individually described in a block which starts with either the keyword **BEGIN** or **(** (parenthesis) with one or more settings for that block then being defined. The block is then terminated with either the keyword **END** or **)** (parenthesis). A block for the same logical section can occur several times within the parameter file.

What Happens When These Parameters Are Set?

The following table shows in which mode each parameter can be set and what the effects are.

For example, PRECOMPILER COBOL LANGUAGE SETTINGS (COBOL II = ON), if set for a client, affects that particular client while in Client/Server Mode, and, if set on the server side in the same mode, sets the default for all clients. If set while in Precompiler mode, it affects both single-user and multi-user applications. This particular setting has no effect in Runtime Mode.

Individual settings of the same parameter (for example the GLOBAL ERROR Setting with DBID, FNR and LANGUAGE) may in some cases affect the client, in other cases the server. Therefore, these parameters have been broken down to reflect each possibility.

One of the following statuses will apply:

D = Default for the Client
Y = Used in this mode
– = No effect / is ignored or issues a warning

Parameter Settings	Client-Server Mode		Precompiler Mode		Runtime-Linked-In Mode	
	Client	Server	Single-user	Multi-user	Single-user	Multi-user
PRECOMPILER						
C LANGUAGE	Y	D	Y	Y	–	–
COBOL LANGUAGE	Y	D	Y	Y	–	–
COMPILATION UNIT IDENTIFIER	Y	D	Y	Y	Y	Y
HOST LANGUAGE	Y	D	Y	Y	–	–
COMPILER						
MAXIMUM	Y	D	Y	Y	Y	Y
EXPECTED UPDATE	Y	D	Y	Y	Y	Y
MODE	Y	D	Y	Y	Y	Y
RUNTIME						
MAXIMUM	Y	D	–	–	Y	Y
ROLLBACK ON ERROR	Y	D	–	–	Y	Y
LOCK WHEN READING	Y	D	–	–	Y	Y
SERVER	Y	D	–	–	–	Y

Parameter Settings	Client-Server Mode		Precompiler Mode		Runtime-Linked-In Mode	
	Client	Server	Single-user	Multi-user	Single-user	Multi-user
GLOBAL						
ADABAS Settings						
– ADABAS MULTIFETCH	–	Y	Y	–	Y	Y
– ADABAS HOLD ON/OFF	Y	D	Y	Y	Y	Y
– ADABAS FREE FILE SEARCH R.	Y	D	–	–	Y	Y
– ADABAS EXU/EXCLUSIVE UPDATE	Y	D	–	–	Y	Y
MULTIFETCH	Y	D	–	–	Y	Y
BUFFER MANAGER	–	Y	Y	–	Y	–
CATALOG	–	Y	Y	–	Y	–
ERROR Settings						
– ERROR DBID	–	Y	Y	–	Y	–
– ERROR FNR	–	Y	Y	–	Y	–
– ERROR LANGUAGE	Y	Y	Y	Y	Y	Y
PREDICT						
– PREDICT DBID	–	Y	Y	–	Y	–
– PREDICT FNR	–	Y	Y	–	Y	–
– PREDICT CROSS REFERENCE	Y	D	Y	Y	Y	Y
– PREDICT CROSS REFERENCE LIB.	Y	D	Y	Y	Y	Y
DEFAULT SCHEMA IDENTIFIER	Y	–	Y	Y	Y	Y
FILE	Y	D	Y	Y	Y	Y

Parameter Settings	Client-Server Mode		Precompiler Mode		Runtime-Linked-In Mode	
	Client	Server	Single-user	Multi-user	Single-user	Multi-user
SERVER						
NAME	–	Y	–	–	–	–
THREADS	–	Y	–	–	–	–
TYPE	–	Y	–	–	–	–
MAXIMUM	–	Y	–	–	–	–

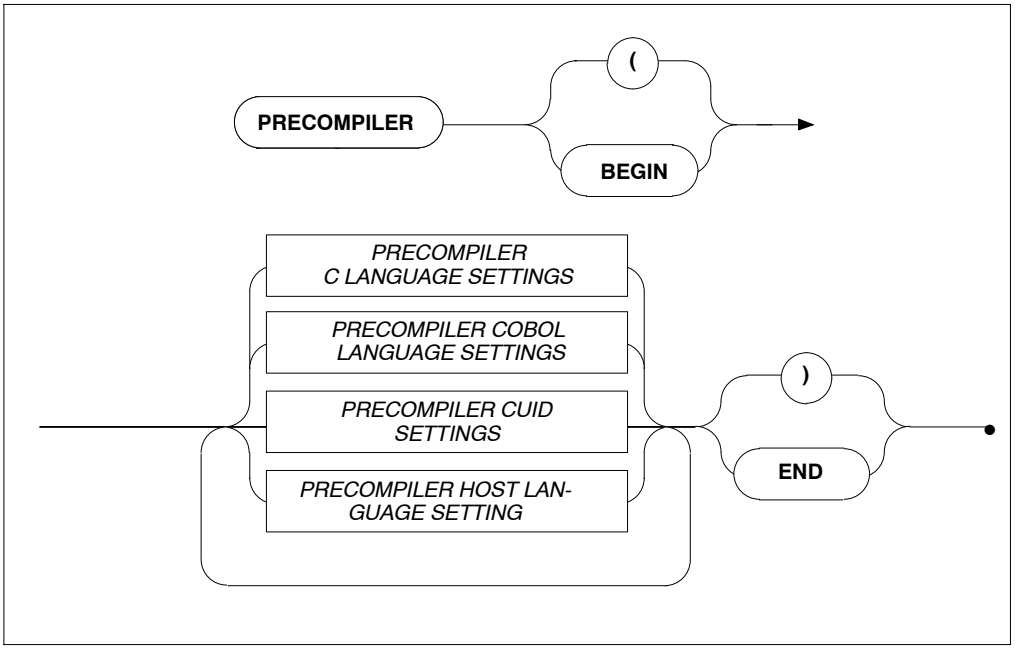
PRECOMPILER Settings

The precompiler-specific settings influence the behavior of the ADABAS SQL Server precompiler.

The types of settings for the precompiler are:

- C LANGUAGE SETTINGS
- COBOL LANGUAGE SETTINGS
- COMPILATION UNIT IDENTIFIER SETTINGS
- HOST LANGUAGE SETTINGS

SYNTAX

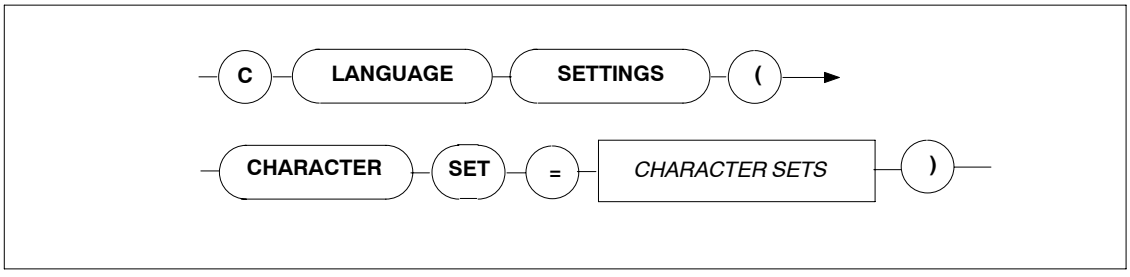


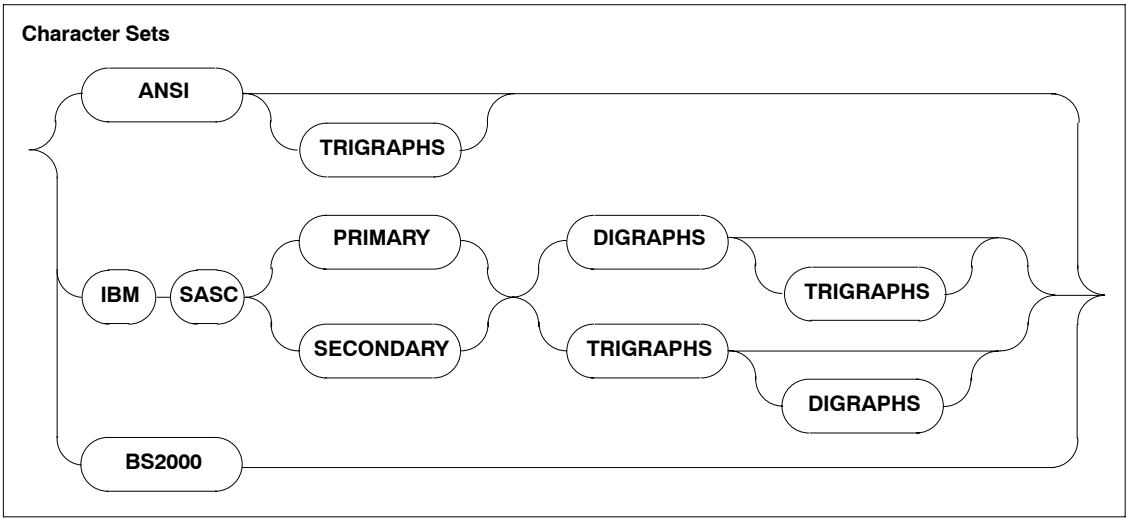
PRECOMPILER C LANGUAGE Settings

Function

To specify the C language environment.

Syntax





Description

These settings allow you to specify the character set used The character sets available are:

ANSI	Normal ASCII character set. There is an added option of TRIGRAPHS which some versions of C have.
IBM SASC PRIMARY	IBM's C compilers primary character set. There are the added options of TRIGRAPHS and DIGRAPHS.
IBM SASC SECONDARY	IBM's C compilers secondary character set. Also has TRIGRAPHS and DIGRAPHS.
BS2000	Siemens BS2000 series C compiler character set.

Limitations

None.

Examples

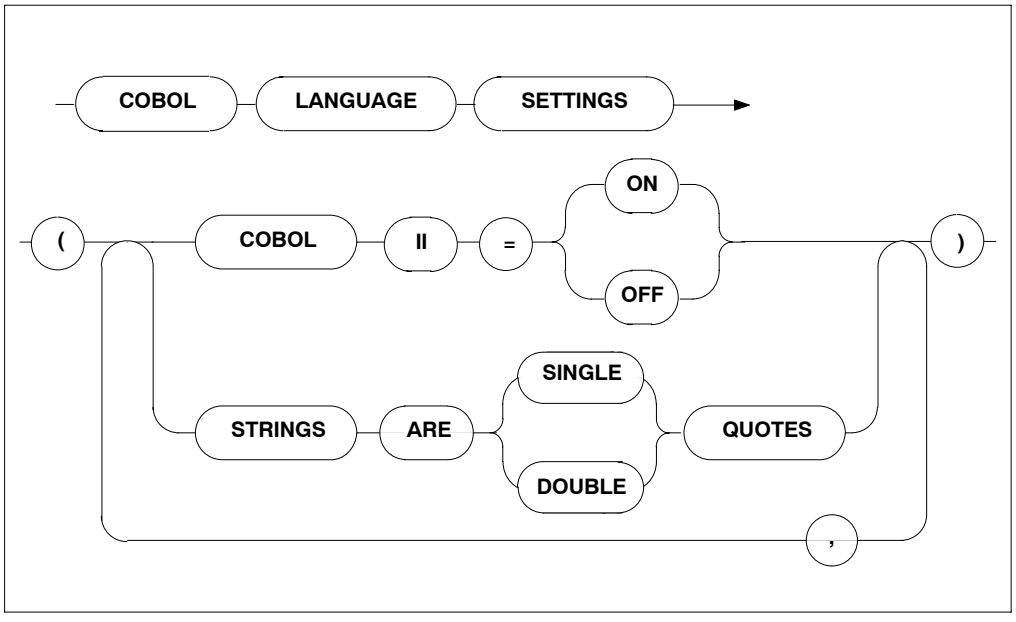
```
PRECOMPILER
BEGIN
     C LANGUAGE SETTINGS ( CHARACTER SET = ANSI TRIGRAPHS)
END

PRECOMPILER
BEGIN
     C LANGUAGE SETTINGS ( CHARACTER SET = IBM SASC PRIMARY )
END
```

PRECOMPILER COBOL LANGUAGE Settings

Function

To set whether the COBOL compiler used is COBOL II compatible or not.

Syntax



Description

If the COBOL compiler used is not COBOL II compatible, it is mandatory to set the COBOL language setting to OFF. In this case, the code generated by the ADABAS SQL SERVER COBOL Precompiler will not include the END-IF or END-PERFORM, etc. clauses and is thus compatible with COBOL as well as COBOL II. If the COBOL II compiler is used, this setting may be set to ON or may be omitted because this is also the default setting.

