Adabas

Adabas Installation for z/VSE

Version 7.4.4

September 2009

Table of Contents

1 Adabas Installation for z/VSE

This document is intended for those who plan or perform Adabas z/VSE installation, and for those who manage or maintain an Adabas database system (such as database administrators and systems programming personnel).

Notation `vrs` or `vr`: When used in this documentation, the notation `vrs` or `vr` stands for the relevant version, release, and system maintenance level numbers. For further information on product versions, see *Version* in the *Glossary*.

2 About This Document

This document provides information for installing and configuring Adabas Version 7.4 on the z/VSE system.

Operating system requirements are provided, as well as procedures for installing Adabas, for connecting Adabas to TP monitor subsystems such as Software AG's Com-plete or IBM's CICS, and for adding new I/O devices.

Other Documentation You May Need

The following Software AG documentation is referred to in this document and may be needed when installing Adabas:

- *Adabas Release Notes*
- *Adabas Operations*
- *Adabas DBA Tasks*
- *Adabas Triggers and Stored Procedures*
- *Adabas Command Reference*
- *Adabas Messages and Codes*
- *Adabas Utilities*
- *Adabas Security* (available only on written request from an authorized user site representative)

For Software AG's System Maintenance Aid (SMA) information, see the *System Maintenance Aid* documentation..

Notation `vrs` or `vr`: If used in the following document, the notation `vrs` or `vr` stands for the relevant version, release, and system maintenance level numbers. For further information on product versions, see *Version* in the *Glossary*.

3 Supported Environments

Before attempting to install Adabas, ensure that the host operating system is at the minimum required level.

Adabas Version 7.4 requires z/VSE Version 2, Releases 4, 5, 6, or 7.

Software AG provides Adabas support for the operating system versions supported by their respective manufacturers. Generally, when an operating system provider stops supporting a version of an operating system, Software AG will stop supporting that operating system version.

Although it may be technically possible to run a new version of Adabas on an old operating system, Software AG cannot continue to support operating system versions that are no longer supported by the system's provider.

If you have questions about support, or if you plan to install Adabas on a release, version, or type of operating system other than those included in the list above, consult Adabas technical support to determine whether support is possible, and under what circumstances.

4 Installation Procedure

This section describes the procedure for Adabas installation:

Installation Checklist

The following is an overview of the steps for installing Adabas on an z/VSE system.

Step	Description	Additional Information
1	Allocate DASD space for the Adabas libraries.	The libraries are restored from the installation tape. Refer to the section *Disk Space Requirements for Libraries*.
2	Allocate DASD space for the Adabas database.	For better performance, distribute the database files over multiple devices and channels. Refer to the section *Disk Space Requirements for the Database*.
3	Specify a z/VSE partition for running the Adabas nucleus.	Refer to the section *Adabas Nucleus Partition/Address Space Requirements*.
4	Define the library before restoration.	See section *Defining the Library*.
5	Restore the Adabas libraries.	See section *Installing the Adabas Release Tape*.
6	Install the Adabas SVC using the ADASIP program.	See section *Initializing the Adabas Communication Environment*.
7	Create the sample JCS job control for installing Adabas.	See section *Installing the Adabas Database*.
8	Customize and run job ADAIOOAL to link the Adabas options table for installation customization.	
9	Customize and catalog the two procedures ADAV*v*LIB and ADAV*v*FIL before placing them back in the procedure library. The following specific items must be customized: ■ file IDs for the database and libraries; ■ volumes for libraries and database files; ■ space allocation for database files	
10	Customize and run ADAFRM to allocate and format the Adabas database.	
11	Customize and run ADADEF to define global database characteristics.	
12	Customize and run ADALODE, ADALODV, and ADALODM to load the demo files.	
13	Customize and run JCLNUC to start the Adabas nucleus to test Adabas communications.	

Step	Description	Additional Information
14	Customize and run ADAREP in MULTI mode with the `CPLIST` parameter to test Adabas partition communication.	
15	Customize and run NATINPL and JCLONL to load the Adabas Online System, if used.	
16	Terminate the Adabas nucleus.	
17	Customize and run ADASAV to back up the database.	
18	Customize and run DEFAULTS to insert the ADARUN defaults with the ZAP utility.	
19	Install the required TP link routines for Adabas	See section ***Installing Adabas with TP Monitors***.

Preparing to Install Adabas

The major steps in preparing for Adabas installation are

- checking for the correct prerequisite system configuration; and
- allocating disk and storage space.

The following sections describe the nominal disk and storage space requirements, and how to allocate the space.

Disk Space Requirements for Libraries

The Adabas library requires a minimum of 3380/3390 disk space as shown below. A certain amount of extra free space has been added to the requirements for library maintenance purposes.

Library	3380 Tracks	3390 Tracks
Adabas Library	330	300

This space is needed for Adabas objects and phases as well as source and JCS samples.

Disk Space Requirements for the Database

The Adabas database size is based on user requirements. For more information, refer to *Adabas DBA Tasks*. Suggested sizes for an initial Adabas database, allowing for limited loading of user files and the installation of Natural, are as follows.

The minimum 3380 disk space requirements are:

Database Component	3380 Cylinders Required	3380 Tracks Required
ASSOR1 (Associator)	30	450
DATAR1 (Data Storage)	70	1050
WORKR1 (Work space)	10	150
TEMPR1 (temporary work space)	15	225
SORTR1 (sort work space)	15	225

The minimum 3390 disk space requirements are:

Database Component	3390 Cylinders Required	3390 Tracks Required
ASSOR1 (Associator)	26	390
DATAR1 (Data Storage)	62	930
WORKR1 (Work space)	10	150
TEMPR1 (temporary work space)	14	210
SORTR1 (sort work space)	14	210

Datasets Required for UES Support

The Software AG internal product libraries (BTE - basic technologies; and APS - porting platform) are required if you intend to enable a database for universal encoding service (UES) support. These libraries are now delivered separately from the product libraries.

For UES support, the following libraries must be loaded and included in the steplib concatenation:

```
BTE421.Laann
APS271.LIBR
APS271.L002
```

—where *aa* is LD, LC, or LS and *nn* is the load library level. If the library with a higher level number is not a full replacement for the lower level load library(s), the library with the higher level must precede those with lower numbers in the libdef concatenation.

Note: If you are using an Adabas load library prior to Version 7.2.2, it contains internal product libraries with an earlier version number and must be ordered below the current internal product libraries in the libdef concatenation.

Also for UES support, the following library must be loaded and included in the session execution JCL:

```
BTE421.LCnn
```

For information about setting up connections to UES-enabled databases through Entire Net-Work and ADATCP, see section ***Connecting UES-Enabled Databases.***

Disk Space Requirements for Internal Product Datasets

The minimum disk space requirements on a 3390 disk for the internal product libraries delivered with Adabas Version 7.4 are as follows:

Library	3390 Cylinders	3390 Tracks
BTE421.LD01	3	44
BTE421.LC01	16	236
BTE421.LS01	1	15
APS271.LIBR	8	109
APS271.L002	5	65

Adabas Nucleus Partition/Address Space Requirements

The Adabas nucleus requires at least 900-1024 KB to operate. The size of the nucleus partition may need to be larger, depending on the ADARUN parameter settings. Parameter settings are determined by the user.

Defining the Library

It is necessary to define the library before restoration. The following two examples show how VSAM and non-VSAM libraries are defined.

Defining a VSAM Library

The following is a job for defining a VSAM library:

```
// JOB DEFINE DEFINE VSAM V7 ADABAS LIBRARY
// OPTION LOG
// EXEC IDCAMS,SIZE=AUTO
DEFINE CLUSTER -
(NAME(ADABAS.Vvrs.LIBRARY) -
VOLUME(vvvvvv vvvvvv) -
NONINDEXED -
RECORDFORMAT(NOCIFORMAT) -
SHR(2) -
TRK(nnnnnn)) -
DATA (NAME(ADABAS.Vvrs.LIBRARY.DATA))
/*
// OPTION STDLABEL=ADD
// DLBL SAGLIB,'ADABAS.Vvrs.LIBRARY',,VSAM
// EXEC IESVCLUP,SIZE=AUTO
A ADABAS.Vvrs.LIBRARY
/*
// EXEC LIBR
DEFINE L=SAGLIB R=Y
DEFINE S=SAGLIB.ADAvrs REUSE=AUTO R=Y
LD L=SAGLIB OUTPUT=STATUS
/*
/&

—where
vvvvvv vvvvvv are the volumes for primary and secondary space.
nnnnnn is the number of tracks for primary and secondary space.
vrs is the Adabas version/revision/system maintenance level.
```

Notes:

1. For FBA devices the tracks (TRK...) operand is replaced by the blocks (BLOCKS...) operand.

2. SAGLIB is the name of the Adabas library. The name SAGLIB can be changed to suit user requirements.

Defining a Non-VSAM Library

The following is a job for defining a non-VSAM library:

```
// JOB DEFINE DEFINE NON-VSAM V7 ADABAS LIBRARY
// OPTION LOG
// DLBL SAGLIB,'ADABAS.Vvrs.LIBRARY',2099/365,SD
// EXTENT SYS010,vvvvvv,1,0,ssss,nnnn
// ASSGN SYS010,DISK,VOL=vvvvvv,SHR
// EXEC LIBR
DEFINE L=SAGLIB R=Y
DEFINE S=SAGLIB.ADAvrs REUSE=AUTO R=Y
LD L=SAGLIB OUTPUT=STATUS
/*
/&

—where
SYS010 is the logical unit for Adabas library.
vvvvvv is the volume for Adabas library.
ssss is the starting track or block for specified library.
nnnn is the number of tracks or blocks for specified library.
vrs is the Adabas version/revision/system maintenance level.
```

Restoring the ADAvrs LIBR File

Restore the ADA*vrs* LIBR file into sublibrary SAGLIB.ADA*vrs*. See the next section for information about preparing modules to run without the ESA option active.

Note: See the *Report of Tape Creation* that accompanies the tape to position the tape to the correct file.

If you have a license for one of the following Software AG products, restore the file into the appropriate sublibrary:

Product	File	Sublibrary
Adabas Caching Facility (ACF)	ACF*vrs*.LIBR	SAGLIB.ACF*vrs*
Adabas Parallel Services (ASM)	ASM*vrs*.LIBR	SAGLIB.ASM*vrs*
Adabas Delta Save Facility (ADE)	ADE*vrs*.LIBR	SAGLIB.ADE*vrs*

For information about installing these products, see the documentation for that product.

Using the ADAvrs LIBR File

Where applicable, modules for Adabas are shipped with `AMODE=31` active.

Storage Above or Below the 16-MB Limit

Adabas can acquire storage above the 16-megabyte addressing limit. This capability allows Adabas to acquire the buffer pool (`LBP`), work pool (`LWP`), format pool (`LFP`), and attached buffers (*`NAB`*) above 16 MB.

Where applicable, modules for Adabas are shipped with `AMODE=31` active. If you prefer to have buffers placed below the 16-megabyte limit, ADARUN must be relinked with `AMODE=24`.

User Program Execution in AMODE=31 and RMODE=ANY

Programs that will execute `AMODE=31` or `RMODE=ANY` must be relinked with the new ADAUSER object module.

In addition, because the IBM VSE LOAD macro cannot be issued in `RMODE=ANY`, the IBM VSE CD-LOAD macro must be used. Therefore, the zap to change the ADAUSER CDLOAD to the LOAD macro cannot be used. See the section describing ***z/VSE User Zaps*** for more information.

Installing Adabas in a VSE VM Guest System

Adabas may be installed and executed on a VSE system that runs as a guest under VM.

When running Adabas in this environment, the CPUID of the guest VSE system must not contain a value of X‘FFFFFFFF’. It it does, the Adabas nucleus terminates abnormally during command queue processing.

Installing the Release Tape

This section explains how to:

- copy dataset COPY.JOB from tape to disk
- modify this member according to your local naming conventions

The JCL in this member is then used to copy all data sets from the tape to disk.

If the datasets for more than one product are delivered on the tape, the member COIPYTAPE.JOB contains JCL to unload the data sets for all delivered products from the tape to your disk, except the data sets that you can directly install from tape, for example, Natrural INPL objects.

You can use the modified dataset to copy all datasets from tape to disk. You will then need to perform the individual install procedure for each component.

Note: If you are using SMA, please refer to Installing Software AG Products with SMA in the *System Maintenance Aid* documentation. If you are not using SMA, please follow the instructions below.

- Step 1: Copy Data Set COPYTAPE.JOB From Tape To Disk
- Step 2: Modify COPYTAPE.JOB
- Step 3: Submit COPYTAPE.JOB

Step 1: Copy Data Set COPYTAPE.JOB From Tape To Disk

The data set COPYTAPE.JOB (file 5) contains the JCL to unload all other existing data sets from tape to disk. To unload COPYTAPE.JOB, use the following sample JCL:

```
* $$ JOB JNM=LIBRCAT,CLASS=0, +
* $$ DISP=D,LDEST=(*,UID),SYSID=1
* $$ LST CLASS=A,DISP=D
// JOB LIBRCAT
* *****************************************
* CATALOG COPYTAPE.JOB TO LIBRARY
* *****************************************
// ASSGN SYS004,nnn <------ tape address
// MTC REW,SYS004
// MTC FSF,SYS004,4
ASSGN SYSIPT,SYS004
// TLBL IJSYSIN,'COPYTAPE.JOB'
// EXEC LIBR,PARM='MSHP; ACC S=lib.sublib' <------- for catalog
/*
// MTC REW,SYS004
ASSGN SYSIPT,FEC
/*
/&
* $$ EOJ

--- where
 nnn is the tape address, and lib.sublib is the library and sublibrary of the catalog.
```

Step 2: Modify COPYTAPE.JOB

Modify COPYTAPE.JOB to conform with your local naming conventions and set the disk space parameters before submitting this job.

Step 3: Submit COPYTAPE.JOB

Submit COPYTAPE.JOB to unload all other data sets from the tape to your disk.

Initializing the Adabas Communication Environment

Communication between the Adabas nucleus residing in a VSE partition and the user (either a batch job or TP monitor such as Com-plete or CICS) in another partition is handled with an Adabas SVC (supervisor call).

The program ADASIP is used to install the Adabas SVC. The system can run ADASIP to dynamically install the SVC without an IPL. Special instructions apply when using VSE with the Turbo Dispatcher as described in the next section below.

For information about messages or codes that occur during the installation, refer to the *Adabas Messages and Codes* documentation.

Installing the Adabas SVC with Turbo Dispatcher Support

The Adabas SVC module supports the IBM Turbo Dispatcher environment available with z/VSE 2.1 and above.

In a Turbo Dispatcher environment, the Adabas SVC runs in parallel mode when entered. Adabas processes multiple SVC calls made by users in parallel.

ADASIP Processing

To enable Turbo support, ADASIP installs a VSE first-level interrupt handler (ADASTUB) that screens all SVCs. When ADASTUB finds an Adabas SVC, it passes control directly to the Adabas SVC.

If your system is capable of running the Turbo Dispatcher and you do not want to run a particular SVC through the Turbo interface, you can set the UPSI flag V to 1 to exclude a particular SVC from use through the Turbo interface. See the ADASIP UPSI statement.

Effective with Version 7.4 of ADASIP, you can activate the ADABAS SVC with multiple CPUs active by specifying UPSI C. ADASIP will dynamically de-activate and re-activate the CPUs if required. If multiple CPUs are active and the UPSI C has not been specified, the following messages will be displayed:

```
ADASIP60 Only 1 CPU can be active during ADASIP
ADASIP79 Should we stop the CPUs? (yes/no)
```

Answering yes to this message will allow activation to occur; the CPUs will be dynamically deactivated and re-activated. Answering no will terminate ADASIP.

The ADASTUB module is installed only once per IPL process. On the first run of a successful ADASIP, the following set of messages are returned:

```
ADASIP63 ADASTUB Module Loaded at nnnnnnnn
ADASIP78 VSE Turbo Dispatcher Version nn
ADASIP69 Turbo Dispatcher Stub A C T I V E
```

When running ADASIP for subsequent Adabas SVC installations, the following message is displayed for information only:

```
ADASIP74 Info : Stub activated by previous ADASIP
```

When dynamically re-installing an Adabas SVC that was previously installed with Turbo Dispatcher support, execute a SET SDL for the Adabas SVC only. Do not execute the SET SDL for ADANCHOR a second time.

Note: Repeated re-installations of an Adabas SVC without an IPL may result in a shortage of 24-bit GETVIS in the SVA.

Running ADASIP

ADASIP requires a prior SET SDL for the SVC, and therefore must run in the BG partition. To install the Adabas SVC without an IPL, execute the following JCS in BG.

Notes:

1. When using the EPAT Tape Management System, EPAT must be initialized before running ADASIP.
2. At execution time, the ADASIP program determines if a printer is assigned to system logical unit SYSLST. If no printer is assigned, messages are written to SYSLOG instead of SYSLST.

For information about the ADASIP parameters, see the section *ADASIP Execution*.

To automatically install the Adabas SVC during each IPL, insert the following JCS (or its equivalent) into the ASI BG JCS procedure immediately before the START of the POWER partition where

nn	is the number of IDT entries
suffix	is the optional two-byte suffix for the z/VSE SVC name to be loaded by ADASIP. The previous VSE SVC version must be linked with a different suffix.
svc	is an available SVC number in your z/VSE system to be used as the Adabas SVC.
volume	is the specified volume for the Adabas library.
vrs	is the Adabas version/revision/system maintenance level.

Without Turbo Dispatcher Support

The following sample is available in member ADASIP.X:

```
// DLBL SAGLIB,'ADABAS.Vvrs.LIBRARY'
// EXTENT SYS010,volume
// ASSGN SYS010,DISK,VOL=volume,SHR
// LIBDEF PHASE,SEARCH=SAGLIB.ADAvrs
SET SDL
ADASVCvr,SVA
/*
// OPTION SYSPARM='svc,suffix' SVC NUMBER
// UPSI 00000000 UPSI OPTIONS FOR ADASIP
// EXEC ADASIP,PARM='NRIDTES=nn'
```

With Turbo Dispatcher Support

The following sample is available in member ADASIPT.X:

```
// JOB ADASIPT INSTALL THE ADABAS SVC (TURBO)
// OPTION LOG,NOSYSDUMP
// DLBL SAGLIB,'ADABAS.Vvrs.LIBRARY'
// EXTENT SYS010,volume
// ASSGN SYS010,DISK,VOL=volume,SHR
// LIBDEF PHASE,SEARCH=SAGLIB.ADAvrs
SET SDL
ADASVCvr,SVA
ADANCHOR,SVA
/*
// OPTION SYSPARM='svc,suffix' SVC NUMBER
// SETPFIX LIMIT=100K REQUIRED; SEE NOTE 2
// UPSI 00000000 UPSI OPTIONS FOR ADASIP
// EXEC ADASIP,PARM='NRIDTES=nn'
```

Notes:

1. A `SETPFIX` parameter is required with Turbo Dispatcher support to page fix ADASIP at certain points in its processing. A value of 100K should be adequate.
2. The SET SDL statement for ADANCHOR is required for Turbo Dispatcher support. This is in addition to the SET SDL statement for ADASVC*vr*.

Finding an Unused SVC

Adabas requires an entry in the VSE SVC table. To find an unused SVC, use one of the following methods:

Method 1

Set the S flag specified in the UPSI for ADASIP to create a list of used and unused SVCs in the VSE SVC table.

Method 2

Obtain a listing of the supervisor being used.

Using the assembler cross-reference, locate the label SVCTAB; this is the beginning of the VSE SVC table. The table contains a four-byte entry for each SVC between 0 and 150 (depending on the VSE version).

Locate an entry between 31 and 150 having a value of ERR21. This value indicates an unused SVC table entry. Use the entry number as input to ADASIP.

Loading a Secondary Adabas SVC

You can optionally specify a suffix to indicate the version of an SVC, as shown in the previous JCS examples. This allows you to run two different versions of the SVC. Before specifying a suffix, however, you must have previously linked the second version of the SVC. In addition, you must have performed a SET SDL operation on the new SVC's name (for example, ADASVC*xx*).

To optionally specify a different Adabas SVC using ADASIP, specify the SVC suffix (the last two bytes in the form, ADASVC*xx*), as follows:

```
// OPTION SYSPARM='svc,xx'
—where xx is the two-byte suffix of the new SVC.
```

ADASIP Execution Parameters

This section describes the ADASIP execution parameters.

OPTION SYSPARM= Statement

An optional correction (zap) can be applied to the Adabas ADASIP program to insert the default SVC so that no SYSPARM need be specified. See the section ***Applying Zaps***.