Furthermore, setting the STRINGS option will determine whether a COBOL string is single or double quoted. The default setting is: DOUBLE

Limitations

None

Examples

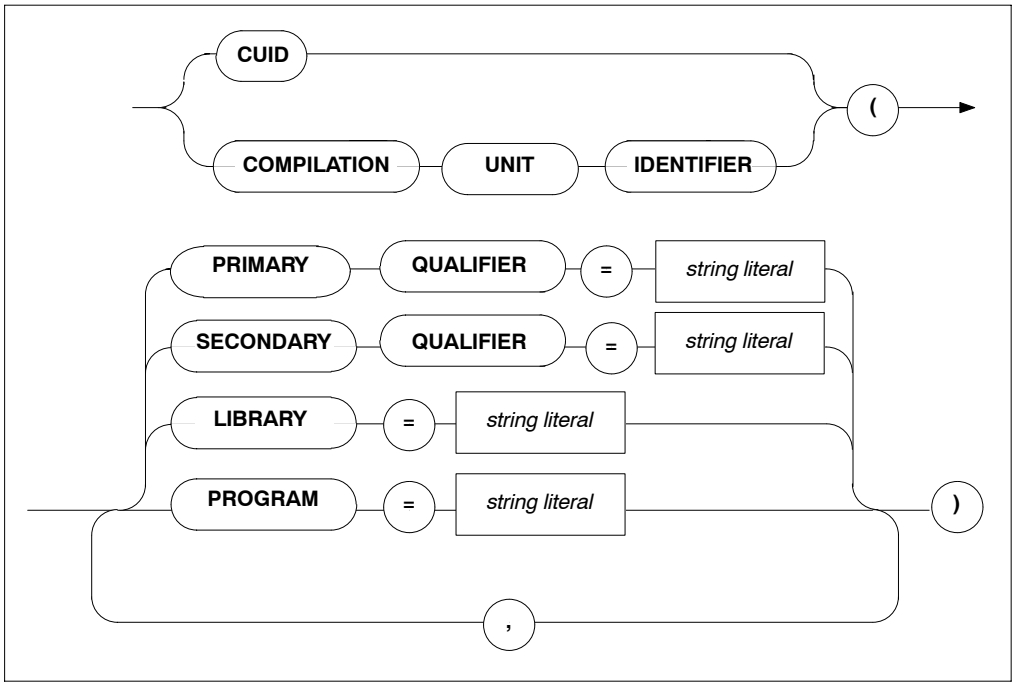
```
PRECOMPILER
BEGIN
     COBOL LANGUAGE SETTINGS ( COBOL II = ON, STRINGS ARE SINGLE QUOTES )
END
```

PRECOMPILER COMPILATION UNIT IDENTIFIER Settings

Function

To set the compilation unit ID. Specification of program is mandatory.

Syntax



Description

The compilation unit identifier is stored in a key field and is used to identify a meta program. This field consists of four user-definable elements and one host language element predefined by the precompiler:

PRIMARY QUALIFIER	any user-defined string of up to 5 characters.
SECONDARY QUALIFIER	any user-defined string of up to 5 characters.
LIBRARY	any user-defined string of up to 8 characters.
PROGRAM	any user-defined string of up to 8 characters.

Note:
The specification of the PROGRAM setting is mandatory.

The default values for the above elements are: blank

Strict naming conventions should be established to make sure that the entire key field amounts to a unique value. A new meta program with the same compilation unit identifier will overwrite an existing one in the catalog.

Limitations

None.

Examples

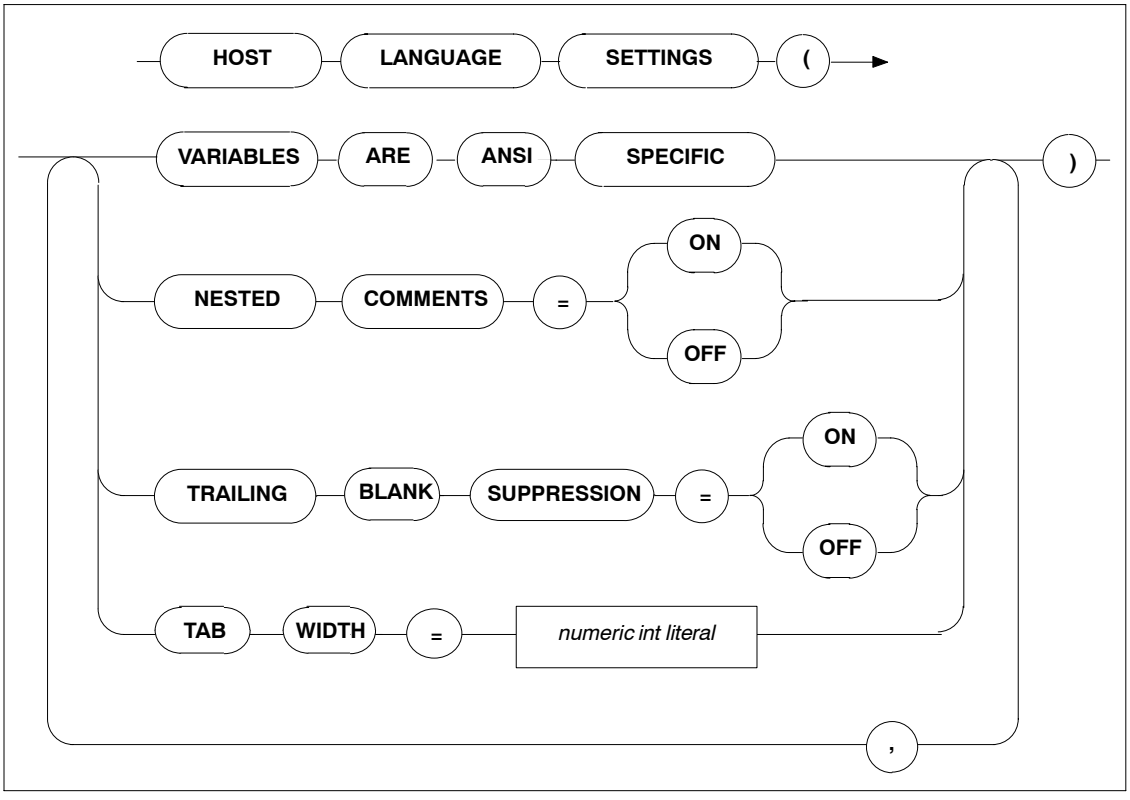
```
PRECOMPILER
BEGIN
     CUID ( LIBRARY = 'ESQ', PROGRAM = 'CR_BASE' )
END

PRECOMPILER
BEGIN
     CUID ( PROGRAM = "CR_ACC", LIBRARY = 'BANKS' )
END
```

PRECOMPILER HOST LANGUAGE Setting

Function

To specify the host language environment.

Syntax



Description

VARIABLES ARE ANSI SPECIFIC	This setting defines that the Precompiler only searches for host variables within an SQL statement when used according to the ANSI specification, i.e. :<host_var>. This is the default condition and can presently not be altered.
NEST COMMENTS	Offsetting the default, this setting defines that the host language compiler allows nested comments. The default setting is OFF.
TRAILING BLANK the SUPPRESSION	This setting defines whether trailing blank strings returned by ADABAS SQL Server are suppressed or not. The default = OFF is also the only valid setting in ANSI mode.
TAB WIDTH	This setting defines the width of a TAB character. This is particularly important for COBOL compilers when a TAB character is used to get to a specific column, i.e. column 8. The default value is 8 characters.

Limitations

None.

Examples

```
PRECOMPILER
BEGIN
    HOST LANGUAGE SETTINGS (VARIABLES ARE ANSI SPECIFIC, TAB WIDTH = 2)
END
```

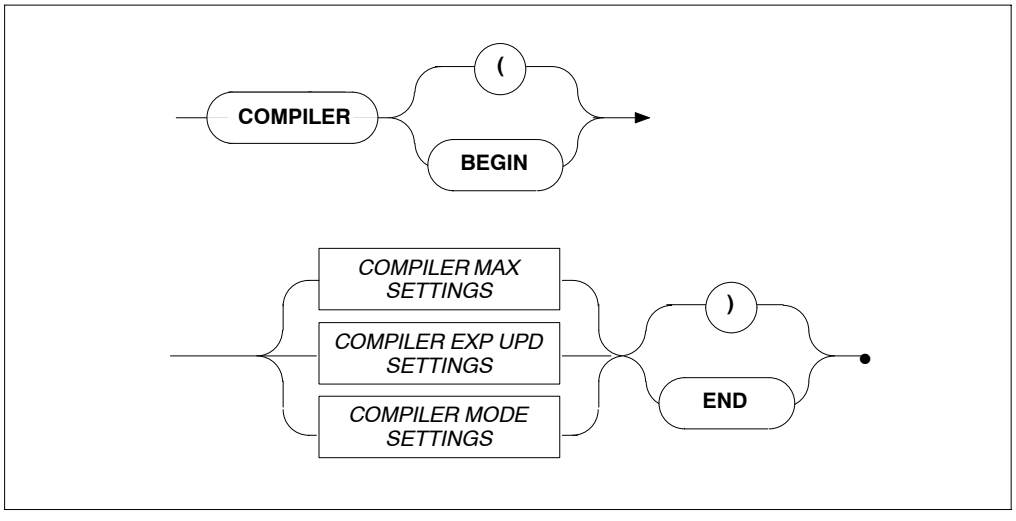
COMPILER Settings

The compiler-specific settings influence the behavior of the ADABAS SQL Server compiler.

The types of COMPILER settings are:

- MAXIMUM Settings
- EXPECTED UPDATE Setting
- MODE Settings

Syntax

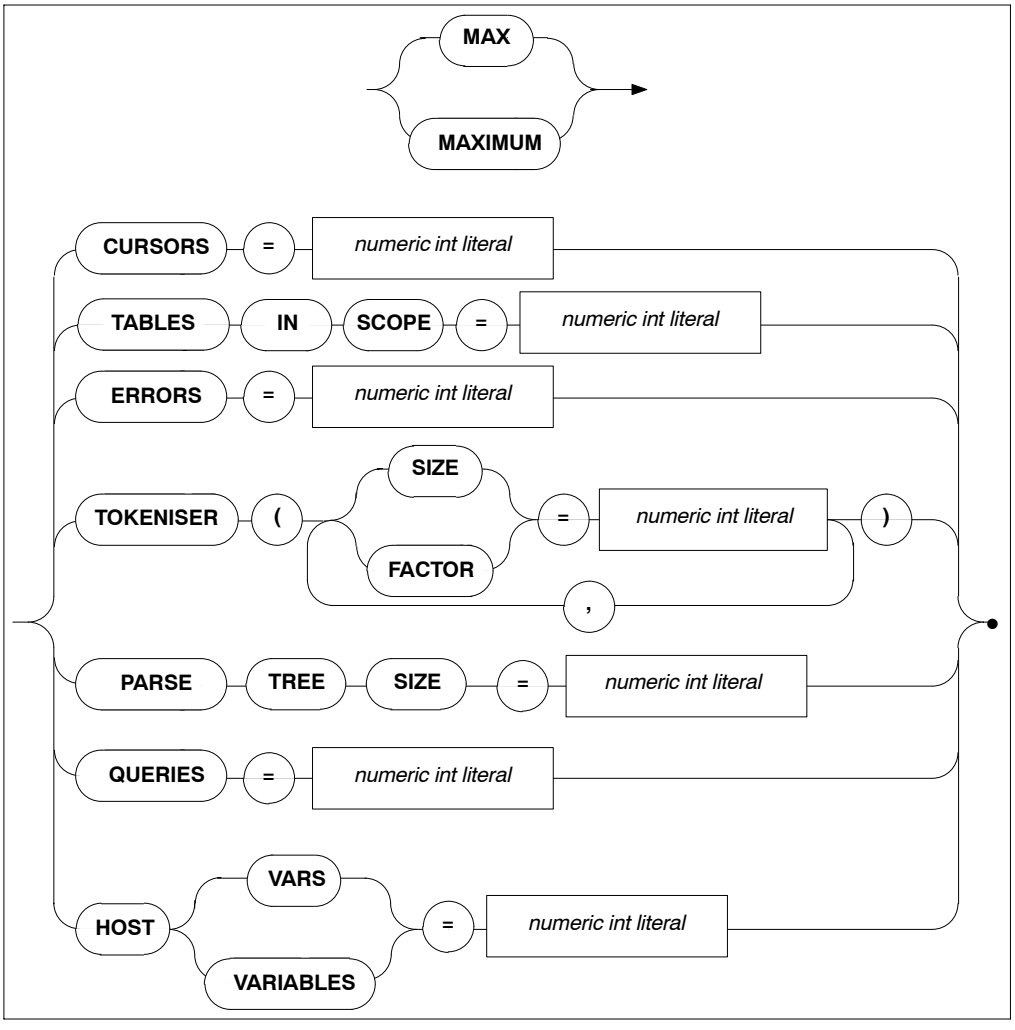


COMPILER MAXIMUM Settings

Function

To set the compile-time system's maximum values.

Syntax



Description

Allows the setting of maximum compile-time system values. The minimum value to be specified is usually 1, with the exception of maximum parse tree size whose minimum setting is 10000 bytes. The maximum value depends on the underlying hardware. Default values are as follows:

CURSORS	The number of different cursors which can be declared in one compilation unit. Default: 32
TABLES IN SCOPE	The number of tables that may be in scope within an SQL statement. Default: 32
ERRORS	The number of errors that can be logged in a compilation unit. Default: 100
PARSE TREE SIZE	The number of bytes in memory reserved for each parse tree generated by the SQL Compiler. Default (as well as minimum value): 10,000.
TOKENISER SIZE	The size of a tokeniser buffer in bytes. Default: 4000
TOKENISER FACTOR	The number of identifiers, strings constants etc... per 100 tokens. Default: 10
QUERIES	The number of subqueries allowed within an SQL statement. Default: 32
HOST VARIABLES	The number of host variables allowed in a compilation unit. Default: 250

Limitations

The higher these values are set, the more memory will be required by the compile-time system.

If the PARSE TREE SIZE is set too small, the compiler attempts to acquire more buffer space and start parsing again, which will result in reduced performance. If the compiler fails in acquiring a larger buffer, then it will abort with a fatal error.

Examples

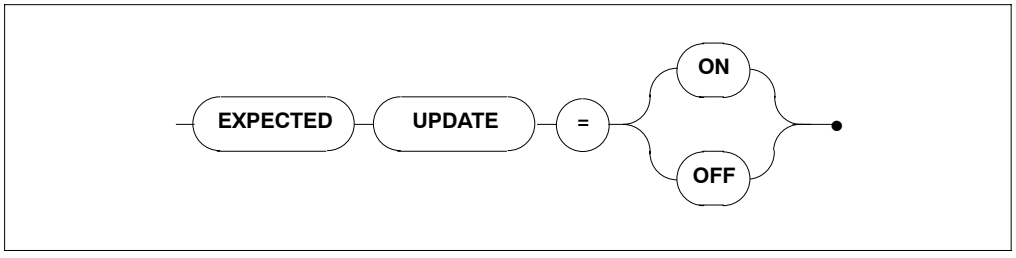
```
COMPILER
BEGIN
     MAX TOKENISER ( SIZE = 32000 )
     MAX CURSORS = 32
     MAX TABLES IN SCOPE = 15
     MAX ERRORS = 50
     MAX QUERIES = 5
     MAX HOST VARS = 32
     MAX PARSE TREE SIZE = 20000
END
```

COMPILER EXPECTED UPDATE (Cursor) Setting

Function

To set whether the default is an updatable cursor or a read-only cursor.

Syntax



Description

If a cursor is defined in one compilation unit without the FOR UPDATE clause and without any UPDATE or DELETE statements, the default results in a read-only cursor. If this cursor is referred to in another compilation unit, a runtime error will be raised. This clause alters the default for cursors.

Limitations

None.

Examples

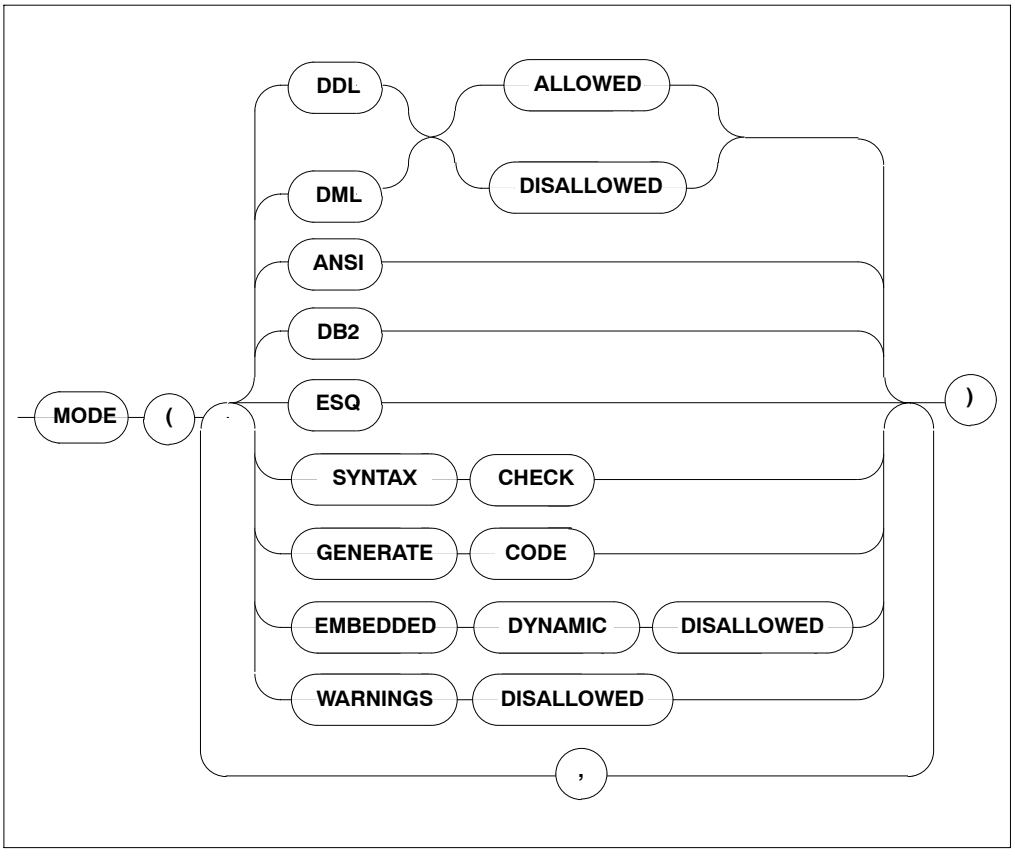
```
COMPILER
BEGIN
     EXPECTED UPDATE = ON
END


COMPILER
BEGIN
     EXPECTED UPDATE = OFF
END
```

COMPILER MODE Settings

Functions

To set the particular mode of operation for the compiler.

Syntax



Description

When a particular mode is set, only statements or options which conform to that mode will be permitted. For example, if DDL is disallowed, it will not be possible to compile DDL statements like CREATE TABLE or DCL statements like GRANT or REVOKE. If the default mode ESQ is set, warnings will still be issued if the ANSI standard is violated. To suppress these warnings, use the option WARNINGS DISALLOWED.

DDL ALLOWED/DISALLOWED	all DDL/DCL statements can be explicitly permitted or forbidden. There is no special keyword for DCL but DDL implicitly includes DCL.
DML ALLOWED/DISALLOWED	DML statements can be explicitly permitted or forbidden.
ANSI	in this mode, only statements which conform to the ANSI standard will be permitted. All deviations from the standard, i.e., ADABAS SQL Server extensions, will result in compilation errors. The same is true for embedded dynamic statements.
DB2	in this mode, only statements which conform to the DB2 syntax will be permitted. All deviations from the standard, i.e., ADABAS SQL Server extensions, will result in compilation errors. Under CICS, implicit COMMIT or ROLLBACK statements are used at runtime at the end of each CICS task: COMMIT in case of a normal task end and ROLLBACK in case of an abnormal task end.
ESQ	this is the ADABAS SQL Server default mode. The use of all extensions is permitted.
SYNTAX CHECK	the output of object code is suppressed and only a list of syntax errors (if any) is displayed.
GENERATE CODE	object code is produced and a list of syntax errors (if any) is displayed.

EMBEDDED DYNAMIC DISALLOWED	this setting is only valid for the default mode and explicitly excludes embedded dynamic statements.
WARNINGS DISALLOWED	warning messages are suppressed, actual error messages will be displayed.

Limitations

DDL and DML must not both be disallowed at the same time. In ANSI mode, DML and DDL must not both be allowed at the same time, while in ADABAS SQL Server and DB2 modes, this is possible.

The specification of only one mode, ANSI, DB2 or ADABAS SQL Server, is permitted per statement/ compilation unit.

Examples

```
COMPILER
BEGIN
    MODE ( DDL ALLOWED, DML ALLOWED, ESQ )
END

COMPILER
BEGIN
    MODE ( ANSI, SYNTAX CHECK )
END

COMPILER
BEGIN
    MODE ( DDL ALLOWED, DML ALLOWED, DB2, EMBEDDED DYNAMIC DISALLOWED )
END
```

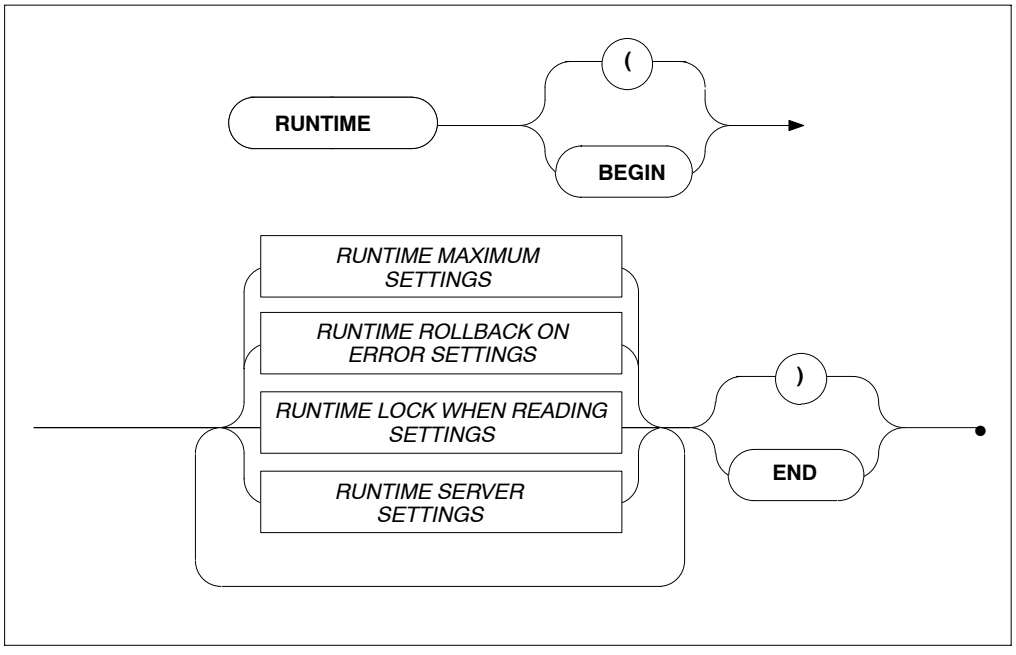
RUNTIME Settings

The Runtime System-specific settings influences the behavior of the Runtime System.

The types of settings for the Runtime System are:

- MAXIMUM RUNTIME SETTINGS
- ROLLBACK ON ERROR SETTING
- LOCK WHEN READING SETTING
- SERVER SETTINGS

Syntax

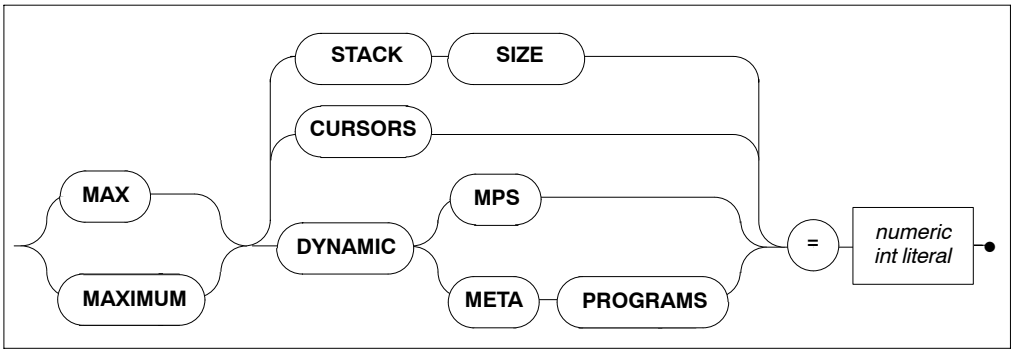


RUNTIME MAXIMUM Settings

Function

To set the maximum Runtime System parameters.

Syntax



Description

These settings are designed so that applications with abnormal requirements do not affect other applications, for example, in memory required.

MAX STACK SIZE	The maximum size of the runtime system stack. Default value is 100, minimum is 1.
MAX CURSORS	The maximum number of cursors expected to be opened. Default value is 3.
MAX DYNAMIC MPS	The maximum number of DYNAMIC meta programs which are expected to be executed. Dynamic meta programs are only in existence while an application is running. Default value is 2.

Limitations

The higher these values are set, the more memory is required. If they are set too low, the Runtime System will terminate with errors.

Examples

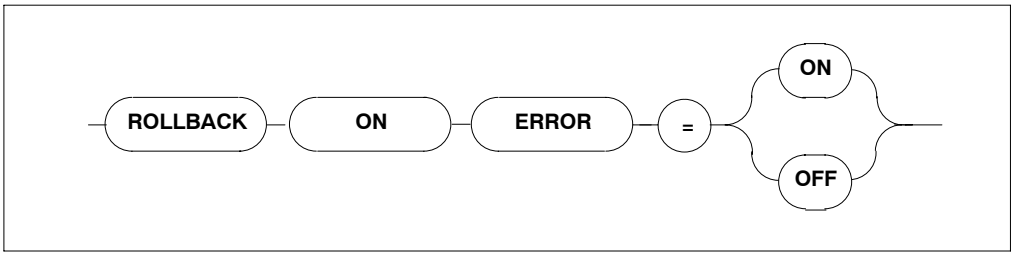
```
RUNTIME
BEGIN
     MAX STACK SIZE = 64 MAX CURSORS = 32
END
```

RUNTIME ROLLBACK ON ERROR Setting

Function

To set that a ROLLBACK occurs when an error is encountered.