SVC	The Adabas SVC number chosen must be unused by VSE or any other third party products (see the section ***Finding an Unused SVC***).
SUFFIX	An optional two-byte value used to load a new version of the Adabas VSE SVC (see the section ***Loading a Secondary Adabas SVC)***..

UPSI Statement

// UPSI DSxTVOCG

Setting the UPSI byte is the user's responsibility. If the UPSI byte is not set, the SVC installation executes normally.

The UPSI byte is used to select the following options:

Option	If option is set to 1
D	ADASIP dumps the Adabas SVC and ID table using PDUMP. This option should be used only after the SVC is installed.
S	ADASIP dumps the VSE SVC table and indicates whether each SVC is used or unused. No SVC number is required when using this function of ADASIP.
T	ADASIP dumps the VSE SVC table and the VSE SVC mode table.
V	The SVC is excluded from use through the Turbo interface.
O	Override the messages that ask if you wish to stop the processors when more than one processor is active. If you choose to override, the processors will be automatically stopped during ADASIP execution and restarted upon ADASIP termination.
C	ADASIP will deactivate and reactivate the CPUs if more than one CPU is active.
G	ADASIP will display SYSTEM GETVIS allocation totals.

NRIDTES PARM= Option

The size of the ID table default supports up to 10 Adabas targets. However, the ADASIP program will allow you to increase this number by using this new option of the PARM operand on the EXEC card. To increase the size of the ID table to *nn* entries, specify the following when executing ADASIP:

// EXEC ADASIP,PARM='NRIDTES=*nn*'

where *nn* is the number of databases to be supported. Refer to the section ***Acquiring Storage for the ID Table*** for information about calculating the correct value for *nn*.

DMPDBID PARM= Option

This ADASIP option allows snap dumps of the Adabas command queue for a specified database ID (DBID). The dump is written to SYSLST. The OPTION SYSPARM statement must specify the SVC number to perform the snap dump. For example, to perform a snap dump of the database 5 command queue, issue:

```
// OPTION SYSPARM='svc,suffix'
// EXEC ADASIP,PARM='DMPDBID=5'
```

Runtime Display

When ADASIP is run, the ADASIP00 message displays the current system level.

```
ADASIP00 ...ADABAS V7 VSE SIP STARTED
SIP IS RUNNING UNDER VSE/systype-mode
ADASIP00 ... (yyyy-mm-dd, SM=sm-level, ZAP=zap-level)
ADASIP00 ... SIP IS RUNNING UNDER OSYS LEVEL Vnnn
ADASIP00 ... SIP IS LOADING ADABAS SVC LEVEL Vnnn
ADASIP00 ... ADASIP IS LOADING ADABAS SVC AMODE=amode
```

Installing the Adabas Database

This section describes installation of the Adabas database for z/VSE systems. Note that all applicable early warnings and other fixes must first be applied. For descriptions of any messages or codes that occur, refer to the *Adabas Messages and Codes* documentation.

- Prepare the Installation Sample JCS for Editing
- Modify, Assemble, and Link the Adabas Options Table
- Catalog Procedures for Defining Libraries and the Database

- Install a New Database

Prepare the Installation Sample JCS for Editing

Note: This step is only necessary if the library cannot be edited directly.

The following sample installation job is available in member INSTALL.X.

Run the following job to load the installation samples:

```
* $$ JOB JNM=PUNINST,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
* $$ PUN CLASS=p,DISP=D
// JOB PUNINST INSTALL SAMPLES FOR ADABAS
// OPTION LOG
// DLBL SAGLIB,'ADABAS.Vvrs.LIBRARY'
// EXTENT SYS010
// ASSGN SYS010,DISK,VOL=volume,SHR
// EXEC LIBR
ACCESS SUBLIB=SAGLIB.ADAvrs
PUNCH ADAPROC.X /* PROCS FOR FILE AND LIBRARY DEFINITIONS */
PUNCH ADAIOOAL.X /* ADABAS OPTIONS TABLE CUSTOMIZATION */
PUNCH ADASIP.X /* ADASIP JOB (NON-TURBO DISPATCHER) */
PUNCH ADASIPT.X /* ADASIP JOB (TURBO DISPATCHER) */
PUNCH ADAFRM.X /* SAMPLE ADAFRM JOB */
PUNCH ADADEF.X /* SAMPLE ADADEF JOB */
PUNCH ADALODE.X /* LOAD DEMO FILE EMPLOYEES */
PUNCH ADALODV.X /* LOAD DEMO FILE VEHICLES */
PUNCH ADALODM.X /* LOAD DEMO FILE MISC */
PUNCH ADANUC.X /* SAMPLE NUCLEUS STARTUP */
PUNCH ADAREP.X /* SAMPLE ADAREP JOB */
PUNCH NATINPL.X /* SAMPLE NATINPL TO INSTALL AOS */
/*
/&
* $$ EOJ

—where
p is the output class for punch
volume is the specified volume for the Adabas library.
vrs is the Adabas version/revision/system maintenance level.
```

Once the selected members in the INSTALL job are within the local editor facility, the customization can begin.

Modify, Assemble, and Link the Adabas Options Table

Customize and run job ADAIOOAL to assemble and link the Adabas options table for installation customization.

The following describes the IORDOSO macro, which must be assembled and linked to the Adabas sublibrary as PHASE ADAOPD. The member X.ADAIOOAL shipped with Adabas can be used for this purpose.

IORDOSO Macro Overview

The IORDOSO macro allows you to customize Adabas operation in the following areas:

- Loading phases;
- IDRC compaction support for 3480 and 3490 tape devices;
- Interfaces to VSE disk space managers such as DYNAM/D;
- Interfaces to VSE tape managers such as DYNAM/T
- An option controlling how the system writes to fixed block addressing (FBA) devices;
- An option to write printer (PRINT and DRUCK) files under either DTFPR or DTFDI control;
- GETVIS message printing;
- Optional job exit processing;
- Options for controlling the creation of VSE JCS with the Adabas Recovery Aid utility ADARAI;
- Sequential file processing under VSAM/SAM;
- Input device control with SYS000 assignment;
- Name of external sort program.

IORDOSO Macro Parameters

CDLOAD

Parameter	Description
`CDLOAD={ NO | YES }`	Determines whether Adabas uses the CDLOAD (SVC 65) or the LOAD SVC (SVC 4) to load modules.

COMPACT

Parameter	Description
`COMPACT={ NO | YES }`	If a sequential protection log (SIBA) is assigned to a 3480 or 3490 tape device, `COMPACT=YES` writes the SIBA in IDRC compaction mode. The default is `COMPACT=NO` (no compaction).

DISKDEV

Parameter	Description
`DISKDEV=`*`devtype`*	Specifies the device type on which space for sequential files is to be allocated (see notes 1 and 2).

DISKMAN

Parameter	Description
`DISKMAN={ NO | YES }`	Indicates to Adabas that a VSE disk space manager such as DYNAM/D is active. If `DISKMAN=YES` is specified, `DISKDEV` or `DISKSYS` must also be specified.

DISKSYS

Parameter	Description
`DISKSYS=`*`sysnum`*	When a disk space manager such as DYNAM/D is present, use the `DISKSYS` parameter to specify the programmer logical unit (LUB). The specified value, which can be from 000 to 255, determines the disk device type for the SAM or VSAM sequential file. There is no default value.

DISKTYP

Parameter	Description
`DISKTYP=`*`text`*	This parameter is for information only, and is processed as a comment. The value *`text`* can be up to 16 bytes long.

DTFDI

Parameter	Description
`DTFDI={ NO | YES }`	`DTFDI=YES` directs the PRINT (SYSLST) and DRUCK (SYS009) output to be device-independent, causing all ADARUN, ADANUC, session, statistics, and utility output to be written to where SYSLST or SYS009 is assigned (printer, disk, or tape). When you specify `DTFDI=YES`, the `PRTDSYS` and `PRTRSYS` parameters are ignored. If you specify `DTFDI=NO` (the default), output is directed using `DTFPR`.

FBAVRF

Parameter	Description
`FBAVRF={ NO | YES }`	FBA users only: the `FBAVRF` parameter specifies whether Adabas does WRITE VERIFY I/O commands, or normal WRITEs. If `FBAVRF=YES` is specified, WRITE VERIFY I/Os are performed; the default is normal WRITE operation.

GETMMSG

Parameter	Description
`GETMMSG={ NO | YES }`	Determines whether or not VSE ADAIOR GETMAIN (GETVIS) messages are printed. No printing is the default.

JBXEMSG

Parameter	Description
`JBXEMSG={ NO| YES | PRT }`	The z/VSE parameter `JBXEMSG` determines whether job exit utility error messages are printed (`JBXEMSG=PRT`), displayed (`JBXEMSG=YES`, the default), or not presented (`JBXEMSG=NO`).

JBXIMSG

Parameter	Description
`JBXIMSG={ NO | YES | PRT }`	The z/VSE parameter `JBXIMSG` determines whether job exit utility information messages are printed (`JBXIMSG=PRT`, the default), displayed (`JBXIMSG=YES`), or not presented (`JBXIMSG=NO`).

JOBEXIT

Parameter	Description
`JOBEXIT={ NO | YES }`	`JOBEXIT=YES` activates the Adabas job exit utility, allowing any * SAGUSER job control statements to override the normal job input.

PFIXRIR

Parameter	Description
`PFIXRIR={ NO | YES }`	Specifies whether or not ADAMPM is page fixed in storage during the nucleus initialization process.

PRTDSYS

Parameter	Description

`PRTDSYS={ sysnum | SYSLST }`	Specifies the programmer logical unit (LUB), and may be any number 000 - 254. If specified, the *sysnum* value replaces the default where the ADARUN messages are printed, which is SYSLST.
	
	The value specified by *sysnum* must be assigned in the partition before running the ADARUN program. For example:
	
	`PRTDSYS=050`
	`.`
	`// ASSGN SYS050,PRINTER`

PRTRSYS

Parameter	Description
`PRTRSYS={ sysnum | SYS009 }`	Specifies the programmer logical unit (LUB). If specified, this *sysnum* value replaces the default where the Adabas utility (DRUCK) messages are printed, which is SYS009.

RAIDASG

Parameter	Description
`RAIDASG={ NO | YES }`	`RAIDASG=YES` specifies that the Adabas Recovery Aid (ADARAI) is to create VSE disk ASSGN statements. Such statements are sometimes not needed with a VSE disk manager facility.

RAITASG

Parameter	Description
`RAITASG={ NO | YES }`	`RAITASG=YES` specifies that the Adabas Recovery Aid (ADARAI) is to create VSE tape ASSGN statements. Such statements are sometimes not needed with a VSE tape manager facility.

SORTPGM

Parameter	Description
`SORTPGM={ sortpgm | SORT }`	Specifies the name of the external sort program to be invoked during execution of the Adabas changed-data capture utility ADACDC. The default name is SORT.

SYS000O

Parameter	Description

`SYS000O={ NO | YES }`	If `SYS000O=NO` (the default) is specified, the ADARUN statements are read normally. If `SYS000O=YES` is specified, Adabas determines the correct DTF for opening, depending on where SYS000 is assigned, as follows:
	
	`Medium - SYS000  DTF Type`
	
	`Card              DTFCD`
	`Disk              DTFSD`
	`Tape              DTFMT`

TAPEDEV

Parameter	Description
`TAPEDEV=devtype`	Specifies the tape device type on which sequential files are written (see notes 1 and 3).

TAPEMAN

Parameter	Description
`TAPEMAN={ NO | YES }`	Indicates that a VSE tape manager such as DYNAM/T is active. If `TAPEMAN=YES` is specified, `TAPEDEV` or `TAPESYS` must also be specified.

TAPESYS

Parameter	Description
`TAPESYS=sysnum`	When a tape manager such as DYNAM/T is present, this parameter is used to specify the programmer logical unit (LUB). The specified value, which can be any value from 000 to 255, determines the tape device type for the sequential file (see note 1). There is no default value.

TAPETYP

Parameter	Description
`TAPETYP=`*`text`*	This parameter is for information only, and is processed as a comment. The value *`text`* can be up to 16 bytes long.

VSAMDEV

Parameter	Description
`VSAMDEV=`*`devtype`*	Specifies the disk device type on which VSAM/SAM space is to be allocated (see notes 1 and 2).

VSAMSEQ

Parameter	Description
`VSAMSEQ={ NO | YES }`	Specifies whether sequential files are to be under the control of VSAM/SAM software. If `VSAMSEQ=YES` is specified, either `VSAMDEV` or `VSAMSYS` must also be specified.

VSAMSYS

Parameter	Description
`VSAMSYS=`*`sysnum`*	Specifies the programmer logical unit (LUB). The specified value, which can from 000 to 255, determines the device type for the sequential file written to VSAM/SAM space (see note 1). There is no default value.

Notes:

1. Adabas requires device type information when opening files. However, there may be situations where the device cannot be determined before the open without additional operations; for example, when a VSE Disk Space Manager or Tape Manager is active, or when using VSAM/SAM sequential files. Adabas also determines the block size to be used for sequential I/O areas by device type.
2. Valid disk device types are 3380, 3390, 9345 and FBA.
3. Valid tape device types are 2400, 3410, 3420, 3480, 3490E, 3590, and 8809.

Additional Parameters Used for Internal Control Only

Three additional parameters are also available but are used only for internal control and should not be changed from their default settings unless otherwise specified by your Software AG technical support representative:

```
IORTRAC={NO | YES}
IORTSIZ={3000 | tablesize}
IORTTYP=(1... 14)(, opt1 ... opt14 ).
```

Catalog Procedures for Defining Libraries and the Database

Note: Sample JCS is available in ADAPROC.X

The job ADAPROC is divided into two procedures:

- ADAV*v*LIB defining the library or libraries; and
- ADAV*v*FIL defining the database.

Customize and catalog the two procedures before placing them back in the procedure library. The following specific items must be customized:

- file IDs for the database and libraries;
- volumes for libraries and database files;
- space allocation for database files.

The Adabas DEMO database files include ASSO, DATA, WORK, TEMP, SORT, CLOG, and PLOG.

Install a New Database

Follow the steps outlined below to install a new Adabas database under z/VSE.

Notes:

1. For information about running ADADEF ADALOD, ADAREP, and ADASAV in steps 1-3, 5, and 8 below, see the *Adabas Utilities* documentation.
2. For information about customizing the nucleus job and about starting and terminating the nucleus in steps 4 and 7 below, see the *Adabas Operations* documentation.

to install a new Adabas database

1 Allocate and format the DEMO database

Note: Sample JCS is available in ADAFRM.X

Customize and run the ADAFRM utility job to format the DEMO database areas. The following specific items must be customized:

- the Adabas SVC number, the database ID, and database device type(s);
- sizes of the datasets for each ADAFRM statement.

2 Define the global database characteristics

Note: Sample JCS is available in ADADEF.X

Customize and run the ADADEF utility job to define the global definition of the database. The following items must be customized:

- the Adabas SVC number, the database ID, and database device type(s);
- ADADEF parameters.

3 Load the demonstration (demo) files

Note: Sample JCS is available in ADALODE.X, ADALODV.X, and ADALODM.X.

Customize and run the job

- ADALODE to load the sample demo file EMPL;
- ADALODV to load the sample demo file VEHI; and
- ADALODM to load the sample demo file MISC.

For each job, the following items must be customized:

- the Adabas SVC number, the database ID, and database device type(s);
- ADALOD parameters.

4 Start the Adabas nucleus and test the Adabas communications

Note: Sample JCS is available in ADANUC.X.

Customize and run the job JCLNUC to start up the Adabas nucleus. The following items must be customized:

- the Adabas SVC number, the database ID, and device type(s);
- ADANUC parameters.

5 Test Adabas partition communications

Note: Sample JCS is available in ADAREP.X.

Customize and run the job ADAREP in MULTI mode with the `CPLIST` parameter to test Adabas partition communications. The following items must be customized:

- the Adabas SVC number, the database ID, and device type(s);
- ADAREP parameters.

6 Load the Adabas Online System, if used

Note: Sample JCS is available in NATINPL.X. Read ***Installing the AOS Demo Version***, elsewhere in this guide, and, if necessary, the installation section of the Adabas Online System documentation.

Customize and run the job NATINPL to load the Adabas Online System into a Natural system file. A Natural file must first be created, requiring an INPL input file (see the Natural installation instructions). The following items must be customized:

- the Adabas SVC number, the database ID, and device type(s);
- the Natural INPL parameters and system file number.

7 Terminate the Adabas nucleus

Communicate with the Adabas nucleus (MSG F*n*) to terminate the session by entering the Adabas operator command `ADAEND` into the Adabas nucleus partition.

8 Back up the database

Customize and run the ADASAV utility job to back up the Version sample database. The following items must be customized:

- the Adabas SVC number, the database ID, and device type(s);
- ADASAV parameters.

9 Insert the ADARUN defaults

Optionally customize and run the DEFAULTS job to set the ADARUN defaults using the MSHP utility and to relink ADARUN. The following items may be customized:

- SVC number;
- database ID;
- device type(s).

10 Install the required TP link routines for Adabas

Refer to the section ***Installing Adabas with TP Monitors*** for the TP link routine procedure.

See the following Adabas manuals for the related information:

- Adabas Utilities documentation for information on running the ADALOD, ADAFRM and ADAREP utility jobs.
- Adabas Operations documentation for information about operator commands for monitoring and controlling the nucleus.

Migrate an Existing Database

Use the ADACNV utility to migrate existing databases to new releases of Adabas. See the *Adabas Utilities* documentation for more information.

Logical Unit Requirements

This section describes the Adabas logical unit requirements.

ADARUN

Logical Unit	File	Storage Medium
SYSLST	PRINT	Printer
SYS000	CARD	Tape / Disk
SYSRDR	CARD	Reader

Utility

Logical Unit	File	Storage Medium
SYS009	DRUCK	Printer
SYSIPT	KARTE	Reader

Nucleus

Logical Unit	File	Storage Medium
SYSLST	PRINT	Printer
SYSRDR	CARD	Reader

The highest logical unit used is SYS038 for the ADASAV utility. The programmer logical units default is described in the section ***Device and File Considerations***. The system programmer should review these requirements to ensure that there are enough programmer logical units to run the desired utilities in the desired partitions.

Job Exit Utility

Adabas provides a job exit to perform two different functions:

- Librarian input override processing

  The exit scans a job stream for Librarian input override statements. These statements indicate that card input (ADARUN CARD or utility KARTE statements) for a job step is to come from Librarian members rather than from SYSRDR or SYSIPT.

- ADARAI JCS capture processing

  The exit captures JCS before it is modified by tape or disk management systems for later use by ADARAI.

You can set the job exit to perform either function or both. By default, the job exit performs Librarian input override processing.

Installation and Initialization

The job exit can be installed during ASI processing or at any time afterward. It is installed in two steps:

to install the job exit:

1. Install programs SAGJBXT and SAGIPT in the SVA.
2. Run program SAGINST to initiate job exit processing.

You can include SAGJBXT in the $JOBEXIT list of eligible exits, but you must still place SAGIPT in the SVA and run SAGINST to allocate the required table(s).

SAGIPT runs above the 16-megabyte line if an appropriate 31-bit PSIZE is available. In addition, the table that stores information from input-override statements and/or the table that stores JCS for ADARAI use is placed in 31-bit GETVIS, if available.

SAGINST reads an input parameter that tells it whether to install the Librarian input override processing, ADARAI JCS capture processing, or both. The following parameter values are valid:

```
PARM=ADALIB (the default) installs Librarian input override processing
PARM=ADARAI installs ADARAI JCS capture processing
```

The following sample job control initializes the job exit:

Note: Sample JCS to initialize the job exit is available in member JBXTINST.X.

```
* $$ JOB JNM=SAGEXIT,CLASS=0
* $$ LST CLASS=A,DISP=D
// JOB SAGEXIT
// LIBDEF *,SEARCH=SAGLIB.ADAvrs
// EXEC PROC=ADAVvLIB
SET SDL
SAGJBXT,SVA
SAGIPT,SVA
/*
// EXEC SAGINST,PARM=ADARAI,ADALIB
/&
* $$ EOJ

—where vrs is the Adabas version, revision, and system maintenance level.
```

Librarian Input Override Processing

If Librarian input override processing is specified, the job exit scans a job stream for input override statements indicating that card input (ADARUN CARD or utility KARTE statements) for a job step is to come from Librarian members rather than from SYSRDR or SYSIPT. By default, the exit can store a maximum of 2000 input override cards simultaneously throughout the system. Adabas uses this facility when processing CARD and KARTE parameters.

Enable Librarian input override processing by adding * SAGUSER control statements to the job control stream between the // JOB and // EXEC statements.