Syntax



Description

If this setting is set to ON, the Runtime System issues a ROLLBACK command (backout transaction) whenever an error occurs. The default setting is OFF.

Limitations

None.

Examples

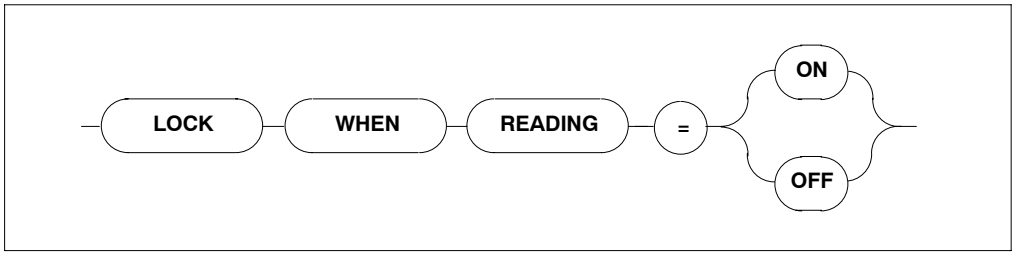
```
RUNTIME
BEGIN
     ROLLBACK ON ERROR = ON
END
```

RUNTIME LOCK WHEN READING Setting

Function

To set whether table rows are locked while they are being retrieved from the database.

Syntax



Description

If LOCK WHEN READING is set to ON, each time a table row is retrieved from the database, that particular row is locked.

If it is set to the default value = OFF, the row will not be locked.

Limitations

None.

Examples

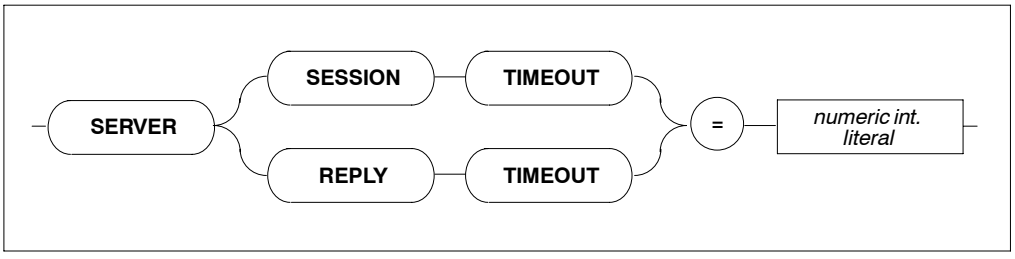
```
RUNTIME
BEGIN
     LOCK WHEN READING = ON
END

RUNTIME
BEGIN
     MAX STACK SIZE = 128 MAX CURSORS = 48 MAX DYNAMIC MPS = 32
END
```

RUNTIME SERVER Settings

Function

To set the length of a timeout for a particular client.

Syntax



Description

This setting defines the server's timeout for a particular client. Each client may set how long it takes for a timeout in minutes.

Zero minutes is infinite (∞).

SESSION TIMEOUT	The length of time a session must be idle before timing out. Default value: 15 minutes.
REPLY TIMEOUT	The length of time a reply may take before timing out. Default value: 15 minutes.

Limitations

For use in client/server mode only.

Examples

```
RUNTIME
BEGIN
     SERVER SESSION TIMEOUT = 5
END
```

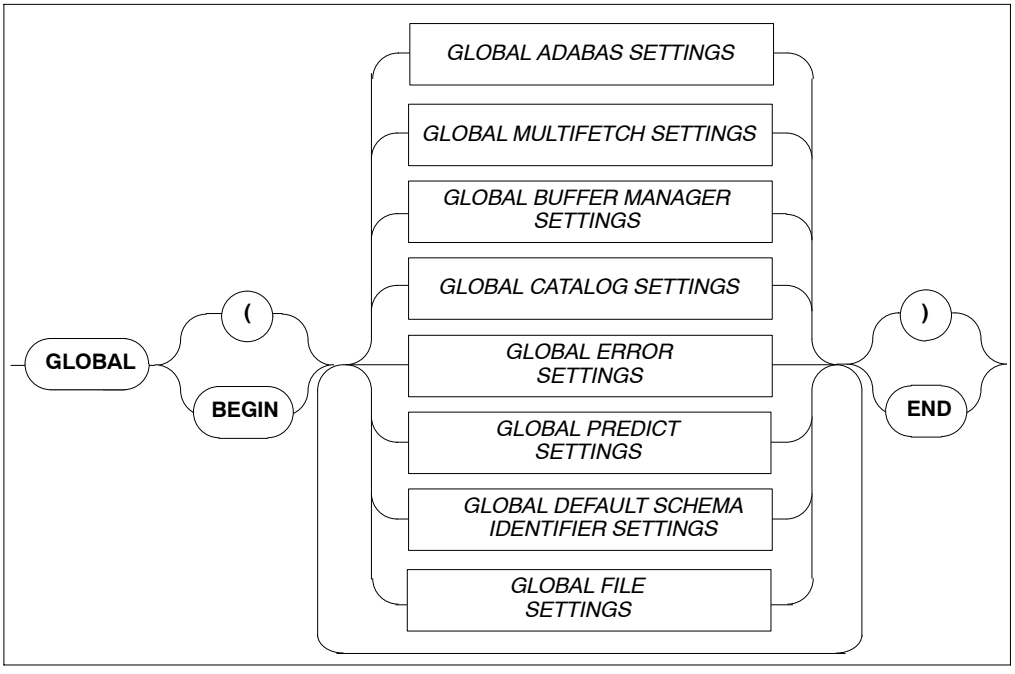
GLOBAL Settings

Within the global section, all parameters which are not limited to the precompiler, compiler, runtime system or trace facility are described. Options set here are effective throughout the entire ADABAS SQL Server system.

The types of settings for the GLOBAL section are:

- ADABAS SETTINGS
- MULTIFETCH SETTINGS
- BUFFER MANAGER SETTINGS
- CATALOG SETTINGS
- ERROR SETTINGS
- PREDICT SETTINGS
- DEFAULT SCHEMA IDENTIFIER SETTINGS
- FILE SETTINGS

Syntax

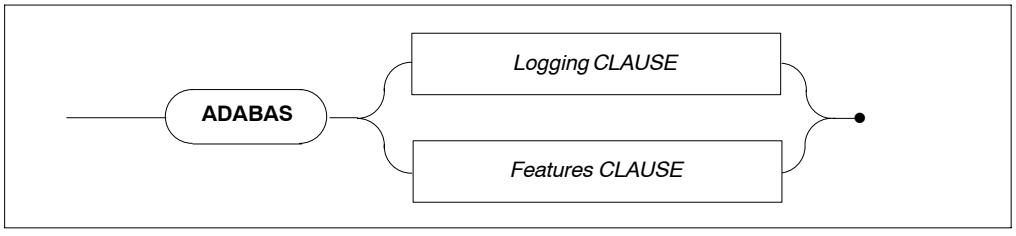

GLOBAL
(
BEGIN
GLOBAL ADABAS SETTINGS
GLOBAL MULTIFETCH SETTINGS
GLOBAL BUFFER MANAGER SETTINGS
GLOBAL CATALOG SETTINGS
GLOBAL ERROR SETTINGS
GLOBAL PREDICT SETTINGS
GLOBAL DEFAULT SCHEMA IDENTIFIER SETTINGS
GLOBAL FILE SETTINGS
)
END

GLOBAL ADABAS Settings

Function

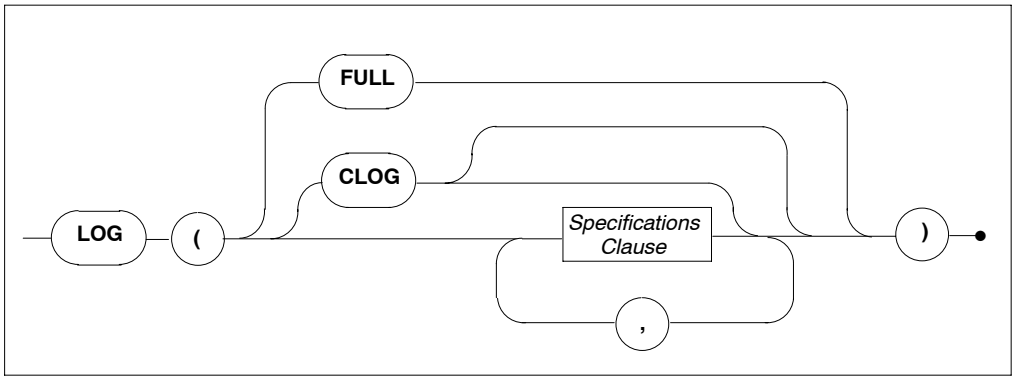
To set ADABAS-specific values that are valid throughout all phases of processing (from precompilation to runtime functions).

Syntax



For information on the individual clauses refer to the description section directly below the relevant syntax diagrams.

Logging Clause

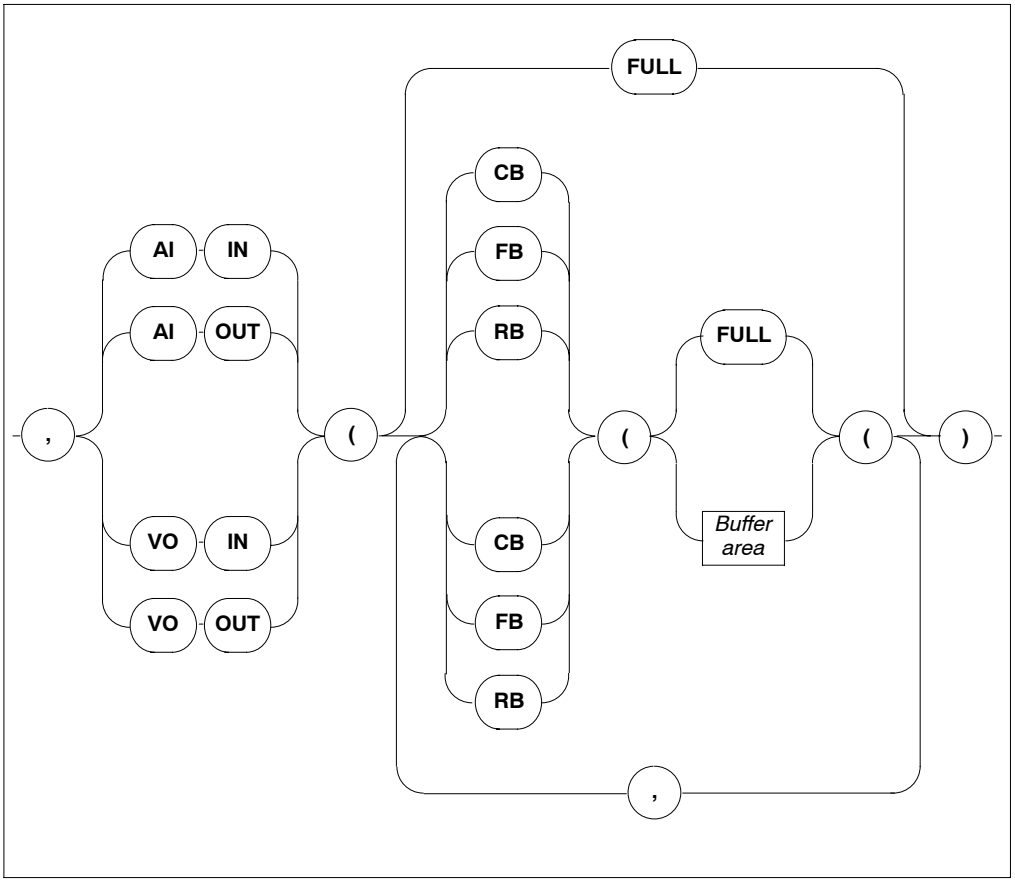


FULL	Full logging of all ADABAS buffers at all log points will be performed, which may be very extensive.
CLOG	If no other details are specified, a compact command logging at the VO–OUT point will be performed. For details regarding the specifications clause see below.

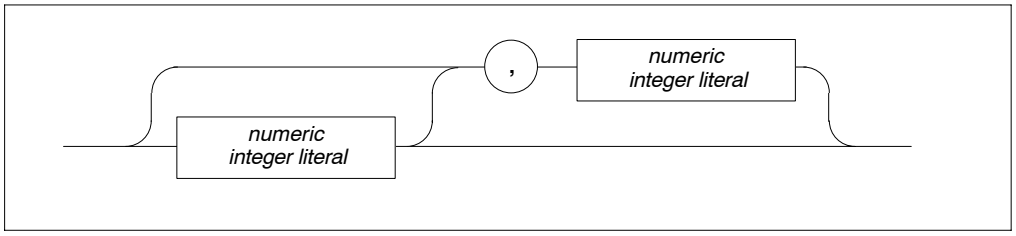
Description

Within the ADABAS logging clause it can be specified at which of the four log points ADABAS calls can be logged. Furthermore, it can be specified which buffer is to be displayed and to what extent. For details on ADABAS Command Logging refer to the chapter **Logging Facilities** earlier in this manual.

Specifications Clause



Buffer Area



AI IN/OUT – VO IN/OUT	Indicates the log point.
CB – SB FULL	Indicates the buffer to be displayed in full.
CB – SB *buffer area*	Indicates the buffer area to be displayed.

Limitations

None

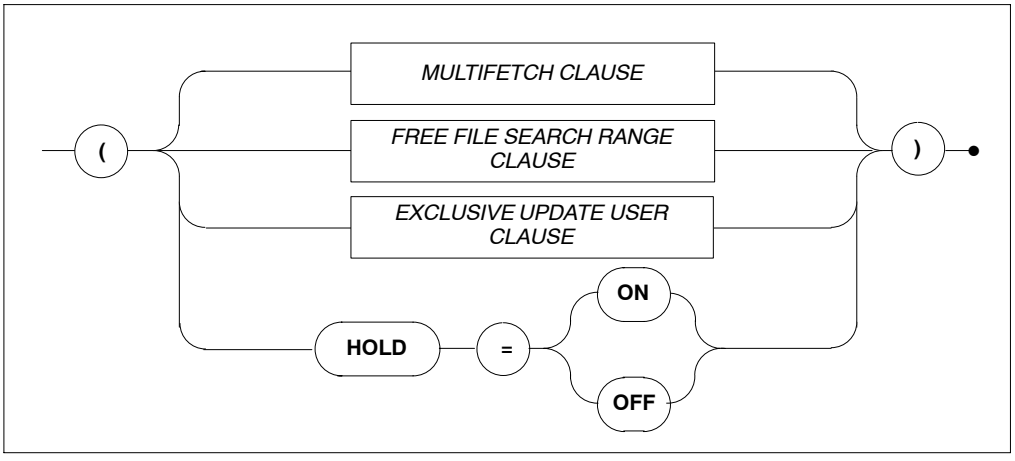
Example

```
GLOBAL BEGIN
ADABAS LOG (AI OUT (CB(10,10),SB(FULL),RB(FULL)))
END

ADABAS LOG (CLOG)
```

Does a compact logging at the “VO OUT” log point.

Features Clause

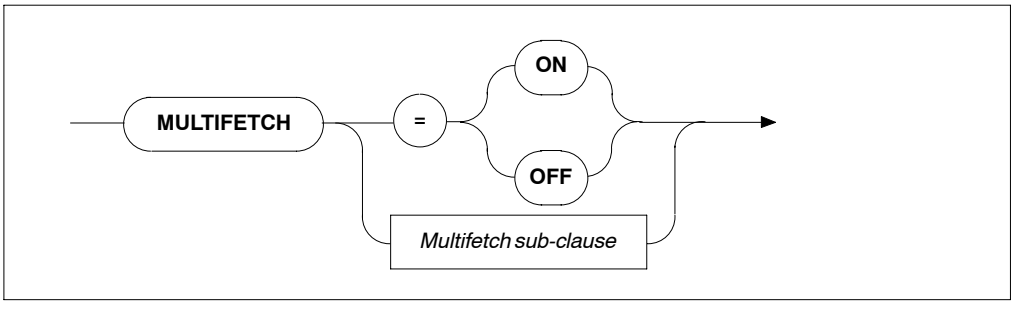
For information on the individual clauses refer to the description section directly below the relevant syntax diagrams.



HOLD = ON — Indicates that if a record is locked by another user, the application will wait until the record becomes available.

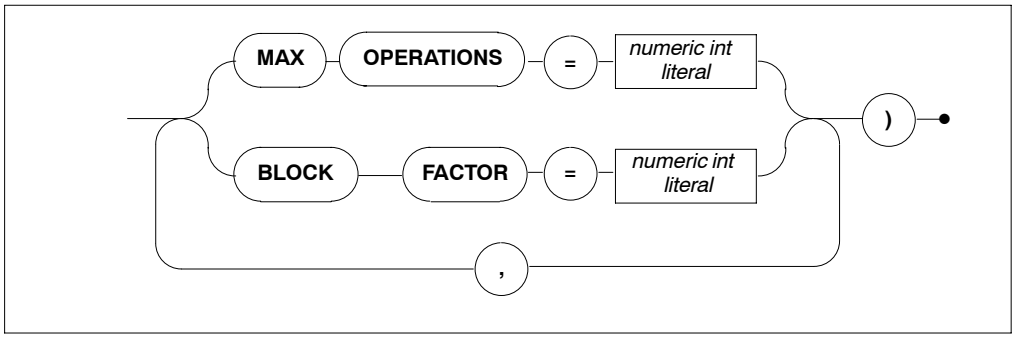
HOLD = OFF — Indicates that if a record is locked by another user, the application will NOT wait, but will get an appropriate response code.

Multifetch Clause



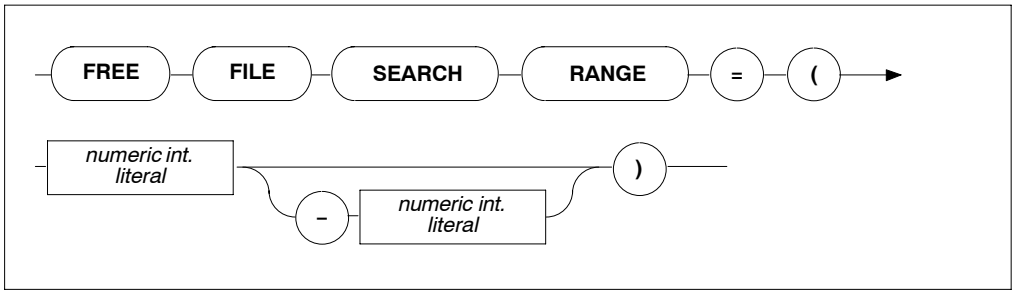
MULTIFETCH ON/OFF — Turns ON/OFF the ADABAS MULTIFETCH feature between ADABAS and the ADABAS SQL Server. If multifetching is not used, severe performance losses can be expected.

Multifetch Sub-clause



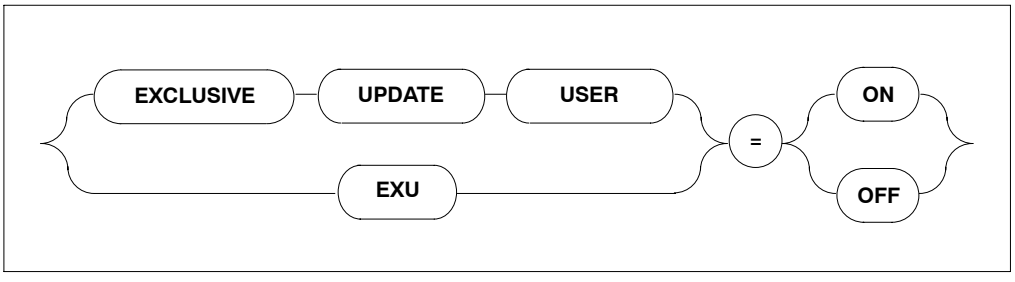
MULTIFETCH MAX OPERATIONS	Defines the number of ADABAS commands which can be used simultaneously with the MULTIFETCH logic. Minimum value : 4 Maximum value : 32 Default value : 8
MULTIFETCH BLOCK FACTOR	To optimize the data transfer between the ADABAS SQL Server and ADABAS, the MULTIFETCH logic of ADABAS is used. The BLOCK FACTOR defines how many records should be retrieved with one ADABAS call. Minimum value: 8 Maximum value: 512 Default value: 16

Free File Search Range Clause



FREE FILE SEARCH RANGE — Defines the range of file numbers that will be checked when creating a table or cluster. No files reserved for security should be within the specified range. This setting is used when the file number is not specified in the tablespace for a CREATE TABLE/CREATE CLUSTER statement or when one of the two default tablespaces is used (hardcoded/schema default tablespace). Specifying only the first value will result in a search starting at that number up to the highest file number supported by the particular ADABAS version.

Exclusive Update User Clause



EXCLUSIVE UPDATE USER — If set to ON, the user will have exclusive update rights according to the predefined EXU-List. See the original ADABAS manuals or the *ADABAS SQL Server Programmer's Guide*, **The ADABAS SQL Server and other SOFTWARE AG Products**. The default value is OFF.

Limitations

The ADABAS MAXIMUM OPERATIONS setting is only necessary in client/server mode.

Examples

```
GLOBAL BEGIN
ADABAS ( MULTIFETCH ( BLOCK FACTOR = 256 ) )
END

GLOBAL BEGIN
ADABAS ( HOLD = ON )
END
```

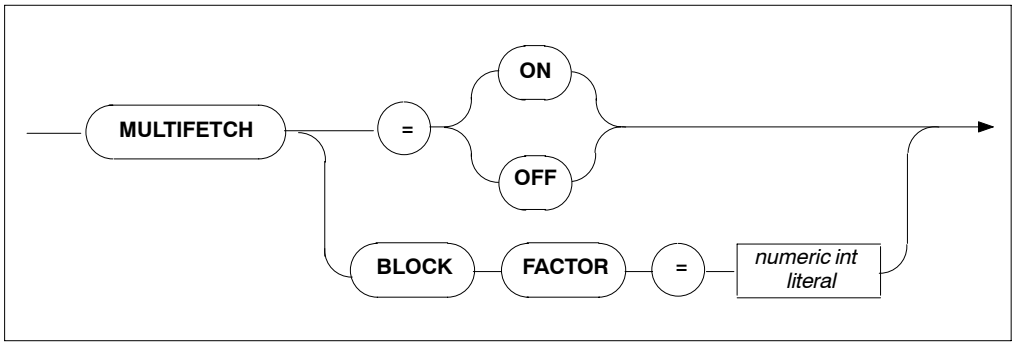
GLOBAL MULTIFETCH Settings

Function

Allows the setting of the following client-specific MULTIFETCH logic:

Syntax

Multifetch Clause



MULTIFETCH ON/OFF	Turns ON/OFF the MULTIFETCH feature of the client. Default value: ON
MULTIFETCH BLOCK FACTOR	Optimizes the data transfer between the client and the ADABAS SQL Server. The BLOCK FACTOR defines how many rows will be retrieved from the ADABAS SQL Server with one FETCH statement. Minimum value: 8 Maximum value: 512 Default value: 16

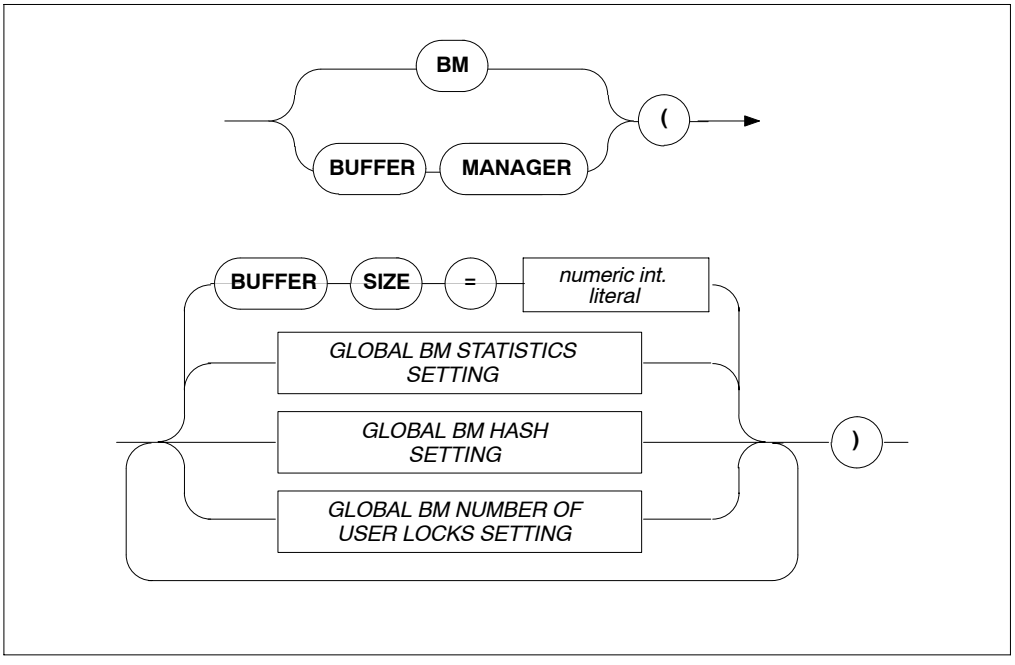
Note:
If a cursor has been defined to be updatable, the use of MULTIFETCH logic will be turned off automatically by the client.