A * SAGUSER statement can have three keyword parameters: `FILE`, `LIBRARY`, and `MEMBER`.

Keyword Syntax	Description
`FILE={ CARD | KARTE }`	The file to be read from a Librarian member. Specify "CARD" for ADARUN statements, or "KARTE" for utility statements.
`LIBRARY={` *`library.sublibrary`* `|` *`libdef.source`* `}`	The library and sublibrary to be searched. If omitted, the current *`libdef.source`* chain is used.
`MEMBER=`*`name`* `[,{` *`type`* `| A } ]`	The member name and optionally the type to be read. If *`type`* is omitted, "A" is assumed.

The following is an example of a * SAGUSER control statement that specifies an alternate job exit member:

```
* SAGUSER FILE=CARD,MEMBER=NUC151
```

In the example above, Adabas searches the current *`libdef.source`* chain for member NUC151 with type A. If NUC151 is found, Adabas uses its contents as the nucleus startup parameters instead of SYSIPT.

To permit flexible startup processing, multiple SAGUSER statements may be specified for each file. In the following example, Adabas reads the input parameters first in member NUC151, then in member IGNDIB:

```
* SAGUSER FILE=CARD,MEMBER=NUC151
* SAGUSER FILE=CARD,MEMBER=IGNDIB
```

The following examples show the use of the `LIBRARY` parameter, and apply to z/VSE systems only:

```
* SAGUSER FILE=CARD,MEMBER=NUC151,LIBRARY=SAGULIB.TESTSRC
```

In the example above, Adabas searches sublibrary TESTSRC in the SAGULIB library for member NUC151 with type A. If NUC151 is not found in sublibrary TESTSRC of library SAGULIB, no further search is made. The DLBL and EXTENT information for the SAGULIB library must be available.

```
* SAGUSER FILE=CARD,MEMBER=NUC151.ADARUN,LIBRARY=SAGULIB.TESTSRC
```

In the example above, Adabas searches sublibrary TESTSRC in the SAGULIB library at nucleus initialization for member NUC151 with type ADARUN. The library member types PROC, OBJ, PHASE, and DUMP are not permitted.

Activating Adabas Use of Job Exit Processing

Specify `JOBEXIT=YES` to allow Adabas to use SAGUSER statements in the job stream and recatalog the Adabas options table (ADAOPD).

Using the Job Exit Utility for ADARAI JCS Capture

Once the job exit utility has been installed for ADARAI, all utilities that write information to the RLOG automatically obtain file information from the ADARAI table that the job exit maintains. Manual intervention is not required.

Job Exit Storage Requirements

The job exit requires from 84 to 298 kilobytes (KB) of SVA storage, depending on whether the Librarian input override interface and/or the ADARAI JCS interface is installed. Of that total,

- 2 kilobytes are used for program storage (PSIZE);
- 82-kilobyte GETVIS for Librarian input override storage; and
- 214-kilobyte GETVIS for ADARAI JCS storage.

When running in ESA or VM/ESA mode on ESA hardware, all of the GETVIS and 1 kilobyte of the PSIZE can be run above the 16-megabyte line.

Optional Console or Printer Messages

You have the option of displaying, printing, or preventing these messages by specifying the `JBXEMSG` and `JBXIMSG` parameters in the Adabas options table.

Diagnostic Functions

After the job exit is installed, you can produce dumps of the two tables for diagnostic purposes. Executing SAGINST with the ADASIP UPSI statement:

- UPSI 10000000 produces a dump of the Librarian input override table;
- UPSI 01000000 produces a dump of the ADARAI JCS table.

If the size of these two tables needs to be changed for any reason, SAGIPT may be zapped before being loaded into the SDL:

- The Librarian input override table size may be changed from the default of X‘00014874’ (84,084 bytes) to an appropriate value by zapping location X‘18’. When altering the SAGIPT.OBJ module, `ESDID=002` is required on the MSHP AFFECTS statement.
- The ADARAI JCS table size may be changed from the default of X‘000355D6’ to an appropriate value by zapping location X‘0C’.

Each element in the Librarian input override table is 42 bytes in length. The default table size assumes 10 SAGUSER statements per file name, 10 file names, and 20 partitions, plus two extra unused entries. This number is an estimate of maximum concurrent residency; each statement is removed from the table after it is used.

Each element in the ADARAI JCS table is 91 bytes in length. The default table size accommodates 2400 entries with each DLBL, TLBL, or EXTENT statement requiring an entry in the table. Whenever a JOB statement is encountered, all entries for that partition (task ID) are cleared from the table.

Acquiring Storage for the ID Table

The SYSTEM GETVIS is used to acquire storage for the ID table (IDT). This storage is acquired using the ADASIP at SVC installation time. The size of storage in the SYSTEM GETVIS depends on the number of IDT entries specified using ADASIP. The default number of IDT entries (IDTEs) is 10. The size can be calculated as follows:

```
SIZE (in bytes) =1024 (IDT prefix) + 96 (IDT header) + (32 x number of IDTEs)
= 1024 + 96 + (32 x 10)
= 1024 + 96 + 320
= 1440 bytes
```

Also, additional SYSTEM GETVIS storage is acquired. This storage permits users to communicate from multiple address spaces when Adabas is not running in a shared partition. In this case, the following formula is used to calculate SYSTEM GETVIS:

```
SIZE (in bytes) = 192 (CQ header) + (192 x NC value) + (4352 x NAB value)
```

It may be necessary to increase the SVA size to meet these requirements. To do so, change the SVA operand in the appropriate $IPL*xxx* procedure, then re-IPL.

Note: By default, the SYSTEM GETVIS is acquired above the 16-megabyte line. To acquire most of this space below the line, linkedit ADARUN AMODE 24.

Acquiring Storage for the IIBS Table

The 31-bit SYSTEM GETVIS is used to acquire storage for the IIBS table (IIBS). This storage is acquired using the ADASIP at SVC installation time. The size of storage in the 31-bit SYSTEM GETVIS is 128K.

Displaying Storage Allocation Totals

Specifying // UPSIG during the ADASIP execution will generate allocation messages on the system console, showing the total 24-bit GETVIS and 31-bit GETVIS storage allocated by Adabas:

```
ADASIP85 GETVIS-24 storage allocated: nnnK
ADASIP85 GETVIS-31 storage allocated: nnnK
```

Calls from Other Virtual Address Spaces

In the non-shared mode of VSE, you may select whether this nucleus will accept calls from another virtual address space. The default for a nucleus running in a non-shared partition causes Adabas to accept calls from partitions in other address spaces, and to acquire storage in the SVA GETVIS area for any required attached buffers. The buffers hold data moved between the nucleus and users in other partitions in other address spaces.

However, if you want the nucleus to accept calls from other partitions within the same address space but not from other address spaces, you must specify the following statement in the Adabas startup JCS:

```
// OPTION SYSPARM='NOSVA'
```

When NOSVA is specified, storage is acquired within the Adabas nucleus partition and not in the SVA GETVIS area (except for ID table storage, which is still taken from the SVA). Calls from users in other partitions in other address spaces will receive an Adabas response code 148, “nucleus not available”.

Dummy Sequential Files

If the file is not needed, it can be unassigned or assigned IGN such as the following:

```
// ASSGN SYS014,UA
—or
// ASSGN SYS014,IGN
```

Backward Processing of Tapes and Cartridges

To perform backward processing of tapes or cartridges, file positioning must occur before the file is opened. This can only be done when an assignment is made for the file. When performing the ADARES BACKOUT utility function, the // ASSGN ... for file BACK must be done explicitly.

No tape management system can be used, because such systems perform the assign operation when the file is opened; the LUB and PUB remain unassigned until this occurs.

Applying Zaps (Fixes)

The jobs described in this section can be used to permanently change defaults and apply corrections (zaps) to the libraries in the supported z/VSE systems.

Two methods are used in z/VSE for applying corrective fixes to Adabas:

- the MSHP PATCH facility requires no definition of Adabas as a product/component on the MSHP history file. This method only alters phases. If the phase is relinked, the zap is lost.
- the MSHP CORRECT facility requires the definition of Adabas as a product/component using MSHP ARCHIVE.

Software AG distributes Adabas zaps to z/VSE users in MSHP CORRECT format and therefore recommends that you use MSHP CORRECT.

- Link Book Update Requirements for Running AMODE 24

Applying Fixes Using MSHP PATCH

A sample job for applying a fix to Adabas using MSHP PATCH is as follows:

Note: This sample job is available in member MSHPPAT.X.

```
// JOB PATCH APPLY PATCH TO ADABAS
// OPTION LOG
// EXEC PROC=ADAVvLIB
// EXEC MSHP
PATCH SUBLIB=saglib.ADAvrs
AFFECTS PHASE=phasenam
ALTER offset vvvv : rrrr
/*
/&

—where
vrs is the Adabas version/revision/system maintenance (SM) level
saglib is the Adabas library name in procedure ADAVvFIL
phasenam is the Adabas phase to be zapped
offset is the hexadecimal offset into the phase
vvvv is the verify data for the zap
rrrr is the replace data for the zap
```

Applying Fixes Using MSHP CORRECT

- MSHP ARCHIVE
- MSHP CORRECT

MSHP ARCHIVE

For new users or users with no requirement to maintain multiple versions of Adabas, the following sample job can be used to define Adabas to MSHP.

Note: This job uses the history file identified by the IJSYSHF label in the VSE standard label area.

Note: This sample JCL is available in member MSHPARC.X.

```
// JOB ARCHIVE ARCHIVE ADABAS
// OPTION LOG
// EXEC PROC=ADAVvLIB
// EXEC MSHP
ARCHIVE ADAvrs
COMPRISES 9001-ADA-00
RESOLVES 'SOFTWARE AG - ADABAS Vv.r'
ARCHIVE 9001-ADA-00-vrs
RESIDENCE PRODUCT=ADAvrs -
PRODUCTION=saglib.ADAvrs -
GENERATION=saglib.ADAvrs
/*
/&

—where
vrs is the Adabas version/revision/system maintenance (SM) level
saglib is the Adabas library name in procedure ADAVvFIL
```

Starting with Version 7.1, a different MSHP history file must be used for each version and revision level of Adabas to which maintenance is applied.

To preserve the MSHP environment of an older version level of Adabas during an upgrade to a new version, it is necessary to create an additional MSHP history file for use by the new version.

The following sample MSHP job can be used to create an additional history file for a new version of Adabas and define Adabas to it.

Note: This sample JCL is available in member MSHPDEF.X.

```
// JOB ARCHIVE DEFINE HISTORY AND ARCHIVE ADABAS
// OPTION LOG
// EXEC PROC=ADAVvLIB
// ASSGN SYS020,DISK,VOL=volhis,SHR
// EXEC MSHP
CREATE HISTORY SYSTEM
DEFINE HISTORY SYSTEM EXTENT=start:numtrks -
UNIT=SYS020 -
ID='adabas.new.version.history.file'
ARCHIVE ADAvrs
COMPRISES 9001-ADA-00
RESOLVES 'SOFTWARE AG - ADABAS Vv.r'
ARCHIVE 9001-ADA-00-vrs
RESIDENCE PRODUCT=ADAvrs -
PRODUCTION=saglib.ADAvrs -
GENERATION=saglib.ADAvrs
/*
/&

—where
vrs is the Adabas version/revision/system maintenance (SM) level
```

```
volhis is the volume on which the Adabas Vvr history file resides
start is the start of the extent on which the Adabas Vvr history file resides
numtrks is the length of the extent on which the Adabas Vvr history file resides
adabas.new.version.history.file is the physical name of the Adabas Vvr history file
saglib is the Adabas library name in procedure ADAVvFIL
```

Once migration to the new version is complete, you can either

- continue to use the new history file to apply subsequent fixes; or
- delete the old version of Adabas from MSHP and merge the new version into the standard MSHP history file.

Caution: Before running any MSHP REMOVE or MERGE jobs, back up your MSHP environment by running MSHP BACKUP HISTORY jobs against all MSHP history files.

A sample MSHP job to remove an old version of Adabas is provided below.

Note: This sample JCL is available in member MSHPREM.X.

```
// JOB REMOVE REMOVE OLD ADABAS
// OPTION LOG
// PAUSE ENSURE MSHP HISTORY FILE BACKUP HAS BEEN TAKEN
// EXEC MSHP
REMOVE ADAvrs
REMOVE 9001-ADA-00-vrs
/*
/&

—where vrs is the old Adabas version/revision/system maintenance (SM) level.
```

A sample MSHP job to merge an additional history file for Adabas into the standard MSHP history file is provided below.

Note: This sample JCL is available in member MSHPMER.X.

```
// JOB MERGE MERGE SEPARATE ADABAS INTO STANDARD HISTORY
// OPTION LOG
// PAUSE ENSURE MSHP HISTORY FILE BACKUPS HAVE BEEN TAKEN
// ASSGN SYS020,DISK,VOL=volhis,SHR
// EXEC MSHP
MERGE HISTORY AUX SYSTEM
DEFINE HISTORY AUX EXTENT=start:numtrks -
UNIT=SYS020 -
ID='adabas.new.version.history.file'
/*
/&

—where
volhis is the volume on which the Adabas Vvr history file resides
```

```
start is the start of the extent on which the Adabas Vvr history file resides
numtrks is the length of the extent on which the Adabas Vvr history file resides
adabas.new.version.history.file is the physical name of the Adabas Vvr history file
```

MSHP CORRECT

The MSHP CORRECT and UNDO jobs use the history file identified by label IJSYSHF in the VSE standard label area. If Adabas is maintained from a different MSHP history file, include the following label information in the CORRECT or UNDO job:

```
// DLBL IJSYSHF,'adabas.new.version.history.file'
// EXTENT SYSnnn
// ASSGN SYSnnn,DISK,VOL=volhis,SHR

—where
volhis is the volume on which the Adabas Vvr history file resides
nnn is the user-defined SYS number
adabas.new.version.history.file is the physical name of the Adabas Vvr history file
```

A sample of the use of MSHP CORRECT to install a fix to Adabas is provided below.

Note: This sample JCL is available in member MSHPCOR.X.

```
// JOB CORRECT APPLY ADABAS FIX
// OPTION LOG
// EXEC PROC=ADAVvLIB
// EXEC MSHP
CORRECT 9001-ADA-00-vrs : Axnnnnn
AFFECTS MODE=modname
ALTER offset vvvv : rrrr
INVOLVES LINK=lnkname
/*
/&

—where
vrs is the Adabas version/revision/system maintenance (SM) level
x is the Adabas component (for example, N for nucleus)
nnnnn is the Adabas fix number
modname is the Adabas object module to be zapped and then relinked
offset is the hexadecimal offset to the beginning of the zap
vvvv is the verify data for the zap
rrrr is the replace data for the zap
lnkname is the link book for the phase affected
```

The CORRECT job updates object and phase in a single job step using the link book feature of MSHP. The INVOLVES LINK= statement automatically invokes the linkage editor after the object module is updated.

For a zap applied with the INVOLVES LINK= statement, the following UNDO can be used to remove the fix from both object module and phase:

Note: This sample JCL is available in member MSHPUND.X.

```
// EXEC MSHP
UNDO 9001-ADA-00-vrs : Axnnnnn
/*

—where
vrs is the Adabas version/revision/system maintenance (SM) level
x is the Adabas component (for example, N for nucleus)
nnnnn is the Adabas fix number
```

Adabas provides a link book containing parameters for invoking the linkage editor for each Adabas phase. The name of each link book begins with LNK-/ESA and includes the type OBJ in the following format:

```
LNKaaaaa.OBJ
—where aaaaa (3 to 5 characters) is underlined in the following list for each Adabas
phase:
```

Note: Link books are not provided for ADAOPD or for TP monitor links. These programs are distributed in source form and continue to be modified using assembly and link jobs.

```
$$BADAS5 ADALCO ADAMPM ADAORD ADASVCMP
ADAACK ADALNK ADAMSG ADAPCS ADATRA
ADANCHOR ADALNKR ADAMXA ADAPLP ADATRC
ADABSP ADALOD ADAMXB ADAPMT ADATRI
ADACCS ADALOE ADAMXD ADAPOB ADATSP
ADACDC ADALOF ADAMXF ADAPOK ADAULD
ADACLX ADALOG ADAMXH ADAPOP ADAUSER
ADACLU ADAMER ADAMXI ADAPOV ADAVAL
ADACMO ADAMGA ADAMXL ADAPRF ADAZAP
ADACMP ADAMGB ADAMXO ADAPRI AFPADA
ADACMR ADAMGC ADAMXR ADAPSM AOSASM
ADACMU ADAMGD ADAMXT ADAPST AVIADA
ADACNS ADAMGE ADAMXU ADARAC AVILOOK
ADACNV ADAMGI ADAMXY ADARAG NOAUTOR (NAU)
ADACOM ADAMGM ADAMXZ ADARAI NOOPRSP (NOP)
ADACON ADAMGN ADANCA ADAREP PINRSP (PNR)
ADACOT ADAMGR ADANCB ADARES PINUES (PNU)
ADACVC ADAMGO ADANCC ADAREX PRILOG (PRL)
ADADBS ADAMG1 ADANCX ADARMT SAGINST
ADADCK ADAMG2 ADANCO ADARST SAGIPT
ADADEC ADAMG3 ADANC1 ADARUN SAGJBXT
ADADEF ADAMG4 ADANC2 ADARVU STPEND
ADADSP ADAMG5 ADANC3 ADASAV STPNAT
ADAECS ADAMG6 ADANC4 ADASCR STPUES
```

```
ADAFDP ADAMG7 ADANC5 ADASEL USEREX2
ADAFRM ADAMG8 ADANC6 ADASIP WTOVSE
ADAICK ADAMG9 ADANC7 ADASQD
ADAINV ADAMIM ADANC8 ADASQR
ADAIOR ADAMLF ADANC9 ADASTUB
ADAIOS ADAMOD ADAOPD ADASVC74
```

If you choose not to take advantage of the link book facility, remove the INVOLVES LINK= statement from any zap before applying it. You can then run the linkage editor step to recreate the phase separately, as before.

This may be done to link a temporary version of a phase into a separate sublibrary for testing purposes. However, it is also possible to maintain a separate test version of Adabas modules by defining an additional VSE system history file. See ***Maintaining a Separate Test Environment in VSE***.

Link Book Update Requirements for Secondary SVC

If you use the link book facility and require a non-standard SVC suffix (for example, if you relink the Adabas 7.4 SVC to phase ADASVC11), you must remember to update the link book for the SVC (LNKSVC.OBJ) to reflect the new phase name.

The link book provided for ADASVC74 is LNKSVC.OBJ. It contains the following:

```
PHASE ADASVC74,*,NOAUTO,SVA
MODE AMODE(31),RMODE(24)
INCLUDE SVCVSE
INCLUDE LCTVSE
ENTRY ADASVC
```

To set up an SVC with suffix -11, you would need to update the link book as follows:

```
// DLBL SAGLIB,'adabas.Vvrs.library'
// EXTENT SYS010
// ASSGN SYS010,DISK,VOL=volser,SHR
// EXEC LIBR
ACCESS SUBLIB=SAGLIB.ADAvrs
CATALOG LNKSVC.OBJ REPLACE=YES
PHASE ADASVC11,*,NOAUTO,SVA
MODE AMODE(31),RMODE(24)
INCLUDE SVCVSE
INCLUDE LCTVSE
ENTRY ADASVC
/+
/*

—where
vrs is the Adabas version/revision/system maintenance (SM) level
```

```
adabas.Vvrs.library is the physical name of the Adabas vrs library
volser is the volume on which the library resides
```

Link Book Update Requirements for Running AMODE 24

If you use the link book facility and require AMODE 24 versions of any modules linked by default as AMODE 31 (ADARUN, ADASVC74), you must update the corresponding link book (LNKRUN.OBJ, LNKSVC.OBJ) to remove the MODE statement.

This link book update can be made using a method similar to that described in the previous section for the SVC suffix update.

Optional z/VSE User Zaps

The following table describes the changes (zaps) currently available for Adabas:

Zap	Module	Offset	VER	REP	Description
1	ADAUSER	0102	4700	47F0	Use VSE LOAD (SVC 4) instead of CDLOAD (SVC 65) when loading ADARUN (see note 1).
2	WTOVSE	0104	4700	47F0	Suppress DUMP (JDUMP) when an error occurs in loading ADARUN.
3	RUNVSE	10FC	4700	47F0	Use VSE LOAD (SVC 4) instead of CDLOAD (SVC 65) when loading Adabas modules.
4	LNKVSE	01F2	4700	47F0	Suppress DUMP (JDUMP) when an error occurs in ADALNK.
5	SIPVSE	0030	0A00	0AXX	Insert the default ADASIP SVC. XX is the hexadecimal SVC number.

Notes:

1. When executing in `AMODE=31` and `RMODE=ANY`, CDLOAD must be used; this zap cannot be installed.
2. Optional zaps for ADAUSER, WTOVSE, and RUNVSE require `ESDID=002` on the respective MSHP AFFECTS statements.

Adalink Considerations

- User Exit B (Pre-Command) and User Exit A (Post-Command)
- LNKUES for Data Conversion

- ADAUSER Considerations

User Exit B (Pre-Command) and User Exit A (Post-Command)

One or two user exits may be linked with an Adalink routine:

- UEXITB receives control *before* a command is passed to a target with the router 04 call.

  Note: Special commands emanating from utilities and from Adabas Online System are marked as physical calls. These calls must be bypassed in user exits. These calls have X‘04’ in the first byte (TYPE field) of the command’s Adabas control block (ACB). UEXITB must check this byte and return if it is set to X‘04’. Be sure to reset R15 to zero on return.

- UEXITA receives control *after* a command has been completely processed by a target, the router, or by the Adalink itself.

At entry to the exit(s), the registers contain the following:

Register	Contents
1	Address of the UB. If the flag bit UBFINUB is reset, the contents of the halfword at Adabas + X’86’ have been moved to UBLUINFO. If those contents are greater than zero, the two bytes starting at UBINFO (UB+X’40’) have been set to zero. If UBFINUB is set, no changes can be made to the UB or ACB (except for ACBRSP).
2	Address of a 16-word save area (for ADALNC only)
13	Address of an 18-word save area (for non-CICS Adalink exits)
14	Return address
15	Entry point address: UEXITB or UEXITA

Any registers except register 15 that are modified by the user exits must be saved and restored; the address of a save area for this purpose is in register 13.

If at return from UEXITB register 15 contains a value other than zero (0), the command is not sent to the target but is returned to the caller. The user exit should have set ACBRSP to a non-zero value to indicate to the calling program that it has suppressed the command: response code 216 is reserved for this purpose.

The UEXITB exit may set the UB field UBLUINFO to any lesser value, including zero; an abend occurs if the user exit sets UBLUINFO to a greater value. The UBLUINFO length cannot be changed when any other exit is used.

The user information received by a UEXITA exit may have been modified; this modification may include decreasing its length, possibly to zero, by any of the Adalink user exits.

An Adalink routine can return the following non-zero response codes in ACBRSP:

Response Code	Description
213	No ID table
216	UEXITB suppressed the command
218	No UB available

At least the following three equates, described at the beginning of the source, can be modified before an Adalink routine is assembled. In some Adalink routines, however, the corresponding information can be zapped:

Equate	Description
LOGID	The default logical ID, ranging in value from 1 to 65535. The default is 1.
LNUINFO	The length of the user information to be passed to Adalink user exits, ranging in value from 0 to 32767. The default is 0.
SVCNR	The Adabas SVC number; its range of values and the default depend on the operating system. This value can be provided as SYSPARM value for assembly of the following Adalink routine: `//EXEC PGM=ass,PARM=‘......,SYSPARM(svcnr)’`

The first 152 (X’98’) bytes of all Adabas Adalinks must maintain the following structure:

Offset	Label	Contents	Meaning
00	ADABAS		Entry code
12		CL6’ADALN*x*’	Program name
18		XL4’*yyyymmdd*’	Assembly date
1C		A(ZAPTAB)	Address of zap table
20	PATCH	XL96’00’	Patch area
80	LNKLOGID	AL2(LOGID)	Default logical ID (default: 1)
82		82 XL2’00’	Reserved
84	LNKSVC	SVC SVCNR	Executable SVC instruction for Adabas SVC (default: operating-system-dependent)
86	LUINFO	Y(LNUINFO)	Length of user information (default: 0)
88	VUEXITA	V(UEXITA)	Address of user exit after call (weak)
8C	VUEXITB	V(UEXITB)	Address of user exit before call (weak)
90	ADABAS51	CL8’ADABAS51’	IDT ID

LNKUES for Data Conversion

The Adabas Version 7 standard batch ADALNK is delivered with UES (Universal Encoding Support). The LNKUES module, as well as the modules ASC2EBC and EBC2ASC, are linked into the standard batch ADALNK. LNKUES converts data in the Adabas buffers and byte-swaps, if necessary, depending on the data architecture of the caller.

Prior to Version 7, Entire Net-Work converted all data for mainframe Adabas. When Entire Net-Work Version 5.5 and above detects that it is connected to a target database that converts data, it passes the data through without converting it.

LNKUES is called only on ADALNK request (X’1C’) and reply (X’20’) calls if the first byte of the communication ID contains X’01’ and the second byte does not have the EBCDIC (X’04’) bit set.

- For requests, LNKUES receives control before UEXITB.
- For replies, LNKUES receives control after UEXITA.

By default, two translation tables are linked into LNKUES/ADALNK:

- ASC2EBC: ASCII to EBCDIC translation; and
- EBC2ASC: EBCDIC to ASCII translation.

Note: It should only be necessary to modify these translation tables in the rare case that some country-specific character other than "A-Z a-z 0-9" must be used in the Additions 1 (user ID) or Additions 3 field of the control block.

If you prefer to use the same translation tables that are used in Entire Net-Work:

- in ASC2EBC and EBC2ASC, change the COPY statements from UES2ASC and UES2EBC to NW2ASC and NW2EBC, respectively.
- re-assemble the translation tables and re-link LNKUES/ADALNK.

Both the Adabas and Entire Net-Work translation table pairs are provided in the section ***Translation Tables***. You may want to modify the translation tables or create your own translation table pair. Be sure to (re)assemble the translation tables and (re)link LNKUES/ADALNK.

The following is a sample job for (re)linking ADALNK with LNKUES and the translation tables:

```
*
// JOB ...
// EXEC PROC=
// LIBDEF *,SEARCH=(search-chain-library.sublib ...)
// LIBDEF PHASE,CATALOG=(lib.sublib)
PHASE ADABAS,*
MODE AMODE(31),RMODE(24)
INCLUDE ADALNK
INCLUDE LNKUES
INCLUDE ASC2EBC
INCLUDE EBC2ASC
ENTRY ADABAS
// EXEC LNKEDT
```

The (re)linked ADALNK must be made available to Entire Net-Work. If you are calling Adabas version 7 and you do not have the correct LNKUES/ADALNK module, Adabas produces unexpected results: response code 022, 253, etc.

ADAUSER Considerations

ADAUSER is a program that links the user to Adabas. It is specific to an operating system and is independent of release level and mode. It can be used in batch and in some TP environments.

ADAUSER contains the entry point ADABAS and should be linked with all user programs that call Adabas. No other programs containing the CSECT or entry point name ADABAS can be linked in these load phases.

On the first Adabas call, ADAUSER (CDLOAD) loads the latest version of ADARUN. This makes the calling process release-independent. Subsequent Adabas calls bypass ADARUN.

ADARUN processes its control statements. For the ADARUN setting `PROGRAM=USER` (the default), ADARUN loads the appropriate TP Adalink if the ADARUN parameter setting `MODE=MULTI` is specified, or the Adabas nucleus (ADANUC) if the ADARUN parameter setting `MODE=SINGLE` is specified. This makes the calling process mode independent.

Setting Defaults in ADARUN

The member DEFAULTS.X is available for setting the ADARUN defaults.

DEFAULTS.X uses MSHP CORRECT to install the fix.

Default Name	Current Value
Device type	3380
SVC number	45
Database ID	1

5 Installing Adabas With TP Monitors

This section provides information needed to install Adabas in batch mode and with teleprocessing (TP) monitors Com-plete, CICS and Shadow. Information about using Adabas with TP monitors is also contained in the section describing Adabas installation.

TP Monitor	TP Monitor / Adalink
All environments	All
CICS command-level	LNKOLSC / LNKOLM
CICS high-performance stub	LNCSTUB
Com-plete	ADALCO
Shadow	ADALNS
Batch	ADALNK, ADALNKR(LNKVSER)

Preparing Adabas Link Routines for z/VSE

High-Level Assembler

To maintain compatibility with year 2000 date format requirements, use the IBM high-level assembler when assembling the Adabas link routines for TP monitors on z/VSE. The high-level assembler generates 4-digit year assembly dates into the load modules using the &SYSDATC assembly variable.

It is possible to assemble the link routines with the old VSE assembler after editing the source members and making the following changes:

- For all link members

▶

1 Remove the keyword `HLASM=YES` from the ASMDATE macro in the source.

2 Supply the following statement on the assembly step to provide the assembly date:

```
// OPTION SYSPARM='yyyymmdd'
—where
yyyy is the 4-digit year
mm is the 2-digit month
dd is the 2-digit day
```

Notes:

1. The ASMDATE macro can also accept a 2-digit year from SYSPARM (// OPTION SYS-PARM='yymmdd')
2. The ASMDATE.E book must be available in the LIBDEF search chain to assemble the link routines with the old VSE assembler.

3 Remove or comment out the AMODE and RMODE directives in the source members.

- For the ADALCO (Com-plete) link routine

  Ensure that the BASSM.E and BSM.E books are in the LIBDEF search chain of the old VSE assembler job step.

Addressing Mode Assembly Directives

The Adabas link routines now have `AMODE` and `RMODE` assembly directives in the source. These allow the linkage editor to produce warning messages when conflicting `AMODE` or `RMODE` linkage editor control statements are encountered in the link JCS or EXECs.

These assembly directives also serve to document the preferred `AMODE` and `RMODE` for each link routine. It is important to note that in and of themselves, these directives do not alter the actual addressing mode of the link routine during execution.

The batch link routine ADALNK has the following `AMODE` and `RMODE` assembly directives:

```
ADABAS AMODE 31
ADABAS RMODE 24
```

Software AG recommends `RMODE 24` for the z/VSE batch link routine.

For the CICS command-level link routines (modules LNKOLM, LNKOLSC), the directives are

```
ADABAS AMODE 31
ADABAS RMODE ANY
```

Modifying the Assembly Directives

These directives may be changed by modifying the source members before assembling them, or they may be overridden by linkage editor control statements. For example, to link the batch/TSO ADALNK module with `AMODE 31` and an `RMODE ANY`, the following control statements may be provided as input to the linkage editor:

```
MODE AMODE(31),RMODE(ANY)
ENTRY ADABAS
```

The linkage editor control statements override the Assembler directives in the source module.

For more information about the `AMODE` and `RMODE` directives and their effects on the assembler, linkage editor, and execution, consult the *IBM MVS/ESA Extended Addressability* Guide.

UES-Enabled Link Routines

For Adabas Version 7.4, UES is enabled by default for the batch/TSO, Com-plete, and IMS link routines. It is not necessary to disable UES support. Applications that do not require UES translation continue to work properly even when the UES components are linked with the Adabas link routines. See the section ***Connecting UES-Enabled Databases*** for more information.

Disabling UES Support

However, if for some reason you feel it necessary to disable UES support in the Adabas link routines, use the following procedure to do so:

1. Edit the source member ADALCO, ADALNI, ADALNK, or ADALNKR. Set the &UES Boolean assembler variable to 0 by commenting out the source line where it is set to 1 and removing the comment from the line where it is set to 0.
2. Assemble the link routine after making any other necessary modifications to the equates and other directives in the source module as required by your installation.
3. Link the Adabas link routine and do not include any of the UES components (that is, LNKUES, ASC2EBC, or EBC2ASC).

Installing Adabas with CICS

The Adabas command-level link routine supports the CICS transaction server (TS) 1.1 running under z/VSE 2.4 and above.

How Adabas is installed on CICS-based systems depends on the level of CICS being run:

- The command-level link can be used with VSE CICS/VS 2.2 and 2.3.
- CICS TS 1.1 running under z/VSE 2.4 and above must run a current version of Adabas and use the command-level link component.

Note: The OPID option for the USERID field is not supported under CICS/VS 2.3 and above; therefore, it is not provided with the command-level link routine.

The following sections describes specific points of Adabas/CICS installation and operation from the CICS perspective, depending on the CICS level being used.

CICS MRO Environment Requirements

If you run the Adabas CICS command-level link routine with the CICS multiple region option (MRO), you must set the ADAGSET option `MRO=YES` and use the default value for the ADAGSET `NETOPT` option.

You can use the ADAGSET `NTGPID` option to provide a 4-byte literal for the Adabas communication ID to be used by the Adabas SVC when applications that call Adabas span multiple application regions.

Alternatively, you can create a user exit B (UEXITB) for the link routine that

- sets UBFLAG1 (byte X'29' in the UB DSECT) to a value of X'08' (UBF1IMSR); and
- places a 4-byte alphanumeric value in the UB field UBIMSID.

The exit then allows the Adabas SVC to provide a proper Adabas communication ID in the Adabas command queue element (CQE) even when transactions originate in multiple regions.

Standard Versus Enhanced Installation

All supported versions of the command-level link routine can be installed using the standard installation, which comprises steps 1 through 3 of the **installation procedure**.

Steps 1 through 5 are required in order to use the enhanced features of the command-level link routine:

- CICS transaction isolation;
- a fully reentrant command-level link.

Step 6 is used to install the optional DISPGWA program. The DISPGWA program is only available with the enhanced installation.

The enhanced installation is required if

- CICS transaction isolation is used;
- the DISPGWA storage display program is used.

CICS Transaction Isolation

The enhanced Adabas CICS command-level link components take advantage of the transaction isolation facility provided by the CICS transaction server (TS) Version 1.1 when running with specific hardware under z/VSE 2.4 or above.

Transaction isolation is an extension of the storage protection mechanism, which permits resources to access either CICS or user storage by using storage protection keys. Resources defined to operate in

- user key may not overwrite CICS storage, thus affording a degree of protection to CICS.
- CICS key may read or write either CICS or user key storage, affording the highest degree of access to CICS resources.

Using the transaction isolation facility of CICS TS 1.1 under z/VSE 2.4 or above, CICS resources are further protected by isolating them in *subspaces*. This protects user key resources from one another, and protects CICS key resources from the CICS kernel.

Transaction isolation can be enabled globally through the TRANISO system initialization (`SIT`) parameter, and for each CICS transaction with the new resource definition `ISOLATE` keyword.

Transaction isolation places some restrictions on CICS resources that must be available both during the life of the CICS system and to all transactions running in the CICS system.

The Adabas CICS command-level link components must be defined to CICS with the proper storage access to ensure proper operation when transaction isolation is active:

- The Adabas CICS command-level link routine, comprising the LNKOLSC, LNKOLM, and CICS entry and exit code, must be defined to CICS as a *user key* program.
- The LNKTRUE and LNKENAB (ADATRUE and ADAENAB) programs must both be defined as *CICS key* programs.

This permits the correct degree of isolation between applications invoking the link routine, and the Adabas CICS task-related user exit (TRUE), which interacts directly with CICS resources.

When CICS transaction isolation is active, user SVCs cannot execute from the CICS region. SVCs can be executed in CICS key, however, during the PLT phase of the CICS startup and termination. For this reason, the module LNKENAB is provided to execute during the PLTPI phase of CICS initialization.

Fully Reentrant Operation

The Adabas command-level link routine can now be made fully reentrant and not self-modifying when the module LNKENAB is executed. During execution, a CICS global work area (GWA) is obtained and passed to the command-level link, which uses the GWA to store initialization addresses. This feature is available for CICS TS 1.1 under z/VSE 2.4 and above.

DISPGWA Module : Displaying the CICS Global Work Areas

The DISPGWA module is a program provided by Software AG as an optional feature for CICS TS 1.1 under z/VSE Version 2.4 and above.

The DISPGWA program displays the global work area (GWA) used by the various command-level link components. It can be used to display other areas of CICS that are important to the Adabas command-level link routine when it is executing as a task-related user exit (TRUE). With the help of Software AG personnel, you can use this program to interrogate important data areas during problem determination.

The DISPGWA module is used only if the LNKTRUE module is used, since it is the LNKTRUE module that actually EXTRACTs the global work area.

LNKENAB and LNKTRUE Modules

This section describes the usage of the LNKENAB and LNKTRUE modules.

LNKENAB Module

The LNKENAB module

- starts and enables the task-related user exit LNKTRUE (see the next section) during PLTPI processing.
- defines the length of storage that CICS gives to LNKTRUE as each task is invoked for the first time. The storage remains in CICS until the task terminates and is used by LNKTRUE as a task work area.
- issues an Adabas command to the default target and Adabas SVC defined in the ADAGSET macro in LNKOLSC. The target need not be active.

The purpose of the command is to derive the address of the IDTH from the first SVC call. All other SVC calls from the command-level link routine are then made using a branch entry into the Adabas SVC.

LNKTRUE Module

When started and enabled by LNKENAB during PLTPI processing, the task-related user exit LNKTRUE module

- permits the command-level link routine to obtain the pointer to the access control environment element (ACEE) in a CICS/ESA 4.1 environment;
- facilitates processing when CICS transaction isolation is installed and enabled; and
- coordinates Adabas transactions through the CICS Resource Manager Interface (RMI) when the Adabas Transaction Manager (ATM) is installed and enabled.

Most of the command-level link components execute under the umbrella of LNKTRUE. This means that any abend condition during the execution of LNKTRUE is serious; CICS may even respond by terminating the entire CICS region.

Items that may cause this condition include, but are not limited to

- invalid application parameter lists for Adabas calls;
- inconsistent or incorrect keyword values coded in the ADAGSET macro for the various command-level components;
- user modification to the command-level link components or the task-related user exit; or
- incorrect coding in UEXITA and/or UEXITB components.

Software AG strongly recommends that you test application programs in a CICS region that supports the non-task-related user exit version (standard installation) of the command-level link before migrating to a task-related user exit version (enhanced installation) of the command-level link routine.

JCL and Source Members

The JCL members use the sources indicated in the following table:

JCL	Source	Source Description
CICSASMC	LNKOLSC/LNKOLM	LNKOLSC is the dependent part of the Adabas command-level link routine. LNKOLM is the independent part of the Adabas command-level link routine.
CICSASMD	DISPGWA	Display program for Adabas global work area (GWA).
CICSASME	LNKENAB	Adabas PLT-enable program.
CICSASMT	LNKTRUE	Adabas task-related user exit.

Sample Resource Definitions

Under CICS/TS 1.1 and above for z/OS and VSE, the preferred method for defining and installing CICS programs and transactions is RDO (resource definition online). The CICS documentation no longer recommends the assembly of PPT and PCT entries to define resources.

The following table provides sample RDO definitions for the Adabas CICS command-level link components. The data has been extracted directly from the CICS CSD file and should be used as a guide for providing comparable information on the CEDA panels.

```
**********************************************************************
* Sample DEFINE control statements for the DFHCSDUP utility.
* For Adabas V7.4 CICS command-level link routine components.
*
* These control statements can be used as input to the DFHCSDUP
* CICS CSD update utility to define the Adabas CICS command-level
* link routine components on a CICS/TS system.
**********************************************************************
DEFINE PROGRAM(ADABAS) GROUP(ADABAS)
DESCRIPTION(ADABAS V74s COMMAND LEVEL LINK ROUTINE)
LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(YES) USAGE(NORMAL)
USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
EXECKEY(CICS) EXECUTIONSET(FULLAPI)

DEFINE PROGRAM(ADAENAB) GROUP(ADABAS)
DESCRIPTION(ADABAS V74s PLTPI ENABLE ADATRUE PROGRAM)
LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
EXECKEY(CICS) EXECUTIONSET(FULLAPI)

DEFINE PROGRAM(ADATEST) GROUP(ADABAS)
DESCRIPTION(ADABAS V74s DISPLAY GWA PROGRAM - DISPGWA)
LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
ELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
EXECKEY(CICS) EXECUTIONSET(FULLAPI)
```

```
DEFINE PROGRAM(ADATRUE) GROUP(ADABAS)
DESCRIPTION(ADABAS V74s TASK RELATED USER EXIT)
LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(YES) USAGE(NORMAL)
USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
EXECKEY(CICS) EXECUTIONSET(FULLAPI)

DEFINE TRANSACTION(DGWA) GROUP(ADABAS)
DESCRIPTION(TRANSACTION TO DISPLAY ADABAS GWA)
PROGRAM(ADATEST) TWASIZE(128) PROFILE(DFHCICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(CICS) STORAGECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO) INDOUBT(BACKOUT)
RESTART(NO) SPURGE(NO) TPURGE(NO) DUMP(YES) TRACE(YES)
RESSEC(NO) CMDSEC(NO)
```

—where *s* is the system maintenance level of Adabas.