GLOBAL BUFFER MANAGER (BM) Settings

Function

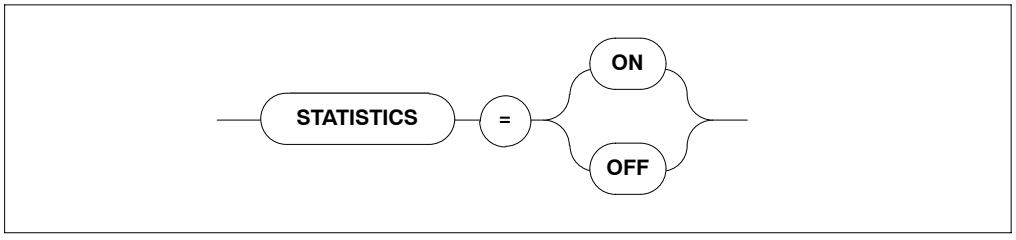
The ADABAS SQL Server catalog buffer stores the objects of the catalog. The buffer is a shared memory in a multi-user environment and a process local memory in a single-user environment.

Syntax



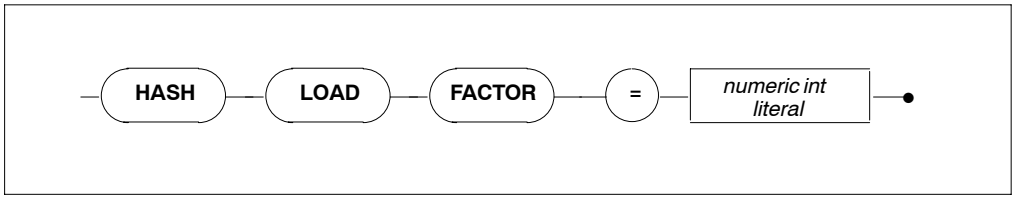
BUFFER SIZE — The size of the catalog buffer in bytes.
Default: 262 144 bytes.
Minimum: 32 768 bytes (also depends on the length of the hash table), for details refer to the section **GLOBAL BM HASH SETTINGS** below.

GLOBAL BM STATISTICS SETTING



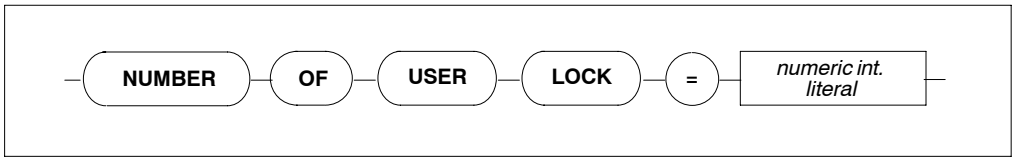
STATISTICS — Defines whether statistics are collected concerning the operation of the buffer manager (slows the system down). Default: OFF

GLOBAL BM HASH SETTING



HASH LOAD FACTOR — Defines the size of the hash table in the catalog buffer. The higher the hash load factor, the faster BM's access to the objects in the buffer.
Default: 20 %
20 % equals 4 KB if the buffer size is smaller than 64 KB
and 8 KB if the buffer size is larger than 64KB.

GLOBAL BM NUMBER OF USER LOCKS SETTING



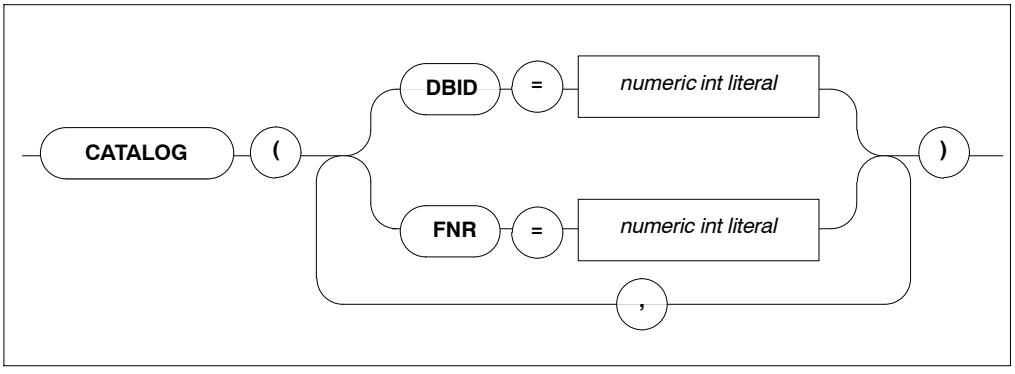
NUMBER OF USER LOCKS — Defines the default number of locks per user if no parameter was specified via BM_R_LOGON(). Default value is 256, minimum is 1 and maximum is 32767.

GLOBAL CATALOG Setting

Function

To set where the catalog can be found.

Syntax



Description

Defines in which ADABAS database and in which ADABAS file the catalog can be found. The catalog contains details about which tables and meta programs are available for the execution of SQL statements.

Default values are DBID = 1 and FNR = 13.
Minimum value is 1, maximum value is 255.

Limitations

For precompiler and runtime files and only necessary in LINKED-IN mode. In client/server mode, this setting is ignored.

The ADABAS database specified by DBID and the file specified by FNR must contain a valid catalog.

If more than one DBID/FNR is specified, only the last specified value is recognized.

Examples

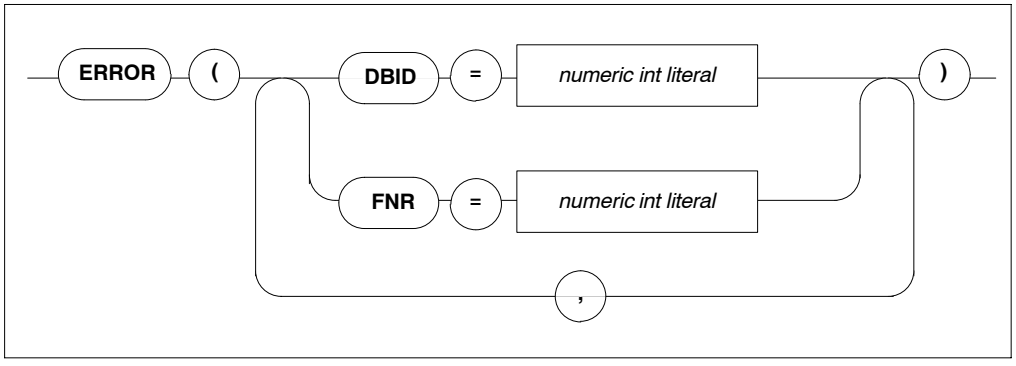
```
GLOBAL
BEGIN
     CATALOG ( DBID = 1, FNR = 13 )
END
```

GLOBAL ERROR Settings

Function

To set where the error messages can be found and to specify the desired language.

Syntax



Description

Defines in which ADABAS database and which ADABAS file the SQL error messages can be found.

Limitations

DBID and FNR are only necessary in client/server mode.

The ADABAS database specified by DBID and the file specified by the file number must contain the valid error messages.

If more than one DBID/FNR is specified, only the last specified value is recognized.

Examples

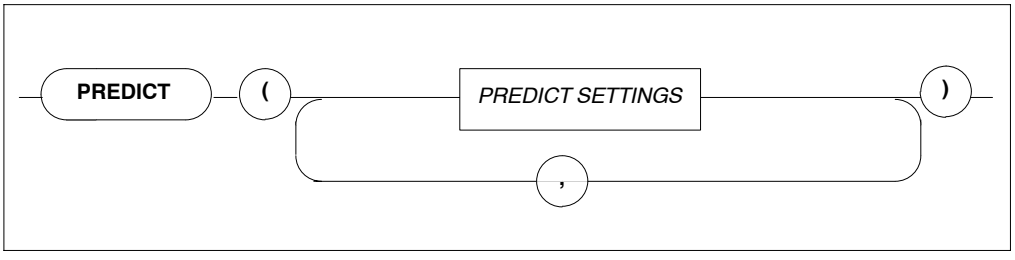
```
GLOBAL
BEGIN
     ERROR ( DBID = 13, FNR = 3 )
END
```

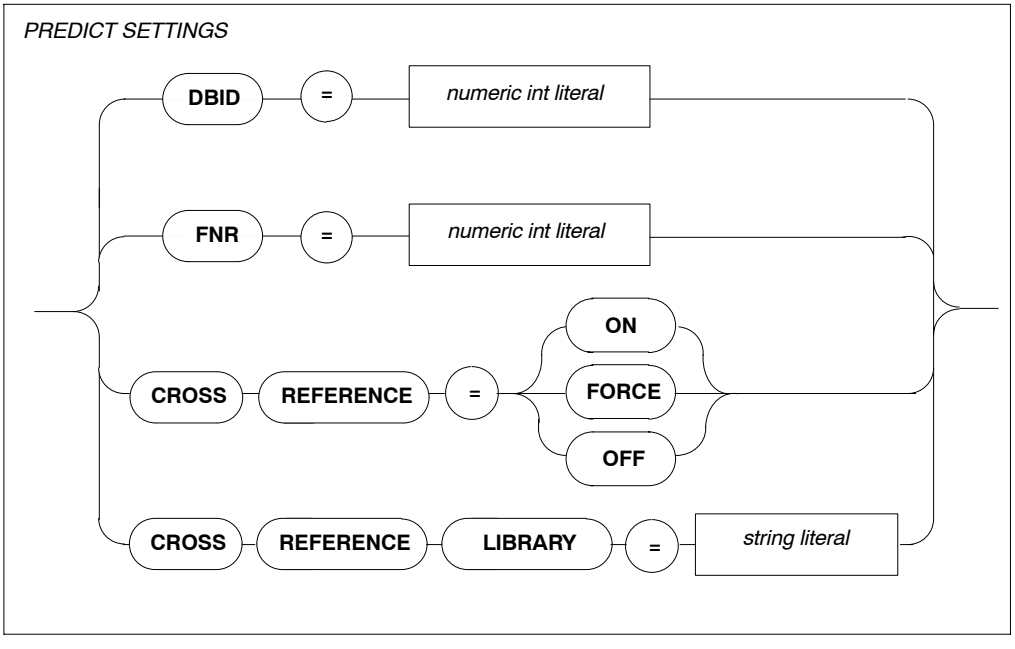
GLOBAL PREDICT Settings

Function

To set PREDICT-specific values which are valid across all parts of the system (from precompilation time to runtime).

Syntax





Description

DBID	Defines in which ADABAS database the PREDICT Cross Reference Library can be found. Minimum value: 1 Maximum value: 255 Default value: 1
FNR	Defines in which ADABAS file the PREDICT Cross Reference Library can be found. Minimum value: 1 Maximum value: 255 Default value: 113
CROSS REFERENCE = OFF/ON	Defines whether the active cross reference feature is to be used If set to ON, cross reference data for the host program will be stored in the appropriate PREDICT entries at the end of a precompilation process.
CROSS REFERENCE LIBRARY	Defines the cross reference library name. If a name is entered, this name has to be documented in PREDICT as well. If no name is entered, a default name will be taken.

Limitations

Note:
Interaction with PREDICT is not possible with all ADABAS SQL Server 1.4 versions.

DBID and FNR are only necessary in client/server mode.

The maximum length of a PREDICT Cross Reference Library name is 8 characters.

Examples

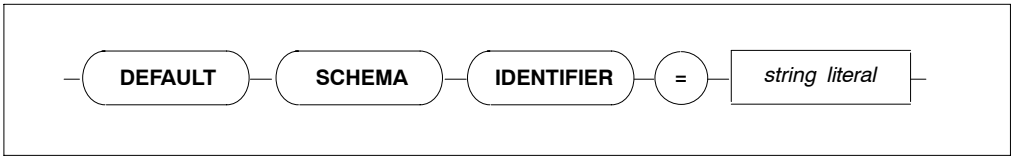
```
GLOBAL
BEGIN
     PREDICT (CROSS REFERENCE = ON)
END
```

GLOBAL DEFAULT SCHEMA IDENTIFIER Setting

Function

Sets the default schema identifier.

Syntax



Description

A table identified by a table identifier only is called an unqualified table specification. The default schema identifier is used to uniquely identify each unqualified table specification occurring in an SQL statement within a compilation unit. This is effective at precompile time for static embedded SQL statements and at runtime for statements processed dynamically via PREPARE or EXECUTE IMMEDIATE statements. The default schema identifier setting has to appear in the precompiler or runtime parameter files, respectively.

If a default schema identifier has not been set explicitly, the unqualified table name will be implicitly qualified by the user identifier set by a CONNECT statement. In the precompiler environment, this user identifier is derived from the operating system user name.

The string containing the default schema identifier is not case-sensitive and must be defined according to the rules of SQL identifiers.

Limitations

The length is limited to 32 characters.