These sample DEFINE statements are located in member DEFADAC in the Adabas Version 7.4 CICS command-level source library. They can be modified and used as input to the IBM DFHCSDUP utility to define the Adabas CICS command-level components. Consult the appropriate IBM CICS documentation for information on the DFHCSDUP utility.

Modifying Source Member Defaults (ADAGSET Macro)

The ADAGSET macro is used to create default settings for the command-level link components. This macro exists in each of the installation source members. The macro settings must be identical in every installation source member used.

To facilitate the assembly of the Adabas command-level link components, Software AG recommends that you modify the ADAGSET macro with site-specific keyword values and place the updated copy in a library that is available in the LIBDEF source search chain for the assembly job step.

If more than one version of the Adabas command-level link routine components is to be used in a given CICS system, modify a separate ADAGSET macro for each version and place these macros in unique sublibraries. Then reference the appropriate sublibrary in the LIBDEF search chain in the assembly JCS.

Note: Beginning with Adabas version 6.2.3, the source members no longer contain any keyword values on the ADAGSET statement. Thus, the macro itself must be modified to provide site-specific defaults.

It is critical that the values for the following keywords agree for all components of the Adabas CICS command-level link routine: LOGID, SVCNO, LUINFO, LRINFO, LUSAVE, NUBS, ENTPT, TRUENAM, and ENABNAM.

Step 1 of the installation procedure identifies the source members that must be edited for standard and enhanced installation.

The ADAGSET parameter options with their default values (underlined) are described below:

AVB: Adabas VSAM Bridge Support

Parameter	Description
`AVB={ NO | YES }`	Indicates whether or not Software AG's Adabas Bridge for VSAM is to be supported by this command-level link routine. ■ `AVB=YES`: Adabas VSAM Bridge is to be supported. ■ `AVB=NO`: Adabas VSAM Bridge is not to be supported.

ENABNM: Entry Point Name for Program to Enable Adabas TRUE

Parameter	Description
`ENABNM={ 'ADAENAB' | 'name' }`	The entry point name for the program that is run to enable the Adabas TRUE during CICS PLTPI processing. The value must be a valid program name that matches the module name specified in the DFHPLT table at your site. This parameter is ignored if `TRUE=NO` is specified.

ENTPT: Name of the Adabas CICS Command-Level Link Routine

Parameter	Description
`ENTPT={ 'ADABAS' | 'name' }`	The name given to the Adabas CICS command-level link routine, which is the combination of LNKOLSC, LNKOLM, and the CICS entry and exit code. This name is used in EXEC CICS LINK commands to invoke Adabas services from CICS application programs. See also notes 1 and 2 in the installation procedure.

LADAFP: Length of Work Area for Adabas Fastpath Exit

Parameter	Description
`LADAFP={ 0 | nn}`	The length of the work area provided to the Adabas Fastpath exit. Values from 0 (the default) to 32767 may be specified. 0 indicates that Adabas Fastpath is not linked with the Adabas command-level link routine. A non-zero value requires that the parameter `TRUE=YES` is also set and the Adabas task-related user exit (TRUE) is used. Consult the Adabas Fastpath documentation for recommended values. **Note:** This parameter is not yet fully implemented. It is provided for future use by Adabas Fastpath.

LOGID: Default Logical Database ID

Parameter	Description
`LOGID= nnn`	The value of the default logical database ID. Valid ID numbers are 1-65535.

LRINFO: Length of Adabas Review Data Area

Parameter	Description
`LRINFO={ 0 | 256}`	The length (in bytes) of the Adabas Review data area to be used by the REVEXITB program. The default is zero (Adabas Review is not being used). The minimum (and recommended) value is 256, the size Adabas Review expects when the REVEXITB program is invoked. See the Adabas Review documentation for more information.

LUINFO: Length of User Data passed to Adabas UEXITA and UEXITB

Parameter	Description
`LUINFO={ 0 | length}`	Length of the user data to be passed from the CICS link routine to Adabas UEXITA and UEXITB. If `LUINFO` is not specified, the default is zero (no user save area is passed).

LUSAVE: Size of User Save Area for Adabas UEXITA and UEXITB

Parameter	Description
`LUSAVE={ 0 | size}`	Size of the user save area to be used by Adabas user exits UEXITA and UEXITB. If `LUSAVE` is specified, a value of 72 or higher must be specified. If `LUSAVE` is not specified, the default is zero (no user data is passed).

LXITAA: Length of Work Area provided to UEXITA

Parameter	Description
`LXITAA={ 0 | nn}`	Length of the work area provided to the UEXITA user exit program. Values from 0 (the default) to 32767 may be specified. 0 indicates that no UEXITA program is linked with the Adabas command-level link routine and no data is passed to UEXITA. **Note:** This parameter is not yet fully implemented. It is provided for future use by the CICS user exit A program linked with LNKOLM.

LXITBA: Length of Work Area for UEXITB

Parameter	Description
`LXITBA={ 0 | nn}`	Length of the work area provided to the UEXITB user exit program. Values from 0 (the default) to 32767 may be specified. 0 indicates that no UEXITB program is linked with the Adabas command-level link routine and no data is passed to UEXITB. **Note:** This parameter is not yet fully implemented. It is provided for future use by the CICS user exit A program linked with LNKOLM.

MRO: Multiple Region Option

Parameter	Description
`MRO={ NO | YES }`	The MRO parameter is used to indicate whether or not the CICS multiple region option is to be used. If you run the CICS command-level link with the CICS multiple region option (MRO), set `MRO=YES`; otherwise, use the default value `MRO=NO`. If `MRO=YES`, `NETOPT` must be set to `NETOPT=NO` (the default) to prevent non-unique LU names from multiple application regions. If `NETOPT=YES` and `MRO=YES` are specified, an assembler MNOTE and a return code of 16 are produced from the assembly step.

NETOPT: Method Used to Create User ID

Parameter	Description
`NETOPT={ NO | YES }`	If `NETOPT=YES` is specified, an 8-byte user ID will be constructed from the VTAM LU name. If `NETOPT=NO` is specified, the user ID is created from the constant "CICS" plus the four-byte CICS terminal ID (TCTTETI) for terminal tasks. For non-terminal tasks, the user ID comprises the constant "CIC" plus the CICS task number. If you run with the CICS multiple region option (MRO), you must use the default value for this option. If `NETOPT=YES` and `MRO=YES` are specified, an assembler MNOTE and a return code of 16 are produced from the assembly step.

NTGPID: Natural Group ID

Parameter	Description
`NTGPID=4-byte-value`	This parameter is used to specify a 4-byte Natural group ID as required for unique Adabas user ID generation in the CICSplex environment with Natural Version 2.2.8 and above. The value is associated with all users who call the Adabas command-level link routine assembled with the specified value. There is no default value. If no value is specified, the Adabas internal user ID is built in the conventional manner. Any 4-byte alphanumeric value may be specified, but it must be unique for each Adabas command-level link routine running in a CICSplex, or z/OS image. If more than one `NTGPID` is required (for example, both test and production Natural 2.2.8), more than one Adabas command-level link routine with associated TRUE must be generated. If you run with the CICS multiple region option (`MRO`), you may use `NTGPID` to provide a 4-byte literal for the Adabas communication ID to be used by the Adabas SVC when multiple application regions call Adabas.

NUBS: Number of User Blocks Created By CICS Link Routine

Parameter	Description
`NUBS={ 50 | blocks }`	The number of user blocks (UBs) to be created by the CICS link routine. The number of blocks must be large enough to handle the maximum possible number of concurrent Adabas requests. **Note:** The Adabas 6.2 and above command-level link routine obtains storage for the user blocks (the UB pool) above the 16-megabyte line.

PARMTYP: Area for Adabas Parameter List

Parameter	Description
`PARMTYP={ ALL | COM | TWA }`	The area which is to contain the Adabas parameter list. TWA picks up the parameter list in the first six fullwords of the transaction work area (TWA). COM picks up the list in the COMMAREA, followed by the normal Adabas parameter list. The COMMAREA list must be at least 32 bytes long and begin with the label ADABAS52. `PARMTYP=ALL` (the default) uses both the COMMAREA and TWA to pass the Adabas parameters; in this case, the COMMAREA is checked first. `PARMTYP=ALL` or `PARMTYP=COM` must be used if the `TRUE=YES` option is specified.

PURGE: Purge Transaction

Parameter	Description
`PURGE={ NO | YES }`	The `PURGE` parameter is used when assembling with CICS 3.2 or above. If `PURGE=YES` is specified, the CICS WAIT EXTERNAL will contain `PURGEABLE` as one of its parameters, allowing the transaction to be purged by CICS if the DTIMOUT value is exceeded and PURGE is specified. If `PURGE=NO` (the default) is specified, the `NONPURGEABLE` option is generated.

RMI: Resource Manager Interface

Parameter	Description
`RMI={ NO | YES }`	The `RMI` parameter is used to indicate whether or not the CICS Resource Manager Interface is to be used. If `RMI=YES` is specified, the Adabas task-related user exit (TRUE) will be executed as a resource manager (RM) using the CICS Resource Manager Interface (RMI). `RMI=YES` is valid only when the Adabas Transaction Manager is installed, enabled, and available to users executing in the CICS environment. Consult the Adabas Transaction Manager documentation for additional instructions related to the installation of the Adabas TRUE.

SAF: Adabas Security

Parameter	Description
`SAF={ NO | YES }`	The SAF paramter is to indicate whether or not the Adabas SAF Security (ADASAF) is used. The default `SAF=NO` must be used for z/VSE.

SAP: SAP Application Support

Parameter	Description
`SAP={ NO | YES }`	The `SAP` parameter is used to indicate whether or not Adabas support for the SAP application system is required. If `SAP=YES` is specified, the LNKOLSC program will detect a SAP initialization call and set the user ID for SAP applications from the constant provided on the initialization call, plus the field ACBADD2. For more information, refer to the supplementary information provided to customers using the SAP application system.

SVCNO: Adabas SVC number

Parameter	Description
`SVCNO={0 | nnn}`	The `SVCNO` parameter is used to specify the value of the Adabas SVC number.

TRUE: Adabas Task-Related User Exit

Parameter	Description
`TRUE={ NO | YES }`	The TRUE parameter is used to indicate whether or not the Adabas task-related user exit is to be used. If `TRUE=YES` is specified, LNKOLSC will use the Adabas task-related user exit LNKTRUE. If `TRUE=YES` is specified, the parameter settings `PARMTYP={ALL | COM}` and `TRUENM='name'` must also be specified.

TRUENM: Name of Adabas Task-Related User Exit

Parameter	Description
`TRUENM= 'name'`	The `TRUENM` parameter is used to specify the name of the Adabas task-related user exit. This parameter is required if `TRUE=YES` is specified. See also notes 1 and 2 in the installation procedure.

UBPLOC: User Block Pool Allocation

Parameter	Description
`UBPLOC= {ABOVE | BELOW}`	The `UBPLOC` parameter is used to specify whether the user block (UB) pool is to be obtained above (the default) or below the 16-megabyte line in CICS. The ECB used by the EXEC CICS WAIT WAITCICS or the EXEC CICS WAIT EXTERNAL is included in the UB pool. The `UBPLOC=BELOW` setting supports versions of CICS that do not allow ECBs above the 16-megabyte line; that is, CICS/ESA 3.2 or below. Refer to the IBM manual *CICS/ESA Application Programming Reference* for more information.

XWAIT: XWAIT Setting for CICS

Parameter	Description
`XWAIT={ NO | YES }`	Specifies whether a standard EXEC CICS WAIT EVENT (`XWAIT=NO`) or a CICS WAIT EVENTS EXTERNAL (`XWAIT=YES`) will be generated into the command-level link component by the assembler process in the LNKOLSC module: ▪ NO (the default) must be used for CICS/VS 2.3 and below. ▪ YES is valid for CICS TS 1.1 under z/VSE 2.4 and above. **Note:** All EXEC CICS commands are processed by the CICS preprocessor; the ADAGSET parameters cause the subsequent assembly step to skip some of the statements. `XWAIT=YES` is the recommended interface for CICS TS 1.1 under z/VSE 2.4 and above. Using the `XWAIT=NO` interface in these environments may result in poor CICS transaction performance or unpredictable transaction results in busy environments.

Installation Procedure

- Step 1: Modify the ADAGSET Macro for the Source Member(s)
- Step 2: Modify the JCS Members
- Step 3: Install the Adabas Command-Level Link Component
- Step 4: Install the Adabas Task-Related User Exit
- Step 5: Install the Adabas PLT-Enable Program (SMA Job Number I070)
- Step 6: Install the DISPGWA Program (Optional)

Step 1: Modify the ADAGSET Macro for the Source Member(s)

Modify the ADAGSET macro for the source members to be used.

See the section *Modifying Source Member Defaults (ADAGSET)* for details. Software AG recommends that you modify a common version of ADAGSET and place it in a library available in the SYSLIB concatenation when the Adabas command-level link components are assembled.

Note: It is no longer necessary to modify the equates inside the LNKOLSC code. Instead, use the ADAGSET macro to set default values before assembly, thus making the process easier and more self-documenting.

For the Standard Installation (Without Enhanced Functions)

Modify the source member ADAGSET to set the following options:

- database ID (LOGID);
- Adabas SVC number (SVCNO);
- any additional options necessary for your site.

For the Enhanced Installation

Modify the source member ADAGSET to set the following options:

- database ID (LOGID);
- Adabas SVC number (SVCNO);
- the command-level link routine name (ENTPT);
- the task-related user exit name (TRUENM);
- any additional options necessary for your site.

Notes:

1. It is critically important that the `ENTPT` and `TRUENM` parameters coded in the LNKENAB, LNKTRUE, and LNKOLSC modules are identical. These module names are used to identify the global work area and task-related user exit (TRUE) storage provided for the CICS transaction using the link-specific link component.
2. If you are installing multiple instances of the command-level link routine, the `ENTPT` and `TRUENM` names must be unique, as there is a one-for-one relationship between a command-level link routine, its associated task-related user exit (TRUE), and the enabling program LNKENAB run at CICS startup.

Step 2: Modify the JCS Members

Use your editor to modify the JCS members necessary for your installation, and set the library names according to your installation's specifications. The Adabas job library contains the following JCS members:

JCS Member	Installation Type	Required/Optional	This job preprocesses, assembles, and links ...
CICSASMC	standard and enhanced	required	enhanced LNKOLSC and LNKOLM to create ADAOLSC and ADAOLM; then, links these modules with the CICS prolog and epilog stubs to create the Adabas CICS command-level link component.
CICSASMD	enhanced	required	the Adabas global work area display program DISPGWA to create the ADATEST module.
CICSASME	enhanced	required	the LNKENAB PLTPI initialization program ADAENAB.
CICSASMT	enhanced	optional	the LNKTRUE source module to create the Adabas task-related user exit module ADATRUE.

Instructions for modifying the JCS are included in the JCS members themselves. These include but may not be limited to the following:

1 Correct the POWER control statements:

change ...	to ...
$*	/*
$&	/&
X $$	* $$

2 Replace the following symbols with site-specific values:

change ...	to ...
LLLLLLLL	Adabas library DLBL name
SSSSSSSS	Adabas sublibrary for cataloging
UUUUUUUU	user ID
EXT1	extent start value for work file 1
EXT2	extent start value for work file 2
EXT1L	extent length value for work file 1
EXT2L	extent length value for work file 2
VVVVVV	volume ID for work files

Step 3: Install the Adabas Command-Level Link Component

to install the Adabas command-level link component, for the JCS members CICSASMC, CICSASMD, CICSASME, and CICSASMT:

1 Include the proper EXEC PROC= statements to define the Adabas library.

2 Modify the LIBDEF statements for proper search order and for the proper sublibrary for cataloging the PHASEs.

3 Ensure that the library containing the PHASEs will be available to the CICS system at execution time.

The JCS members use two disk work files for SYSPUNCH output from the preprocessor and assembler steps.

4 Check the DLBL and EXTENT information for these work files.

5 Use DFHCSDUP or the CEDA RDO entry panels to add the following definition to your CICS CSD file:

```
DEFINE PROGRAM(ADABAS) GROUP(ADABAS)
DESCRIPTION(ADABAS V74s COMMAND LEVEL LINK ROUTINE)
LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(YES) USAGE(NORMAL)
USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
EXECKEY(USER) EXECUTIONSET(FULLAPI)

—where s is the system maintenance level of Adabas.
```

The Adabas command-level link routine is now installed. This completes the standard installation. To install the enhanced functions, continue with step 4.

Step 4: Install the Adabas Task-Related User Exit

The Adabas CICS components reside in the Adabas base source library.

to install the Adabas Task-Related User Exit:

1 Execute CICSASME.

This job preprocesses, assembles, and links the Adabas task-related user-exit-enabling program LNKENAB (ADAENAB).

2 Use DFHCSDUP or the CEDA RDO entry panels to add the following definition to your CICS CSD file:

```
DEFINE PROGRAM(ADATRUE) GROUP(ADABAS)
DESCRIPTION(ADABAS V74s TASK RELATED USER EXIT)
LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(YES) USAGE(NORMAL)
USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
EXECKEY(CICS) EXECUTIONSET(FULLAPI)
```

—where *s* is the system maintenance level of Adabas.

3 After defining LNKENAB to CICS, add the following entry to your PLTPI table, DFHPLT:

```
DFHPLT TYPE=ENTRY,PROGRAM=ADAENAB
```

This entry should follow the first DFHPLT `TYPE=DELIM` statement to ensure that LNKENAB will be executed in either stage II or stage III of the CICS PLTPI process. This is necessary because the CICS EXEC interface environment must be present to support the writing of console messages using the EXEC CICS `WRITE OPERATOR` command employed by the LNKENAB module.

4 Code an appropriate `PLTPI=xx` parameter in the CICS start-up data. *xx* should match the suffix value given in the DFHPLT table.

The Adabas command-level link enhanced functions are now installed. This completes the enhanced installation; however, you may optionally continue with Step 6, Installing the DISPGWA Program.

Step 5: Install the Adabas PLT-Enable Program (SMA Job Number I070)

The Adabas CICS components reside in the Adabas base source library ADA74*s*.SRCE.

to install the Adabas PLT-enable program:

1 Execute CICEASM.

This job preprocesses, assembles, and links this module into a staging library, and then links it with CICS entry and exit code into a CICS RPL library.

2 Use DFHCSDUP or the CEDA RDO entry panels to add the following definition to your CICS CSD file:

```
DEFINE PROGRAM(ADAENAB) GROUP(ADABAS)
DESCRIPTION(ADABAS V74s PLTPI ENABLE ADATRUE PROGRAM)
LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
USE
EXECKEY(CICS) EXECUTIONSET(FULLAPI)
```

—where *s* is the system maintenance level of Adabas.

3 After defining LNKENAB to CICS, add the following entry to your PLTPI table, DFHPLT:

```
DFHPLT TYPE=ENTRY,PROGRAM=ADAENAB
```

This entry should follow the first DFHPLT `TYPE=DELIM` statement to ensure that LNKENAB will be executed in either stage II or stage III of the CICS PLTPI process. This is necessary because the CICS EXEC interface environment must be present to support the writing of console messages using the EXEC CICS `WRITE OPERATOR` command employed by the LNKENAB module.

4 Code an appropriate `PLTPI=`*xx* parameter in the CICS start-up data. *xx* should match the suffix value given in the DFHPLT table.

Step 6: Install the DISPGWA Program (Optional)

to install the DISPGWA program:

1. Execute CICDASM.

   This job preprocesses, assembles, and links this module into a staging library, then links it with CICS entry and exit code into a CICS RPL library. The final link step creates the load module ADATEST.

2. Add a CICS transaction to execute the ADATEST program. RDO may be used to do this. Sample DEFINE statements for the ADATEST (DISPGWA) program and the transaction to execute it are:

```
DEFINE PROGRAM(ADATEST) GROUP(ADABAS)
DESCRIPTION(ADABAS V74s DISPLAY GWA PROGRAM - DISPGWA)
LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
EXECKEY(CICS) EXECUTIONSET(FULLAPI)

DEFINE TRANSACTION(DGWA) GROUP(ADABAS)
DESCRIPTION(TRANSACTION TO DISPLAY ADABAS GWA)
PROGRAM(ADATEST) TWASIZE(128) PROFILE(DFHCICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(CICS) STORAGECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO) INDOUBT(BACKOUT)
RESTART(NO) SPURGE(NO) TPURGE(NO) DUMP(YES) TRACE(YES)
RESSEC(NO) CMDSEC(NO)
```

   —where *s* is the system maintenance level of Adabas.

   The Adabas command-level link routine, enhanced functions, and DISPGWA program are now installed.

Installing the CICS High-Performance Stub Routine

The Adabas high-performance stub routine extends the direct call interface (DCI) facility that is available with the Adabas CICS command-level link component to applications written in languages other than Software AG's Natural (for example, Assembler, COBOL, PL/I).

Note: The stub routine must be used with the Adabas CICS command-level link component. The stub routine will not function properly with the Adabas CICS macro-level link component.

The DCI allows a CICS TS 1.1 application running under z/VSE 2.4 or above to call Adabas through the Adabas command-level link routine. The overhead incurred when the EXEC CICS LINK and

EXEC CICS RETURN command set is used to transfer program control is thus avoided. Once the proper environment has been established with the initial call (IC) command from the high-performance stub or Natural 3.1 or above, the DCI permits a BALR interface to be used.

The high-performance stub routine is written in Assembler language. When linked with the application program, it serves as an interface between the application and the Adabas CICS command-level link component. The application program can then issue CALL statements to access the stub routine when executing an Adabas command.

A CICS TS 1.1 application under z/VSE 2.4 or above derives the following advantages from the high-performance stub:

- improved performance and throughput when issuing Adabas commands under CICS/ESA 3.2 or above due to the reduced use of CICS services related to the CICS LINK and RETURN program control mechanism.
- a call mechanism for Adabas requests under CICS/ESA 3.2 or above which is simpler than the methods normally employed to pass control with information from one program to another in the CICS environment.

Restrictions and Requirements

The following restrictions and requirements apply to the high-performance stub routine:

- CICS TS 1.1 under z/VSE 2.4 Required

  The Adabas high-performance stub routine is supported for CICS TS 1.1 under z/VSE 2.4 and above. Earlier versions of CICS are not supported.

  A CICS transaction work area (TWA) of at least 28 bytes must be provided to the application for the proper execution of the high-performance stub routine.
- CICS Command-Level Link Required

  The application program must be written using the CICS command-level interface and instructions, and may not issue any CICS macro level commands.
- Supported Programming Languages

  The application program may be written in ALC (Assembler language), COBOL, COBOL II, PL/I, or C. Installation verification programs (IVPs) are provided in ALC and COBOL on the distribution tape.

Additional requirements for specific programming languages are discussed later in the sections relating to each language.

Stub Components

Type	Member	Description
Source	ALCSIVP COBSIVP LNCSTUB	source for ALC install verification source for COBOL install verification source for high-performance stub
Job control	JCLALCI.X JCLCOBI.X JCLLNCS.X	sample JCL for ALC installation verification sample JCL for COBOL installation verification sample JCL LNCSTUB (high-performance stub). The final step of the job catalogs the LNCSTUB.OBJ module so it is ready to be included when the application program is linked. See the JCLALCI or JCLCOBI sample jobs.

Installation Overview

Use the following procedure to install the Adabas CICS high-performance stub routine:

- Edit, preprocess, assemble and link the LNCSTUB module.
- (Optional) Modify, preprocess, compile or assemble, link, and execute the desired installation verification program (IVP).
- Modify, preprocess, compile or assemble, link, and execute the application programs.

Step 1: Install the LNCSTUB Module

The Adabas CICS high-performance stub routine is an Assembler language module provided in source form on the distribution tape in member LNCSTUB.

Step 1 has the following substeps:

- Edit the ADAGSET macro.
- Change the LNCNAME field value, if necessary.
- Modify member JCLLNCS.X.
- Preprocess, assemble, and link the LNCSTUB module.
- Place the LNCSTUB load module in a library that is available to your application programs when they are linked.

Edit the ADAGSET Macro

Note: For information about editing the ADAGSET macro, refer to the section *Modifying Source Member Defaults (ADAGSET Macro)*.

Edit the LNCSTUB module to provide parameters on the ADAGSET macro located at the beginning of the source deck. The values provided to the ADAGSET macro should match those provided on the Adabas CICS command-level link routine LNKOLSC.

The values given for the ADAGSET parameters are primarily for documentation purposes within the LNCSTUB module, but may be used at a later time in the stub routine at the discretion of Software AG.

Change the LNCNAME Field Value

If your Adabas CICS command-level link component program has been linked with a name other than ADABAS, change the constant value in the field LNCNAME to match the name used (see the **ADAGSET option ENTPT**). The value in this field is used in the priming EXEC CICS LINK command issued by LNCSTUB.

Modify Member JCLLNCS.X

Member JCLLNCS.X is used to preprocess, assemble, and catalog the LNCSTUB module.

To modify this JCL to meet your site requirements

1 Modify the POWER job information, CLASS, DEST, and LDEST values:

2 Throughout the member, make the following changes:

change ...	to ...
$*	/*
$&	/&
X $$	* $$

3 Provide any site-specific names for the Adabas JCS PROC or other required PROCs.

4 Replace the following symbols with site-specific values:

change ...	to ...
SSSSSSSS	Adabas sublibrary for cataloging
UUUUUUUU	user ID
EXT1	extent start value for work file 1
EXT2	extent start value for work file 2
EXT1L	extent length value for work file 1
EXT2L	extent length value for work file 2
VVVVVV	volume ID for work files

5 Software AG recommends that you use the high-level Assembler to assemble the LNCSTUB module.

If you must use the old VSE Assembler, supply the following statement on the assembly step to provide the assembly date:

```
// OPTION SYSPARM='yymmdd'
—where
yy is the 2-digit year
mm is the 2-digit month
dd is the 2-digit day
```

6 Check the access and catalog statements on the final LIBR step that catalogs the LNCSTUB.OBJ module.

Preprocess, Assemble, and Link the LNCSTUB Module

Because of the possible use of the 31-bit instructions, high-level Assembler (ASMA90) should be used to assemble the LNCSTUB module after CICS preprocessing.

to preprocess, assemble, and link the LNCSTUB Module

1 Provide library definitions for the Adabas and CICS libraries using PROCs or DLBL statements.

2 Construct LIBDEF statements for the search order and for the library where the LNCSTUB.OBJ module is to be cataloged.

Make the LNCSTUB Available to Application Programs

The LNCSTUB module has an entry name of ADABAS, which can be used by the application program as the object of a CALL statement to pass control to LNCSTUB with a list of parameters. The language-specific calling conventions for LNCSTUB are discussed later in this section.

The LNCSTUB load module must be available to the link step of the application program that is to use the DCI facility.

Note: In the same step, the CICS load library should be available; otherwise, the external references to the CICS stub modules will not be resolved.

Place the LNCSTUB load module in a library available to your application language assembler or compiler so that it will be included when the application programs are linked.

Step 2: (Optional) Install and Execute an IVP

Two installation verification programs (IVPs) are provided in source form: one for Assembler language, and one for COBOL. These programs are samples for implementing the Adabas high-performance stub routine in your applications. They also provide a way of verifying the proper installation of the LNCSTUB module.

Step 2 has the following substeps:

- Modify the Assembler (ALCSIVP) or COBOL (COBSIVP) source decks to provide the proper Adabas database ID and file number on your site's database for the Software AG-provided PERSONNEL file.
- Modify the JCL provided to preprocess and compile (assemble) the desired IVP.
- Preprocess, compile or assemble, and link the IVP using the sample JCL provided as a guide.
- Add PPT and PCT entries to your CICS system to execute the IVPs.
- Execute the IVPs to verify the LNCSTUB module (ALCSIVP and COBSIVP).

Install and Execute the Assembler IVP: ALCSIVP

The source member ALCSIVP is provided to demonstrate and verify the use of the Adabas DCI using the LNCSTUB module. This program issues a series of Adabas commands using the conventional CICS LINK/RETURN mechanism, produces a partial screen of output data, then reexecutes the same call sequence using the Adabas DCI and the LNCSTUB subprogram.

to modify source member ALCSIVP:

1. Edit the database ID and file number fields DBID (line 321) and DBFNR (line 322) to be sure they match the values needed to access the PERSONNEL file on the database you intend to use.
2. Check the fields FBUFF, SBUFF and VBUFF for values consistent with your PERSONNEL file's FDT and data content.
3. Check the name used in the EXEC CICS LINK statement (line 242) to be sure it matches the name of your Adabas CICS command-level link component program.

to modify the JCLALCI.X sample job stream:

1. Member JCLALCI.X is used to preprocess, assemble, and link the Assembler IVP. Place the ALCIVP.PHASE in a library that will be available to your CICS partition.
2. Modify JCLALCI.X to meet your site requirements.

   Modify the POWER job information, CLASS, DEST, and LDEST values:

Throughout the member, make the following changes:

change ...	to ...
$*	/*
$&	/&
X $$	* $$

Provide any site-specific names for the Adabas JCS PROC or other required PROCs.

Replace the following symbols with site-specific values:

change ...	to ...
SSSSSSSS	Adabas sublibrary for cataloging
UUUUUUUU	user ID
EXT1	extent start value for work file 1
EXT2	extent start value for work file 2
EXT1L	extent length value for work file 1
EXT2L	extent length value for work file 2
VVVVVV	volume ID for work files

3 Software AG recommends that you use the high-level Assembler to assemble the LNCSTUB module.

If it is necessary to use the old VSE Assembler, supply the following statement on the assembly step to provide the assembly date:

```
// OPTION SYSPARM='yymmdd'
—where
yy is the 2-digit year
mm is the 2-digit month
dd is the 2-digit day
```

to preprocess, assemble, and link ALCSIVP:

- Using the modified sample JCLALCI.X member, preprocess, assemble, and link ALCSIVP.

to add PPT and PCT entries:

- Add the following PPT and PCT entries to your CICS system, or use the RDO facility to add the STB1 transaction to run the ALCSIVP program:

```
DFHPPT TYPE=ENTRY,PROGRAM=ALCSIVP
DFHPCT TYPE=ENTRY,PROGRAM=ALCSIVP,TRANSID=STB1, X
TPURGE=YES,SPURGE=NO,TWASIZE=28
```

to execute ALCSIVP:

- Run the STB1 transaction to execute ALCSIVP. Executing ALCSIVP verifies the LNCSTUB module.

Install and Execute the COBOL IVP: COBSIVP

Member COBSIVP illustrates the use of the Adabas DCI with a COBOL program. COBIVP produces a screen showing output lines produced by a series of Adabas calls executed by the CICS LINK/RETURN facility, followed by the reexecution of these Adabas commands using the DCI.

to modify source member COBSIVP:

1. Edit the fields WORK-DBID and WORK-FNR to place the desired database ID and file number in the VALUE clauses to access the PERSONNEL file on your site's database.
2. Ensure that the value in the field LINK-NAME matches the name used in your Adabas CICS command-level link component program.
3. Ensure that the values (literals in the PROCEDURE DIVISION) in the following fields are consistent with the requirements of the PERSONNEL file FDT and data content you are using:

```
ADABAS-FORMAT-BUFFER,
ADABAS-SEARCH-BUFFER, and
ADABAS-VALUE-BUFFER
```

to modify the JCLCOBI.X sample job stream:

1. Member JCLCOBI.X is used to preprocess, compile, and link the COBSIVP installation verification program.

   Modify JCLCOBI.X to meet your site requirements.

   Modify the POWER job information, CLASS, DEST, and LDEST values:

Throughout the member, make the following changes:

change ...	to ...
$*	/*
$&	/&
X $$	* $$

Provide any site-specific names for the Adabas JCS PROC or other required PROCs.

Replace the following symbols with site-specific values:

change ...	to ...
cuu	channel and unit address for SYSPCH
lib	Adabas library DLBL name
sublib	Adabas sublibrary for cataloging
UUUUUUUU	user ID
rtrk	relative track number for the extent
ntrks	number of tracks in the extent
volid	volume ID for work files

Replace the names in lowercase with the site-specific library and dataset names.

2 Member JCLCOBI.X is used to preprocess, compile, and link the COBSIVP installation verification program.

to preprocess, compile, and link COBSIVP:

1 Use the modified JCLCOBI.X job to preprocess, compile, and link the COBSIVP program. Assemble ADASTWA into a library available to COBOL programs when they are linked. Include the ADASTWA load module in the link of COBSIVP.

COBSIVP uses the ADASTWA subroutine when it issues Adabas calls through the standard CICS LINK/RETURN mechanism. ADASTWA is supplied in source form in the Adabas source library.

The LNCSTUB subroutine does not use ADASTWA because it places the passed Adabas parameters in the TWA. Thus, the ADASTWA routine is not required when linking COBOL applications that utilize the Adabas DCI through the LNCSTUB module.

Even though the LNCSTUB routine does not require the ADASTWA subroutine, it is included in the COBSIVP program to illustrate the usual way in which a COBOL application places the Adabas call parameters into the CICS TWA.

2 Link the COBSIVP program with the LNCSTUB object module and the ADASTWA object module. Make the LNCSTUB load module available to the linkage editor to be included with the COBSIVP load module.

Note: The CICS stub modules are also resolved in the link step.

to add PCT and PPT entries:

- Add the following PPT and PCT entries to your CICS system, or use the RDO facility to add the STB2 transaction to run the COBSIVP program:

```
DFHPPT TYPE=ENTRY,PROGRAM=COBSIVP
DFHPCT TYPE=ENTRY,PROGRAM=COBSIVP,TRANSID=STB2, X
TPURGE=YES,SPURGE=NO,TWASIZE=28
```

to execute COBSIVP:

- Run the STB2 transaction to execute COBSIVP. Executing COBSIVP verifies the LNCSTUB module.

Step 3: Link and Execute the Application Program

Once the IVP programs have been successfully executed, the Adabas DCI is ready to be used with real application programs. In step 3, the application program interface (API) is coded to utilize the LNCSTUB subprogram.

Step 3 has the following substeps:

- Modify the application programs that will utilize the Adabas CICS high-performance stub routine in accordance with the guidelines described in the following section.
- Preprocess, compile or assemble, and link the application programs to include the LNCSTUB module.
- Execute the application programs using the Adabas CICS high-performance stub.

Guidelines for Modifying the Application Program

The LNCSTUB load module must be linked with your application program. The application program invokes the DCI interface using a standard batch-like call mechanism. The LNCSTUB module makes any additional CICS requests required to pass data to the Adabas CICS command-level link component.

- Programming Languages Supported by LNCSTUB

The LNCSTUB program functions with application programs written in Assembler language, VS/COBOL, COBOL II, PL/I, and C.

- Transaction Work Area Required

  A transaction that uses the Adabas DCI or the Adabas CICS command-level link component must provide a transaction work area (TWA) at least 28 bytes long. Failure to provide an adequate TWA will result in an abend U636 (abnormal termination of the task).

- Reentrant Requirement

  The application program may or may not be reentrant. The LNCSTUB module has been written to be reentrant, but using linkage editor parameters to mark the LNCSTUB load module as reentrant is not recommended.

- CICS Requests Issued by LNCSTUB

  The LNCSTUB module issues the following command-level CICS requests whenever it is invoked:

  ```
  EXEC CICS ADDRESS TWA
  EXEC CICS ASSIGN TWALENG
  EXEC CICS ADDRESS EIB
  ```

- DCI Entry Point Address

  An EXEC CICS LINK command is issued by LNCSTUB at least once to acquire the DCI entry point from the Adabas CICS command-level link component program. This address is then used for BALR access on all subsequent Adabas calls for a transaction. Thus, the calling application program must provide a fullword (4-byte) field to hold the DCI entry point address obtained by LNCSTUB. This 4-byte field is the first parameter passed to the LNCSTUB module by the call mechanism. The remaining parameters comprise the standard Adabas parameter list needed to execute an Adabas request.

- DCI Parameter List

  The Adabas DCI parameter list expected by the LNCSTUB program is composed of a pointer to the DCI entry point in the Adabas CICS command-level link component followed by the six pointers to the Adabas control block and buffers: format, record, search, value, and ISN.

  For information on coding the standard Adabas control block and buffers, refer to the *Adabas Command Reference*.

The parameter list offsets are summarized in the table below:

Offset	Pointer to the ...
0	DCI entry point in the Adabas command-level link component
4	Adabas control block
8	Adabas format buffer
12	Adabas record buffer
16	Adabas search buffer
20	Adabas value buffer
24	Adabas ISN buffer

All of the parameters except the first (the DCI entry point) are built and maintained by the application program in accordance with the requirements of an Adabas call.

The DCI entry point parameter should be set to binary zeros at the beginning of a task, and should not be modified by the application program thereafter. Software AG strongly recommends that the fields comprising the parameter list be placed in CICS storage (WORKING-STORAGE for COBOL and the DFHEISTG user storage area for Assembler) to maintain pseudo-reentrability.

The following is a sample parameter list for an assembler language program:

```
DFHEISTG DSECT
.
PARMLIST DS OF
DS A(DCIPTR)
DS A(ADACB)
DS A(ADAFB)
DS A(ADARB)
DS A(ADASB)
DS A(ADAVB)
DS A(ADAIB)
.
DCIPTR DS F
ADACB DS CL80
ADAFB DS CL50
ADARB DS CL250
ADASB DS CL50
ADAVB DS CL50
ADAIB DS CL200
.
DFHEIENT CODEREG=(R12),EIBREG=(R10),DATAREG=(R13)
.
LA R1,PARMLIST
L R15,=V(ADABAS)
BALR R14,R15
```

```
.
END
```

Note: The DFHEIENT macro in the Assembler example uses a `DATAREG` parameter of register 13. This is a strict requirement of the LNCSTUB program. When the LNCSTUB program is invoked, register 13 should point to the standard CICS save area (DFHEISA) and register 1 should point to the parameter list. The best way to ensure this standard is to code the Assembler application with a DFHEIENT macro like the one in the example.

The following is a sample parameter list for a COBOL language program:

```
WORKING-STORAGE SECTION.
.
01 STUB-DCI-PTR PIC S9(8) COMP VALUE ZERO.
01 ADACB PIC X(80).
01 ADAFB PIC X(50).
01 ADARB PIC X(250).
01 ADASB PIC X(50).
01 ADAVB PIC X(50).
01 ADAIB PIC X(200).
.
PROCEDURE DIVISION.
.
CALL 'ADABAS' USING STUB-DCI-PTR,
ADACB,
ADAFB,
ADARB,
ADASB,
ADAVB,
ADAIB.
.
EXEC CICS RETURN END-EXEC.
.
GOBACK.
```

- Restrictions on Application Program Coding

  In all other respects, the application program should be coded like a standard CICS command-level routine. As long as the DCI parameter list is correct when LNCSTUB is called, there are no restrictions on the CICS commands that an application can issue.

- Standard Batch Call Mechanism Used

  As shown in the Assembler and COBOL language program parameter list examples above, the call to "ADABAS" (the LNCSTUB entry point) is accomplished like a batch application. Likewise, calls for the other supported languages should be coded with their standard batch call mechanisms.

Link the Application Programs to Include the LNCSTUB Module

To properly link the LNCSTUB module with application programs, link the application program to include the LNCSTUB module and the CICS stub modules. The method for doing this varies with the programming language used for the application:

- Assembler language programs should include the DFHEAI and DFHEAI0 CICS modules;
- COBOL applications should include DFHECI and DFHEAI0.

To avoid a double reference to the DFHEAI0 module, code the linkage editor REPLACE DFHEAI0 control statement at the beginning of the SYSLIN data deck.

for linking Assembler language programs:

- For an Assembler program, the SYSLIN input is similar to:

```
INCLUDE DFHEAI
```

For an Assembler program, the input to the linkage editor is

```
PHASE pgmname,*
MODE AMODE(ANY),RMODE(ANY)
INCLUDE DFHEAI
INCLUDE LNCSTUB
INCLUDE DFHEAI0
ENTRY pgmname

—where pgmname is the name of your application program.
```

The Assembler object deck looks like

```
REPLACE DFHEAI0
INCLUDE SYSLIB(LNCSTUB)
INCLUDE SYSLIB(DFHEAI0)
NAME ALCSIVP(R)
```

When examining the cross-reference from the linkage editor, the symbol ADABAS must have the same starting location as the LNCSTUB module in the link map.

for linking COBOL language programs:

- For a COBOL program, the input to the linkage editor is

  ```
  PHASE pgmname,*
  MODE AMODE(ANY),RMODE(ANY)
  INCLUDE LNCSTUB
  INCLUDE DFHECI
  ENTRY pgmname

  —where pgmname is the name of your application program.
  ```

  The COBOL object input is similar to:

  ```
  INCLUDE SYSLIB(LNCSTUB)
  INCLUDE SYSLIB(DFHEAIO)
  NAME COBSIVP(R)
  ```

  When examining the cross-reference from the linkage editor, the symbol ADABAS must have the same starting location as the LNCSTUB module in the link map.

for linking PL/I and C language programs:

- Refer to the IBM manual *CICS/ESA System Definition Guide* for information about linking PL/I and C applications under CICS.