Example

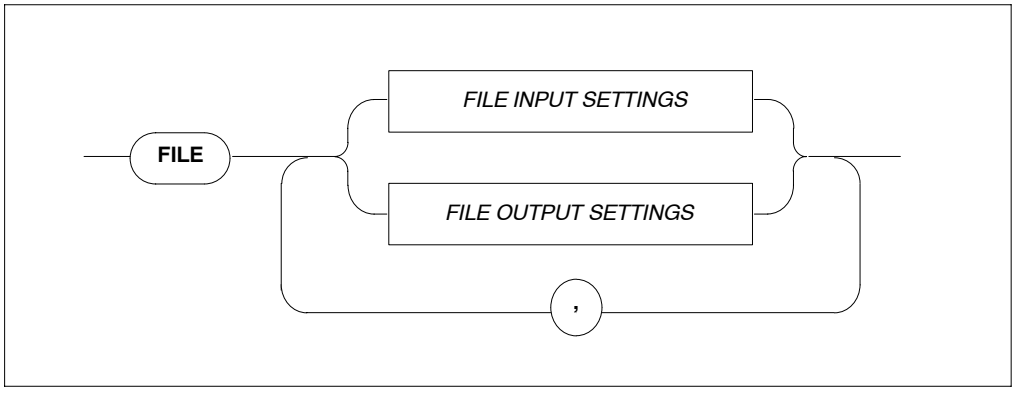
```
GLOBAL
BEGIN
     DEFAULT SCHEMA IDENTIFIER = “ESQ”
END

GLOBAL
BEGIN
     DEFAULT SCHEMA IDENTIFIER = “robert”
END
```

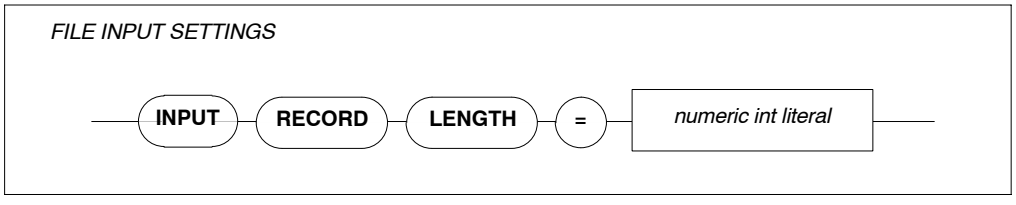
GLOBAL FILE Settings

Function

Sets the input/output record length.

Syntax





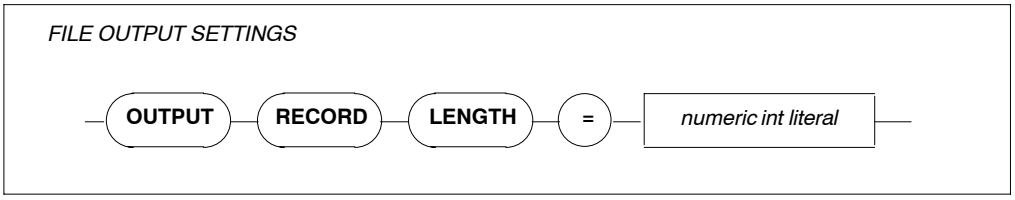
INPUT RECORD LENGTH Maximum record length (integer).

The input record length is used by PPL and the precompiler. The default value is 72 characters.

Example

Sets the record length to 132 characters.

```
FILE INPUT RECORD LENGTH = 132
```



OUTPUT RECORD LENGTH Maximum record length (integer).

The output record length is used by the precompiler. Default values: 256 characters for non-mainframe environments and 80 characters for mainframe environments.

Note:
On mainframe platforms, the record length of the output file can be specified beforehand and is not overwritten by the specification of the GLOBAL FILE parameter setting. This means that truncation occurs if the predefined output record length is shorter than the input record length.

Example

Sets the record length to 72 characters.

```
FILE OUTPUT RECORD LENGTH = 72
```

SERVER Settings

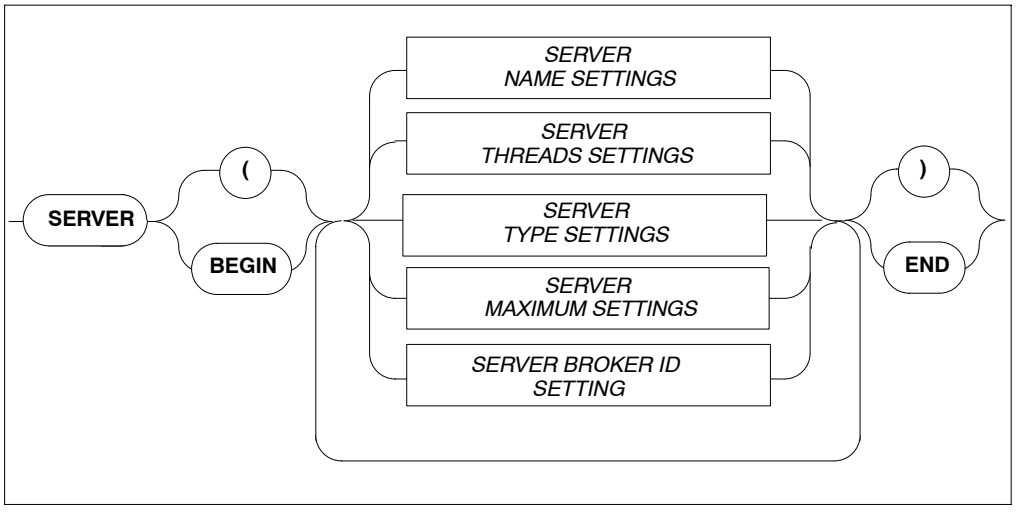
Function

The SERVER settings define the server-specific environment and are needed when running in client/server mode only. They are used during start-up of a server.

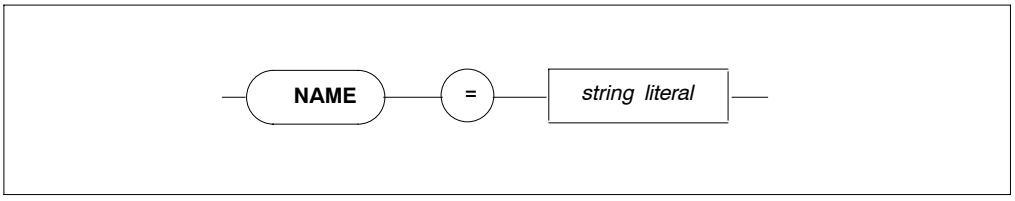
The types of settings for the SERVER section are:

- SERVER NAME SETTINGS
- SERVER THREAD SETTINGS
- SERVER TYPE SETTINGS
- SERVER MAXIMUM SETTINGS
- SERVER BROKER ID SETTING

Syntax

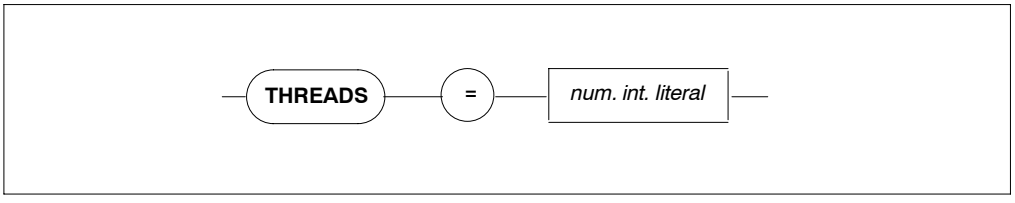


SERVER NAME Setting



Specifies the the name of the server, which must not be longer than 8 characters. Note that this parameter is not used on IBM platforms.

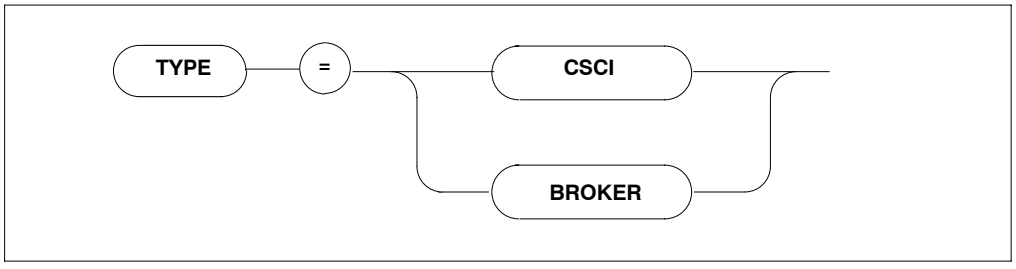
SERVER THREADS Setting



Specifies the number of threads per server. More threads mean more resources required.

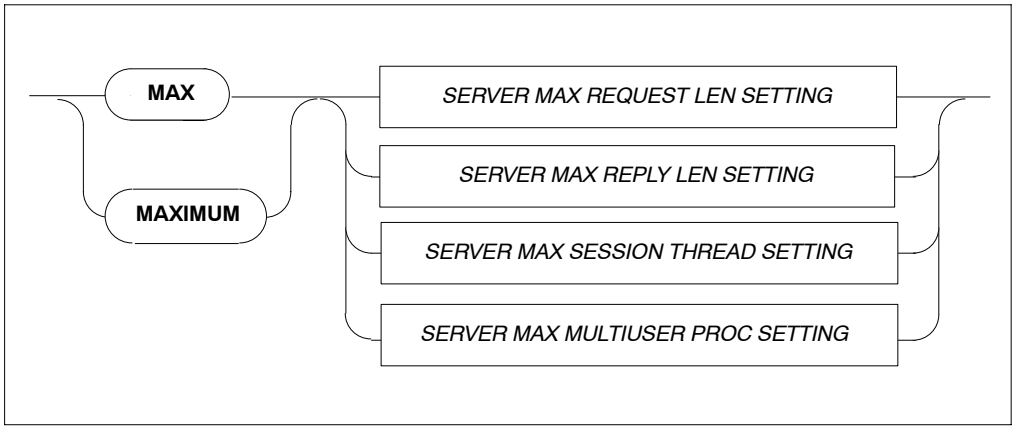
Default value: 5, minimum value: 0.

SERVER TYPE Setting



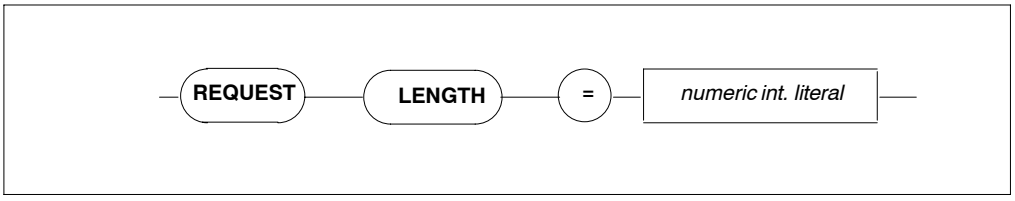
Specifies the type of client-server communications mechanism. The default communication mechanism is CSCI. Must be set to BROKER if the SERVER BROKER ID Setting is to be specified.

SERVER MAXIMUM Settings



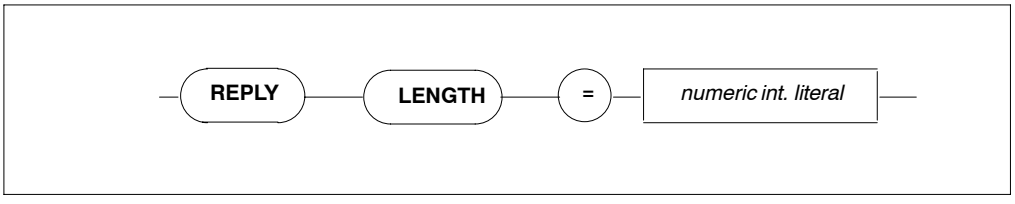
Sets the maximum values for the server.

SERVER MAXIMUM REQUEST LENGTH Setting



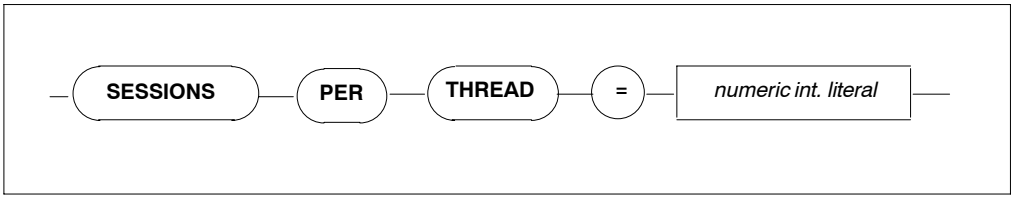
Specifies the maximum request length for the client-server communications. The default value is 8000 bytes, maximum length is 64000 bytes. Recommended minimum value for UNIX only: 32000 bytes.

SERVER MAXIMUM REPLY LENGTH Setting



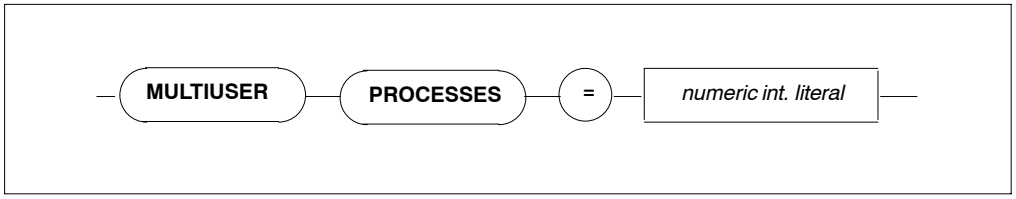
Specifies the reply length for the client/server communications. The default value is 8000 bytes, maximum length is 64000 bytes.