Performance Using LNCSTUB

To obtain the best performance from applications using the Adabas direct call interface (DCI), examine how the DCI interface functions at the logical level.

A CICS application using the standard LINK/RETURN mechanism to access the Adabas link routines invokes the CICS program control service for every Adabas request made to the link routine. The LNCSTUB module permits a BALR interface to be used. A BALR interface can substantially reduce the CICS overhead required to pass control from the application program to the Adabas CICS command-level link component.

The LNCSTUB module accomplishes this by using the standard EXEC CICS LINK/RETURN mechanism to make an Initial Call (IC) to the Adabas CICS command-level link routine. The link routine recognizes this call, and returns the entry point address of the DCI subroutine to LNCSTUB. LNCSTUB must then save this address in a location that can be assured of existence throughout the duration of the invoking task. This is why the calling program must provide the 4-byte field to hold the DCI entry point address. After the DCI address has been obtained, and for as long as LNCSTUB receives this address as the first parameter passed to it on subsequent Adabas calls, LNCSTUB utilizes the BALR interface to pass control to the Adabas CICS command-level link component program.

As a consequence of this logic, the more Adabas requests made between ICs, the more efficient the application in terms of passing data to and from Adabas under CICS. In fact, pseudo-conversational applications that issue one Adabas call each time a task is invoked should not be coded to use the DCI because there will be an IC request for each Adabas command issued by the calling program.

An additional performance improvement can be realized by taking advantage of the fact that the Adabas CICS command-level link component program must be defined as resident in CICS. This fact should allow the DCI entry point to be stored across CICS tasks, making it possible for different programs to call the LNCSTUB module with a valid DCI entry point. The IC at each program startup is thus avoided. When this procedure is used, however, any change to the CICS environment that invalidates the entry point address (such as a NEWCOPY) will lead to unpredictable and possibly disastrous results.

It is imperative that at least one IC be made to the Adabas CICS command-level link component program using CICS services. This call is used to trigger the acquisition of shared storage for the Adabas user block (UB) and (in the case of migration aids) an array of register save areas. If no IC request is made, Adabas calls will not execute due to a lack of working storage, and to the fact that critical control blocks used by the link routines and the Adabas SVC are not built.

Installing Adabas with Com-plete

Certain Adabas parameters are required by Com-plete, Software AG's TP monitor, when installing Adabas. For more information, see the *Com-plete System Programmer's* manual.

For Com-plete Versions 4.5 and above, the link routine for Com-plete initialization (module/phase ADALCO) is provided in the Adabas distribution library. ADALCO is loaded during Com-plete initialization to service Adabas calls. The Adabas library containing ADALCO should be placed in the z/VSE LIBDEF search chain.

Com-plete branches to the module/phase ADALCO according to the `AMODE` setting established in the link of ADALCO. The Adabas Version 7.4 ADALCO routine uses the ESA BASSM and BSM instructions to

- call the Com-plete wait service in the proper `AMODE`; and
- return to the caller of ADALCO in the caller's `AMODE`.

Installing Adabas with Shadow

The Adabas link routine specific to Shadow, ADALNS, is provided in source form along with the job SHADASM to assemble it. SHADASM must be customized to select the Shadow macro (source) library containing the macros SAVED, $WAIT, RELOCD, RETURND, TCBD.

Selecting Options for ADALNS

Customizing the source member ADALNS means selecting the following options:

Option	Default	Specify . . .
SVCNR	0	Adabas SVC number
LOGID	1	Default logical database ID (range 1-255).
NUBS	50	Number of UBs (user blocks) to be created by ADALNS. This must be high enough to handle the maximum possible number of concurrent Adabas requests.
PLINTWA	Y (yes)	N if the Adabas parameter list is passed in register 1 instead of at offset 0 in the Shadow TWA.
LNUINFO	0	Length for the user data to be passed from the ADALNS link routine to the Adabas user exit 4.

Shadow Table Entry for ADALNS

The user must specify the following entry in the Shadow PCT table:

```
PCT PROG=ADABAS,DISP=INITL,LANG=BAL,SAVE=YES,PURGE=YES
```

It is important that Adabas be made resident. Under Shadow, the `ADABAS` parameter is normally passed in the first 24 bytes of the TWA.

The user exit called from ADALNS gains control before the Adabas call (UEXITB), and can be used to modify the eight-byte UBUID field. This allows users who process the command log to have a unique terminal, since the command log presently contains only a four-byte field. This field does not contain a unique ID. The user exit could then be used to make the first four bytes unique. The user exit must create a unique user exit for each user. For Shadow, the UBUID field normally contains the constant SHAD in the high-order four bytes, followed by the value from ITRMTYPE.

Installing Adabas with Batch / TSO

ADALNK is the standard Adalink for running Adabas in batch. ADALNKR (LNKVSER) is supplied as a reentrant batch link routine.

However, it is important to note that user programs linked with ADAUSER also load ADARUN. ADARUN, in turn, loads other modules. To start a user program linked with ADAUSER, the following modules must be available in the LIBDEF search chain:

```
ADAIOR
ADAIOS
ADALNK
ADAMLF
ADAOPD
ADAPRF
ADARUN
```

6 Device and File Considerations

This section provides information for the following device- and system file-related topics:

- installing on fixed-block addressing (FBA) devices;
- defining new devices; and
- changing defaults for sequential files.

Supported Device Types

The standard characteristics of the device types supported by Adabas are summarized in the following table. The Adabas block sizes and RABNs per track are provided for each component for each device type.

Device	Trks/Cyl	ASSO	DATA	WORK	PLOG/RLOG	CLOG	TEMP/SORT/DSIM	Notes
1512	16	2048:128	4096:64	4096:64	4096:64	4096:64	4096:64	
2512	16	4096:64	8192:32	8192:32	8192:32	8192:32	8192:32	
3375	12	2016:15	4092:8	4096:8	4096:8	4096:8	8608:4	
3380	15	2004:19	4820:9	5492:8	5492:8	4820:9	7476:6	2
3390	15	2544:18	5064:10	5724:9	5724:9	5064:10	8904:6	2
3512	16	4096:64	16384:16	16384:16	16384:16	16384:16	16384:16	
5121	15	2048:16	4096:8	4096:8	4096:8	4096:8	4096:8	
5122	15	4096:8	8192:4	8192:4	8192:4	8192:4	8192:4	
5123	15	4096:8	16384:2	16384:2	16384:2	16384:2	16384:2	
8345	15	4092:10	22780:2	22920:2	22920:2	22920:2	22920:2	
8380	15	3476:12	6356:7	9076:5	9076:5	9076:5	9076:5	1
8381	15	3476:12	9076:5	11476:4	11476:4	9076:5	9076:5	1
8385	15	4092:10	23292:2	23468:2	23468:2	23468:2	23468:2	1
8390	15	3440:14	6518:8	10706:5	10706:5	8904:6	8904:6	1
8391	15	4136:12	10796:5	13682:4	13682:4	8904:6	18452:3	1
8392	15	4092:12	12796:4	18452:3	18452:3	18452:3	18452:3	1
8393	15	4092:12	27644:2	27990:2	27990:2	27990:2	27990:2	1
9345	15	4092:10	7164:6	11148:4	11148:4	22920:2	22920:2	2

Notes:

1. The 8350, 838*n*, and 839*n* are pseudo-device types physically contained on a 3350, 3380, and 3390 device, respectively, but for which some or all of the standard block sizes are larger.
2. The IBM RAMAC 9394 emulates devices 3390 Model 3, 3380 Model K, or 9345 Model 2.

FBA Devices

All device definitions for Adabas control statements for FBA disks should specify one of the following devices types:

- FBA SCSI devices: Specify device types of 1512, 2512, or 3512.
- Virtual FBA devices: Specify device types of 5121, 5122, or 5123.

Note: Virtual FBA devices are not permanent and are, therefore, only suitable for holding temporary or work data sets.

Choose a device type based on the block sizes given in the following tables:

SCSI Device Types:

Dev Type	Asso blksz	Data blksz	Work blksz	Temp blksz	Sort blksz	PLOG blksz	CLOG blksz
1512	2048	4096	4096	4096	4096	4096	4096
2512	4096	8192	8192	8192	8192	8192	8192
3512	4096	16384	16384	16384	16384	16384	16384

Virtual FBA Device Types:

Dev Type	Asso blksz	Data blksz	Work blksz	Temp blksz	Sort blksz	PLOG blksz	CLOG blksz
5121	2048	4096	4096	4096	4096	4096	4096
5122	4096	8192	8192	8192	8192	8192	8192
5123	4096	16384	16384	16384	16384	16384	16384

The pseudo-cylinder for each of these devices has a different number of blocks as described below:

```
1512 cylinder = FBA blocks/8192
2512 cylinder = FBA blocks/8192
3512 cylinder = FBA blocks/8192
5121 cylinder = FBA blocks/960
5122 cylinder = FBA blocks/960
5123 cylinder = FBA blocks/960
```

The size definitions for FBA devices on Adabas control statements can specify the number of pseudo-cylinders or the number of Adabas blocks (RABNs).

Make sure that the starting block and the number of FBA blocks on the VSE EXTENT statement are on an FBA pseudo-cylinder boundary, which is based on the device as specified above for each Adabas file comprising the database:

- An SCSI pseudo-cylinder comprises 8,192 elements of 512 bytes each, or 4M per pseudo-cylinder. For example, an EXTENT entry for a ten cylinder SCSI device might consist of:

  ```
  // EXTENT SYS123,,,,8192,81920
  ```

- A virtual FBA pseudo-cylinder comprises 960 elements of 512 bytes each, or 480 K per pseudo-cylinder. For example, an EXTENT entry for a ten cylinder virtual FBA device might consist of:

  ```
  // EXTENT SYS123,,,,512,5120
  ```

ECKD Devices

Adabas supports ECKD DASD devices such as the IBM 3390 with the 3990 controller and ESCON channels.

During an open operation, ADAIOR determines which DASD device types are being used for the ASSO, DATA, WORK, SORT, and TEMP datasets. At that time, Adabas issues an informational message for each Adabas database component, where `type` is the component:

```
ADA164 ... FILE DDtype HAS BEEN OPENED IN ckd/eckd MODE - RABN SIZE rabn-size
```

Note: Software AG strongly recommends that you avoid mixing ECKD and CKD extents within a file, because the file will be opened only in CKD mode. Mixing extents could degrade performance when file I/O operations are performed.

Adding New Devices

Support for new device types that include user-defined block sizes can be implemented in ADAIOR by modifying one of the table of device-constant entries (TDCEs) reserved for this purpose.

A TDCE is X'40' bytes long and the first free TDCE can be identified by X'0000' in its first two bytes (TDCDT).

For all versions of Adabas prior to Version 6.2, the address of the first TDCE is at offset ADAIOR+ X'34'.

For Adabas Version 6.2, TDCE entries are in the ADAIOR CSECT TDCON: the first TDCE entry is at offset 0; the first free TDCE entry is at offset X'400'.

For Adabas Version 7.4, TDCE entries are in the ADAIOS CSECT TDCON, which corresponds to ESDID 1A8 in object module IOSVSE.OBJ. The first TDCE entry is at offset X'EF90' into IOSVSE.OBJ; the first free TDCE entry is at offset X'F550'.

This information is valuable when adding an additional TDCE entry, and when zapping the object module and relinking ADAIOS under VSE.

The z/VSE MSHP control statements to add a TDCE entry at the first free entry thus take the form:

```
// EXEC MSHP
CORRECT 9001-ADA-00-741 :AD99998
AFFECTS MODULE=IOSVSE,ESDID=1A8
ALTER F550 0000 : nnnn
ALTER F552 0000 : nnnn
.
. (etc.)
.
INVOLVES LINK=LNKIOR/*
```

- Information to be Zapped into the First Free ADAIOR TDCE
- General Rules for Defining Device Block Sizes
- Using 3480/3490 Tape Cartridge Compression (IDRC)

Information to be Zapped into the First Free ADAIOR TDCE

The information in the following tables must be zapped into the first free TDCE. The rules described in the section *General Rules for Defining Device Block Sizes* must be followed when changing the TDCE.

Label	Offset	Contents
TDCDT	00	Device type in unsigned decimal (X'3385'), must be numeric, and unique among all TDCEs.
TDCKSN	02	Constant set number: must be uniquely chosen from the values X'2B' or X'2E'.
TDCF	03	The flag bit must be set—TDCFCKD (X'40') for CKD devices, TDCFECKD (X'60') for ECKD devices or TDCFECKD (X'61') for ECKD, not user defined devices.
TDCDT1	04	(see note)
TDCDT2	05	(see note)
TDCDT3	06	(see note)
TDCDT4	07	(see note)
TDCMSBS	08	Refer to the TDCMSBS default table in *Maximum Sequential Block Size* in the Adabas z/OS installation instructions for more system- and device-related information.
TDCTPC	0A	Number of tracks per cylinder.
TDCCIPT	0C	Number of FBA blocks or PAM pages per track (if TDCFFBA is set).

Label	Offset	Contents
TDCBPCI	0E	Number of bytes per FBA block or PAM page (2048 if TDCFFBA is set).
TDCABPT	10	Number of Associator blocks per track.
TDCABS	12	Associator block size.
TDCACPB	14	Number of FBA blocks or PAM pages per Associator block (if TDCFFBA is set).
TDCDBPT	16	Number of Data Storage blocks per track.
TDCDBS	18	Data Storage block size.
TDCDCPB	1A	Number of FBA blocks or PAM pages per Data Storage block (if TDCFFBA is set).
TDCWBPT	1C	Number of Work blocks per track.
TDCWBS	1E	Work block size.
TDCWCPB	20	Number of FBA blocks or PAM pages per Work block (if TDCFFBA is set).
TDCTSBPT	22	Number of TEMP or SORT blocks per track
TDCTSBS	24	TEMP or SORT block size.
TDCTSCPB	26	Number of FBA blocks or PAM pages per TEMP or SORT block (if TDCFFBA is set).
TDCPBPT	28	Number of PLOG blocks per track.
TDCPBS	2A	PLOG block size.
TDCPCPB	2C	Number of FBA blocks or PAM pages per PLOG block (if TDCFFBA is set).
TDCCBPT	2E	Number of CLOG blocks per track.
TDCCBS	30	CLOG block size.
TDCCCPB	32	Number of FBA blocks or PAM pages per CLOG block (if TDCFFBA is set).

Note: One or more z/VSE codes for identifying the device type: PUB device type from PUBDEVTY (refer to the IBM MAPDEVTY macro).

In addition, the length of a sequential protection log block may have to be increased. This length is contained in the corresponding PTT entry in CSECT I_PTT of the load module ADAIOR. The address of the first PTT entry is contained in the fullword at ADAIOR+X'4C8'. PTT entries begin at offset 0 into CSECT I_PTT.

Each PTT entry is X'10' bytes long and has the structure given below:

Label	Offset	Contents
PTTPN	00	Program number
PTTFT	01	File type
PTTN	02	DD name characters 2 - 8
PTTF	08	Flags: OUT (X'80') output BSAM (X'40') BSAM BACK (X'20') read backwards JCL (X'10') BLKSIZE/LRECL/RECFM taken from DATADEF statement or label

Label	Offset	Contents
		UNDEF (X'04') undefined record format VAR (X'02') variable record format
-	09	Reserved
PTTMBS	0A	Maximum block size
-	0C	Reserved

The PTT entry for the sequential protection log can be identified by X'12F1' in its first two bytes.

General Rules for Defining Device Block Sizes

The following general rules must be followed when defining Adabas device block sizes:

- All block sizes must be multiples of 4.
- A single block cannot be split between tracks (that is, the block size must be less than or equal to the track size).

Block Rules for ASSO/DATA

The following rules are applicable for Associator and Data Storage:

- Associator block size must be greater than one-fourth the size of the largest FDT, and should be large enough to accept definitions in the various administrative blocks (RABN 1 - 30) and in the FCB;
- The block sizes for Associator and Data Storage should be a multiple of 256, less four bytes (for example, 1020) to save Adabas buffer pool space.
- The Associator and Data Storage block sizes must be at least 32 less than the sequential block size.
- Data Storage block size must be greater than: (maximum compressed record length + 10 + padding bytes).

Block Rule for WORK

The following rule is applicable for Work::

- The Work block size must be greater than either (maximum compressed record length + 110) or (Associator block size + 110), whichever is greater.

Block Rules for TEMP/SORT

The following rules are applicable for TEMP and SORT:

- Block sizes for TEMP and SORT must be greater than the block sizes for Data Storage.

- If ADAM direct addressing is used:

```
size > (maximum compressed record length + ADAM record length + 24);
size > 277 (maximum descriptor length + 24)
```

TEMP and SORT are generally read and written sequentially; therefore, the larger the TEMP/SORT block size, the better.

Block Rule for PLOG or SIBA

The following rules are applicable for PLOG and SIBA:

- The PLOG or SIBA block size must be greater than either (maximum compressed record length + 110) or (Associator block size + 110), whichever is greater.
- It is also recommended that PLOG/SIBA be defined larger than the largest Data Storage block size. This avoids increased I/O caused by splitting Data Storage blocks during online ADASAV operations.

  The block size (BLKSIZE) of a sequential file is determined as follows:

```
if PTTF(JCL) then BLKSIZE is taken from file assignment statement or label;
if PTTMBS > 0 then BLKSIZE = PTTMBS;
if PTTMBS = 0 then
if tape then BLKSIZE = 32760;
else BLKSIZE = TDCMSBS;
else if BLKSIZE in file assignment statement or label then use it;
if PTTF(OUT) then
if QBLKSIZE > 0 then BLKSIZE = QBLKSIZE;
if tape then BLKSIZE = 32760;
else BLKSIZE = TDCMSBS;
else error.

Note:
QBLKSIZE is an ADARUN parameter.
```

Using 3480/3490 Tape Cartridge Compression (IDRC)

The use of hardware compression (IDRC) is not recommended for protection log files. The ADARES BACKOUT function will run much longer when processing compressed data. Also, the BACKOUT function is not supported for compressed data.

User ZAPs to Change Logical Units

The specified zaps should be added to the module IORVSE / phase ADAIOR, not to the specified utility.

For Adabas version 7.4 or above, PPT entries are in the ADAIOR CSECT I_PTT. The first PTT entry is at offset 0 into CSECT I_PTT.

When zapping the object module and relinking ADAIOR, note that the ADAIOR CSECT I_PTT corresponds to ESDID 007 in object module IORVSE.OBJ. For Adabas 7.4, the offset of the CSECT I_PTT into IORVSE.OBJ is X'1000'.