SERVER MAXIMUM SESSION THREAD Setting



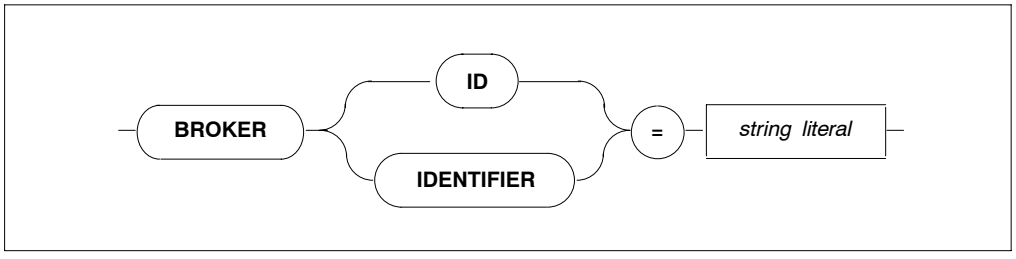
Specifies the maximum number of sessions per thread; each user process is one session. Zero means infinite (∞). For Version 1.4 the maximum number of sessions per thread must be 1.

SERVER MAXIMUM MULTIUSER PROCESSES Setting



Specifies the number of additional processes which may use the server e.g., precompiler. The default value is 64.

SERVER BROKER ID Settings



BROKER ID

Defines the name under which the communication between the ADABAS SQL Server and SOFTWARE AG's middleware communication protocol BROKER will take place. This setting will only take effect when the SERVER TYPE Setting is defined as BROKER.

B

APPENDIX B — SOFTWARE PROCESS AND ARCHITECTURE

Scope and Contents of this Appendix

This appendix is valid for OpenVMS platforms only and contains a graphical overview of the ADABAS SQL Server Software components and their interaction and a very brief description of their functions.

Remote Client Nodes

Remote clients use the logical name ESQLNK to communicate with the server. ESQLNK uses ENTIRE CSCI/BROKER to transport the client/server requests. Furthermore, ENTIRE CSCI/BROKER uses ENTIRE NET-WORK as its protocol. During execution of the first request, ESQLNK reads the Client Parameter File. The content of this file is passed to the server which overwrites the default client parameters on the server side.

Client Environment

In addition, local clients communicate with the server using the logical name ESQLNK. For local clients, ADABAS SQL Server offers the additional possibility of running in the so-called linked-in mode. In such a case, ESQLNK does not communicate with the server but rather uses the environment variable ESQTHS to execute the SQL requests in the application process context. The program ESQINT can be used to execute interactive SQL statements, also from remote nodes.

Precompiler Environment

ADABAS SQL Server offers a precompiler for the application language C. This precompiler scans C-application program sources containing embedded EXEC SQL statements and generates a source which can be directly compiled with the application language compiler. The generated sources contain host language calls to the ADABAS SQL Server, replacing the SQL statements in the original sources. The precompiler reads the Precompiler Parameter File to get information about the parameters to be used.

Server Environment

The server consists of one supervisor process ESQINI, which creates the required shared memory areas and then starts the thread processes ESQSRV. The thread processes dynamically load the environment variable ESQTHS, which executes the SQL requests. Each server thread process waits for incoming client requests passed by ENTIRE CSCI. The server ESQSRV receives a request and passes it to ESQTHS for execution. ESQTHS itself calls ADABAS using ADALNK.

Database Environment

The ADABAS SQL Server catalog and the user data are stored in an ADABAS database. Via ADALNK, it is also possible to access ADABAS servers which are not located on the current node. For this, ENTIRE NET-WORK is required.

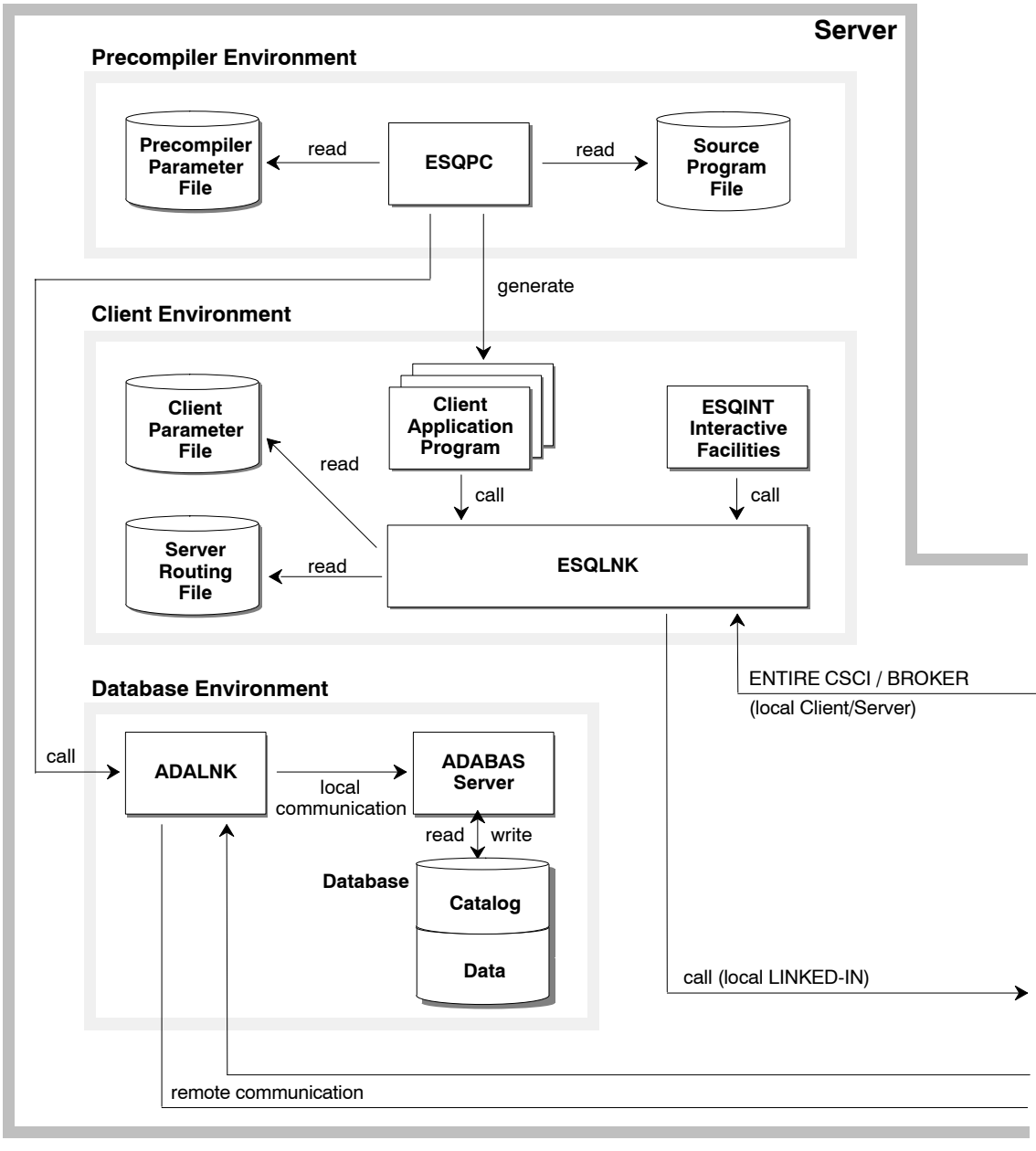
As already described above under section **Server Environment**, the process ESQINI is the supervisor of the server. This process is started by the command ESQSTART. First of all, ESQINI reads the Server Parameter File to establish the server parameters and the default client parameters. The shared memories for the Server Control Block and the Catalog Buffer are then created. The server parameters and the default client parameters are both stored in the Server Control Block.

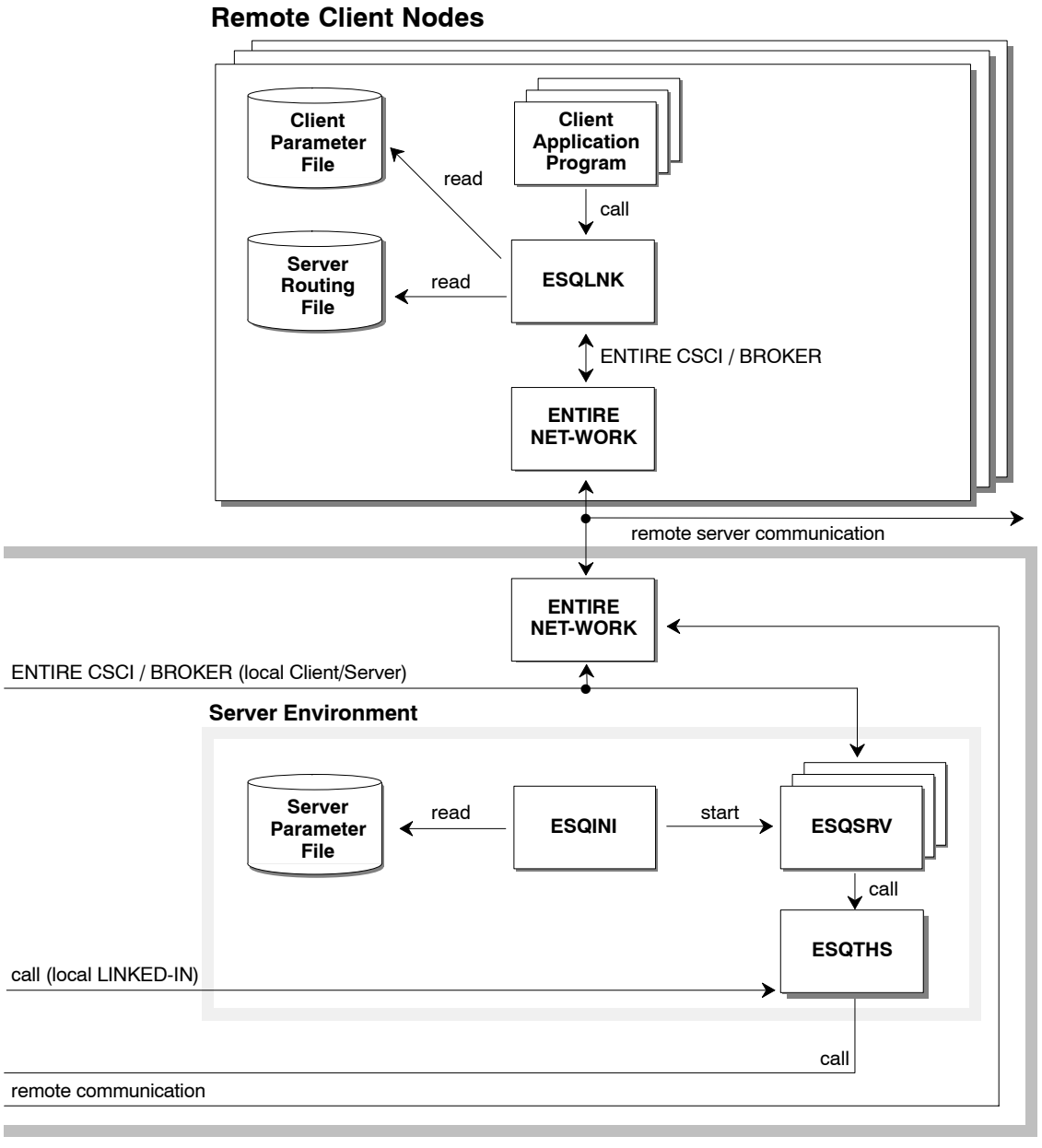
Then the thread processes with the program ESQSRV are started. The number of thread processes to be started can be specified in the Server Parameter File. The default is 5. The program ESQSRV attaches to the created shared memory areas. Each ESQSRV itself loads ESQTHS.EXE where the SQL request execution will be performed. After startup, all server thread processes wait for incoming requests passed by ENTIRE CSCI. For this purpose, the shared library SPI.EXE of ENTIRE CSCI is loaded.

If a client passes an SQL request to the server, then it is picked up by one thread process via ENTIRE CSCI. The program ESQSRV gives the request to the shared image ESQTHS.EXE via a call which executes it. Using the shared image ADALNK.EXE, database calls are performed.

General platform-dependent functionality is implemented in ESQVOLIB.EXE. This shared image is used by all programs of the ADABAS SQL Server. The shared image is part of the ADABAS SQL installation and stands for SOFTWARE AG's Virtual Operating System.

The following pages show an overview of the SOFTWARE AG components.


Server
Precompiler Environment
Precompiler Parameter File
read
ESQPC
read
Source Program File
generate
Client Environment
Client Parameter File
read
Client Application Program
call
ESQINT Interactive Facilities
call
Server Routing File
read
ESQLNK
ENTIRE CSCI / BROKER
(local Client/Server)
Database Environment
call
ADALNK
local communication
ADABAS Server
read
write
Database
Catalog
Data
call (local LINKED-IN)
remote communication


Remote Client Nodes
Client Parameter File
Client Application Program
read
call
Server Routing File
read
ESQLNK
ENTIRE CSCI / BROKER
ENTIRE NET-WORK
remote server communication
ENTIRE NET-WORK
ENTIRE CSCI / BROKER (local Client/Server)
Server Environment
Server Parameter File
read
ESQINI
start
ESQSRV
call
ESQTHS
call (local LINKED-IN)
call
remote communication

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W