Utility	File	Default SYS Number	PTT Offset	VER	REP
ADACDC	SIIN	SYS010	608	1A0A	1Axx
ADACMP	AUSBA	SYS012	08	820C	82xx
	EBAND	SYS010	18	180A	18xx
	FEHL	SYS014	28	820E	82xx
ADACNV	FILEA	SYS010	5E8	820A	82xx
ADAGEN	BAND	SYS010	38	820A	82xx
ADALOD	FILEA (OUTPUT)	SYS012	48	820C	82xx
	FILEB (INPUT)	SYS012	58	020C	02xx
	EBAND	SYS010	68	1A0A	1Axx
	ISN	SYS016	78	1A10	1Axx
	OLD	SYS014	88	820E	82xx
ADAMER	EBAND	SYS010	98	1A0A	1Axx
ADANUC	LOG	SYS012	A8	820C	82xx
	SIBA	SYS014	B8	C20E	C2xx
ADAORD	FILEA (OUTPUT)	SYS010	C8	820A	82xx
	FILEA (INPUT)	SYS010	D8	120A	12xx
ADAPLP	PLOG	SYS014	E8	1A0E	1Axx
ADARAI	OUT	SYS010	618	800A	80xx
ADAREP	SAVE	SYS010	F8	1A0A	1Axx
	PLOG	SYS011	108	1A0B	1Axx
ADARES	SIIN	SYS020	118	1A14	1Axx
	BACK	SYS020	128	2C14	1Cxx
	SIAUS1	SYS021	138	8215	82xx
	SIAUS2	SYS022	148	8216	82xx
ADASAV	SAVE1	SYS011	158	820B	82xx

Utility	File	Default SYS Number	PTT Offset	VER	REP
	SAVE2	SYS012	168	820C	82xx
	SAVE3	SYS013	178	820D	82xx
	SAVE4	SYS014	188	820E	82xx
	SAVE5	SYS015	198	820F	82xx
	SAVE6	SYS016	1A8	8210	82xx
	SAVE7	SYS017	1B8	8211	82xx
	SAVE8	SYS018	1C8	8212	82xx
	DUAL1	SYS021	1D8	8215	82xx
	DUAL2	SYS022	1E8	8216	82xx
	DUAL3	SYS023	1F8	8217	82xx
	DUAL4	SYS024	208	8218	82xx
	DUAL5	SYS025	218	8219	82xx
	DUAL6	SYS026	228	821A	82xx
	DUAL7	SYS027	238	821B	82xx
	DUAL8	SYS028	248	821C	82xx
	REST1	SYS011	258	1A0B	1Axx
	REST2	SYS012	268	120C	1Axx
	REST3	SYS013	278	120D	1Axx
	REST4	SYS014	288	120E	1Axx
	REST5	SYS015	298	120F	1Axx
	REST6	SYS016	2A8	1210	1Axx
	REST7	SYS017	2B8	1211	1Axx
	REST8	SYS018	2C8	1212	1Axx
	FULL	SYS030	2D8	1A1E	1Axx
	DEL1	SYS031	2E8	1A1F	1Axx
	DEL2	SYS032	2F8	1A20	1Axx
	DEL3	SYS033	308	1A21	1Axx
	DEL4	SYS034	318	1A22	1Axx
	DEL5	SYS035	328	1A23	1Axx
	DEL6	SYS036	338	1A24	1Axx
	DEL7	SYS037	348	1A25	1Axx
	DEL8	SYS038	358	1A26	1Axx
	PLOG	SYS010	368	1A0A	1Axx
ADASEL	EXPA1	SYS011	378	820B	82xx
	EXPA2	SYS012	388	820C	82xx
	EXPA3	SYS013	398	820D	82xx

Utility	File	Default SYS Number	PTT Offset	VER	REP
	EXPA4	SYS014	3A8	820E	82xx
	EXPA5	SYS015	3B8	820F	82xx
	EXPA6	SYS016	3C8	8210	82xx
	EXPA7	SYS017	3D8	8211	82xx
	EXPA8	SYS018	3E8	8212	82xx
	EXPA9	SYS019	3F8	8213	82xx
	EXPA10	SYS020	408	8214	82xx
	EXPA11	SYS021	418	8215	82xx
	EXPA12	SYS022	428	8216	82xx
	EXPA13	SYS023	438	8217	82xx
	EXPA14	SYS024	448	8218	82xx
	EXPA15	SYS025	458	8219	82xx
	EXPA16	SYS026	468	821A	82xx
	EXPA17	SYS027	478	821B	82xx
	EXPA18	SYS028	488	821C	82xx
	EXPA19	SYS029	498	821D	82xx
	EXPA20	SYS030	4A8	821E	82xx
	SIIN	SYS010	4B8	1A0A	1Axx
ADATRA	TRA	SYS019	4C8	820A	82xx
ADAULD	OUT1	SYS010	4D8	820A	820xx
	OUT2	SYS011	4E8	820B	820xx
	ISN	SYS012	4F8	820C	820xx
	SAVE	SYS013	508	1A0D	1Axx
	PLOG	SYS014	518	1A0E	1Axx
	FULL	SYS030	528	1A1E	1Axx
	DEL1	SYS031	538	1A1F	1Axx
	DEL2	SYS032	548	1A20	1Axx
	DEL3	SYS033	558	1A21	1Axx
	DEL4	SYS034	568	1A22	1Axx
	DEL5	SYS035	578	1A23	1Axx
	DEL6	SYS036	588	1A24	1Axx
	DEL7	SYS037	598	1A25	1Axx
	DEL8	SYS038	5A8	1A26	1Axx
ADAVAL	FEHL	SYS014	5B8	820E	820xx

7 Connecting UES-Enabled Databases

Overview

Prior to Adabas Version 7, Entire Net-Work converted all data for mainframe Adabas when necessary from ASCII to EBCDIC. Starting with Version 7, Adabas is delivered with its own data conversion capability called universal encoding support (UES). Entire Net-Work detects when it is connected to a target database that converts data and passes the data through to Adabas without converting it.

For Adabas Version 7.4, the link routines ADALNK and ADALCO are delivered with UES enabled. That is, LNKUES and the translation tables are linked in.

Note: The use of UES-enabled link routines is transparent to applications, including applications that do not require UES translation support: it is not necessary to disable UES support.

Load Modules

The load modules for ADALNK and ADALCO have been linked with LNKUES and the default translation tables.

LNKUES converts data in the Adabas buffers and byte-swaps, if necessary, depending on the data architecture of the caller.

The two standard translation tables are

- ASC2EBC: ASCII to EBCDIC translation; and
- EBC2ASC: EBCDIC to ASCII translation.

The Adabas translation table pair is provided in the section ***Translation Tables***.

Default or Customized Translation Tables

You may use the load modules with the default translation tables linked in, or you may prepare your own customized translation tables, re-assemble the tables, and link them with the LNKUES module that is delivered.

Notes:

1. It should only be necessary to modify these translation tables in the rare case that some country-specific character other than "A-Z a-z 0-9" must be used in the Additions 1 (user ID) or Additions 3 field of the control block.
2. The load module LNKUESL delivered with earlier levels of Adabas version 7 is no longer supplied since the link jobs now specify the LNKUES module and the translation tables separately.
3. The LNKUES module is functionally reentrant; however, it is not linked that way in the Adabas load library.
4. When linking the LNKUES load module and the translation tables, the linkage editor may produce warning messages concerning the reentrant or reusability status of the linked module. These warning messages can be ignored.

Source Modules

The ADALNK and ADALCO source modules have been coded to enable UES support by default when assembled:

- The &UES Boolean assembly variable is set to 1 by default.; the statement to set it to 0 has been commented out.
- The setting of the other Boolean variables and equates such as the SVC number and the database ID remain unchanged from earlier deliveries of the source modules.

Job Steps

Job library members ALNKLCO, ALNKLNK, and ALNKLNKR are set up to assemble and link the ADALCO, ADALNK, and ADALNKR modules, respectively, with the UES components in three steps:

- Assemble the link module into the Adabas load library.
- Assemble the two translation tables into the Adabas load library.
- Link the link module with LNKUES and the translation tables and put the resulting load module into a user load library.

Calling LNKUES

LNKUES is called only on ADALNK request (X'1C') and reply (X'20') calls if the first byte of the communication ID contains X'01' and the second byte does not have the EBCDIC (X'04') bit set.

- For requests, LNKUES receives control before UEXITB.
- For replies, LNKUES receives control after UEXITA.

Required Environment

The Adabas database must be UES-enabled. See *DBA Tasks* and the ADACMP and ADADEF utility sections in the *Adabas Utilities* documentation for more information.

Connection Possibilities

UES-enabled databases can be connected to machines with different architectures through Com-plete or Smarts, or through Entire Net-Work.

Connection Through Com-plete or Smarts

Adabas SQL Gateway (ACE) clients may not be strictly EBCDIC in an environment where databases are connected through Software AG's internal product Smarts (APS).

The relevant Adabas link routine ADALCO is UES-enabled by default. The sample jobstream to assemble the ADALCO module is ASMLCO.X.. Sample JCL to link the ADALCO.PHASE may be found in job LINKS.X.

The assembled and linked ADALCO (as delivered or with customized and reassembled translation tables) is placed in the Smarts sublib or a user sublib and made available in the LIBDEF chain for the job.

- Step 1: Assemble the ADALCO Module into the Adabas Sublibrary (SMA Job Number I070)
- Step 2: Assemble the Two Translation Tables into the Adabas Load Library (SMA Job Number I056)
- Step 3: Link the ADALCO.PHASE with the Translation Tables and LNKUES (SMA Job Number I088)

- Step 4: Make ADALCO Available to Smarts

Step 1: Assemble the ADALCO Module into the Adabas Sublibrary (SMA Job Number I070)

Modify the member ASMLCO.X to assemble ADALCO as described in the example below:

```
X $$ JOB JNM=ASMLCO,CLASS=0,DISP=D 000100
X $$ LST CLASS=A,DISP=D 000200
// JOB ASMLCO ASSEMBLE THE COM-PLETE/SMARTS LINK
* *****************************************************
* ASSEMBLE ADALCO AS LCOVSE.OBJ
* ******************************************************
*
* Customize this job as follows:
*
* Change the ”X $$” to ”* $$” on the power cards.
* Change ”$*” to”/*”.
* Change ”$&” to ”/&”.
* Change ”X EOJ” to ”* EOJ”.
* Customize the CLASS, DEST and other information as needed.
* Change vvvvvv to the volume for the punch work file.
* Change ssss to the extent location for the punch file.
* Change ttt to the size of the work file extent.
* Provide your PROC information for the Adabas PROC.
* Provide proper LIBDEF information for the search chain.
* Provide the proper sublib information on the LIBR parm.
*
* ******************************************************
// EXEC PROC=ADAV7LIB 000400
// OPTION DECK,NOLINK
// LIBDEF SOURCE,SEARCH=(ADALIB.SUBLIB,...),TEMP
// DLBL IJSYSPH,'PUNCH.WORK1',0
// EXTENT SYSPCH,vvvvvv,1,0,ssss,ttt
ASSGN SYSPCH,DISK,VOL=vvvvvv,SHR
// EXEC ASMA90,SIZE=(ASMA90,50K), X
PARM='EXIT(LIBEXIT(EDECKXIT(ORDER=AE)))'
COPY ADALCO
END
$*
CLOSE SYSPCH,PUNCH
* CATALOG LCOVSE
// DLBL IJSYSIN,'PUNCH.WORK1'
// EXTENT SYSIPT,vvvvvv
ASSGN SYSIPT,DISK,VOL=vvvvvv,SHR
// EXEC LIBR,PARM='AC S=ADALIB.SUBLIB;CATALOG LCOVSE.OBJ R=Y'
$*
CLOSE SYSIPT,READER
$&
// JOB RESET
* ******************************************************
* RESET SYSIPT AND SYSPCH
* *****************************************************
ASSGN SYSIPT,READER
ASSGN SYSPCH,PUNCH
$*
$&
X $$ EOJ
```

Step 2: Assemble the Two Translation Tables into the Adabas Load Library (SMA Job Number I056)

Assemble the ASCII to EBCDIC and EBCDIC to ASCII translation tables, either default or customized. The source modules are in the distribution source sublibrary as ASC2EBC.A and EBC2ASC.A.

Use the ASMLNK.X jobstream as a guide to prepare a jobstream for assembling and cataloging the ASC2EBC and EBC2ASC object modules.

Be sure to:

- Use either COPY ASC2EBC or COPY EBC2ASC as the input to the assembly step after the EXEC ASMA90 JCL statement.
- Specify the desired object module name on the `LIBR` parm, ASC2EBC for the ASCII to EBCDIC translation table or EBC2ASC for the EBCDIC to ASCII translation table.

Step 3: Link the ADALCO.PHASE with the Translation Tables and LNKUES (SMA Job Number I088)

Link the ADALCO, ASC2EBC, EBC2ASC and LNKUES object modules into a final ADALCO phase that is UES-enabled. Place this phase into a sublibrary which will be made available through a LIBDEF search chain at execution time. The LINKS.X job contains a step to do this. A segment of this JCL is:

```
PHASE ADALCO,*
MODE AMODE(31),RMODE(24)
INCLUDE LCOVSE
INCLUDE LNKUES
INCLUDE ASC2EBC
INCLUDE EBC2ASC
ENTRY ADABAS
// EXEC LNKEDT,PARM='MSHP'
```

Step 4: Make ADALCO Available to Smarts

The (re)linked ADALCO must be made available to Smarts. If you are calling Adabas version 7 and you do not have the correct LNKUES/ADALCO module, Adabas produces unexpected results: response code 022, 253, etc.

Additional Steps for Installing UES Support for Adabas

The following LIBR sublibraries are distributed with ADABAS for UES support:

```
SAGLIB.BTE421CS
SAGLIB.BTE421DS
SAGLIB.APS27102
SAGLIB.APS271
```

Note: In Version 7.4.2, the link routines come linked with UES support by default.

to install these libraries:

1. Create a VSE library for the BTE and APS libraries.

```
* $$ JOB JNM=CRUESL,CLASS=0,DISP=D,LDEST=(,XXXXXX)
* $$ LST CLASS=A,DISP=D
// JOB LIBRDEF
// ASSGN SYS005,DISK,VOL=vvvvvv,SHR
// DLBL DDECSOJ,'SAG.ECS.ADA74.LIB',2099/365,SD
// EXTENT SYS005,vvvvvv,1,0,ssss,tttt
// EXEC LIBR
DEFINE L=DDECSOJ R=Y
/*
/&
* $$ EOJ
```

2. Restore the BTE and APS libraries to this file. Refer to the *Report of Tape Creation* for the file positions on the distribution tape.

```
* $$ JOB JNM=RESTECS,DISP=D,CLASS=0,LDEST=(,xxxxxx)
* $$ LST DISP=D,CLASS=A
// JOB RESTECS
// ASSGN SYS005,DISK,VOL=vvvvvv,SHR
// DLBL DDECSOJ,'SAG.ECS.ADA74.LIB',2099/365,SD
// EXTENT SYS005,vvvvvv,1,0,ssss,tttt
// ASSGN SYS006,cuu
// MTC REW,SYS006
// EXEC LIBR,PARM='MSHP'
```

```
RESTORE SUBLIB=SAGLIB.BTE421CS:DDECSOJ.BTE421CS -
TAPE=SYS006 -
LIST=YES -
REPLACE=YES
/*
// MTC REW,SYS006
// ASSGN SYS006,UA
/&
* $$ EOJ
```

3 Repeat for BTE421DS and APS271 APS27102.

4 Modify the Adabas startup JCL, adding the UES env section after ADARUN parms:

```
ADARUN .....
ADARUN ....
/*
ENVIRONMENT_VARIABLES=/DDECSOJ/APS271/ENVVARS.P
/*
/&
* $$ EOJ
```

5 Reference the library where the libraries were restored in your Adabas proc:

```
// ASSGN SYS005,DISK,VOL=vvvvvv,SHR
// DLBL DDECSOJ,'SAG.ECS.ADA74.LIB',2099/365,SD
// EXTENT SYS005,vvvvvv,1,0,ssss,tttt
and add the libraries to the libdef chain:
(be sure APS27102 is referenced before APS271)
// EXTENT SYS005,vvvvvv,1,0,ssss,tttt
// LIBDEF PHASE,SEARCH=(SAGLIB.USRLIB,SAGLIB.ADA742... X
SAGLIB.AOS742,SAGLIB.ADE742,SAGLIB.ACF742,...X
DDECSOJ.BTE421CS,DDECSOJ.BTE421DS, X
DDECSOJ.APS27102,DDECSOJ.APS271)
// LIBDEF OBJ,SEARCH=(SAGLIB.USRLIB,SAGLIB.ADA742... X
SAGLIB.AOS742,SAGLIB.ADE742,SAGLIB.ACF742, X
DDECSOJ.BTE421CS,DDECSOJ.BTE421DS, X
DDECSOJ.APS27102,DDECSOJ.APS271)
// LIBDEF SOURCE,SEARCH=(SAGLIB.USRLIB,SAGLIB.ADA742,... X
SAGLIB.AOS742,SAGLIB.ADE742,SAGLIB.ACF742, X
DDECSOJ.BTE421CS,DDECSOJ.BTE421DS, X
DDECSOJ.APS27102,DDECSOJ.APS271)
// LIBDEF PHASE,CATALOG=SAGLIB.USRLIB
```

6 Modify the ENVVARS.P file, adding the following line in the APS271 library:

```
* This member contains Environment Variables used by SMARTS and
* SMARTS based applications.
*
ECSOBJDIR=FILE://DDECSOJ/BTE421CS
```

7 Run the ADADEF utility set `UES=YES`:

```
* $$ JOB JNM=ADADEF,CLASS=0,DISP=D,LDEST=(,xxxxxx)
* $$ LST CLASS=A,DISP=D
// JOB ADADEF EXECUTE THE ADABAS VERSION 7 UTILITY ***DEF***
// OPTION LOG,PARTDUMP
*
// EXEC PROC=ADALIB
// EXEC PROC=ADAFIL
*
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADADEF,MODE=SINGLE,SVC=svc,DEVICE=dddd,DBID=nnnn
/*
ADADEF MODIFY UES=YES
/*
/&
* $$ EOJ
```

8 Start the database.

You should see the following message:

```
ENTIRE CONVERSION SERVICES INITIALIZED
```

Connection Through Entire Net-Work

UES-enabled databases are connected through Software AG's Entire Net-Work (WCP) using the Adabas nonreentrant batch or TSO link routine ADALNK. The sample jobstream to assemble and link the nonreentrant ADALNK module is ASMLNK.X.

The assembled and linked nonreentrant, batch ADALNK (as delivered or with customized and reassembled translation tables) should be placed in the LIBDEF search chain with Entire Network.

- Step 1: Assemble the ADALNK Module into the Adabas Load Library (SMA Job Number I055)
- Step 2: Assemble the Two Translation Tables into the Adabas Load Library (SMA Job Number I056)
- Step 3: Link the Translation Tables and LNKUES into ADALNK (SMA Job Number I088)

- Step 4: Make ADALNK Available to Entire Net-Work

Step 1: Assemble the ADALNK Module into the Adabas Load Library (SMA Job Number I055)

Modify the member ASMLNK.X to assemble and link ADALNK, as follows:

```
* Change the "* $$" to "* $$" on the power cards.
* Change "$*" to"/*".
* Change "$&" to "/&".
* Change "X EOJ" to "* EOJ".
* Customize the CLASS, DEST and other information as needed.
* Change vvvvvv to the volume for the punch work file.
* Change ssss to the extent location for the punch file.
* Change ttt to the size of the work file extent.
* Provide your PROC information for the Adabas PROC.
* Provide proper LIBDEF information for the search chain.
* Provide the proper sublib information on the LIBR parm.

* $$ JOB JNM=ASMLNK,CLASS=0,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ASMLNK ASSEMBLE ADALNK CATALOG IT AS LNKVSE.OBJ
* *******************************************************
* ASSEMBLE ADALNK CATALOG IT AS LNKVSE.OBJ
* ********************************************************
* ********************************************************
// EXEC PROC=ADALIB
// OPTION DECK,NOLINK
// LIBDEF SOURCE,SEARCH=(ADALIB.SUBLIB,...),TEMP
// DLBL IJSYSPH,'PUNCH.WORK1',0
// EXTENT SYSPCH,vvvvvv,1,0,ssss,ttt
ASSGN SYSPCH,DISK,VOL=vvvvvv,SHR
// EXEC ASMA90,SIZE=(ASMA90,50K), X
PARM='EXIT(LIBEXIT(EDECKXIT(ORDER=AE)))'
COPY ADALNK
/*
CLOSE SYSPCH,PUNCH
* CATALOG LNKVSE.OBJ
// DLBL IJSYSIN,'PUNCH.WORK1'
// EXTENT SYSIPT,vvvvvv
ASSGN SYSIPT,DISK,VOL=vvvvvv,SHR
// EXEC LIBR,PARM='AC S=ADALIB.SUBLIB;CATALOG LNKVSE.OBJ R=Y'
/*
CLOSE SYSIPT,READER
/&
// JOB RESET
* ********************************************************
* RESET SYSIPT AND SYSPCH
* ********************************************************
ASSGN SYSIPT,READER
ASSGN SYSPCH,PUNCH
```

```
/*
/&
$$ EOJ
```

Step 2: Assemble the Two Translation Tables into the Adabas Load Library (SMA Job Number I056)

Sample VSE jobs to assemble and catalog the ASC2EBC and EBC2ASC translate table modules may be found in members ASMA2E.X and ASME2A.X in the distribution sublibrary. The jobs should be customized according to the same instructions as the ASMLNK.X job cited above.

If you prefer to use the same translation tables that are used in Entire Net-work, change the COPY statements in ASC2EBC and EBC2ASC from UES2ASC and UES2EBC to NW2ASC and NW2EBC, respectively. After modifying the translation tables, be sure to (re)assemble them and link them with the delivered LNKUES module.

The Entire Net-Work translation table pair is also provided in the section ***Translation Tables***.

Step 3: Link the Translation Tables and LNKUES into ADALNK (SMA Job Number I088)

Link the ADALNK, ASC2EBC, EBC2ASC, LNKUES, and other user exit modules into a final ADALNK module that is UES-enabled. Place this phase into a sublibrary which will be made available through a LIBDEF search chain at execution.

```
* $$ JOB JNM=ADALNKS,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADALNKS
// EXEC PROC=ADALIB <===== ADABAS PROCEDURE NAME
// OPTION CATAL
PHASE ADALNK,*,NOAUTO
MODE AMODE(31),RMODE(24)
INCLUDE LNKVSE
INCLUDE LNKUES
INCLUDE ASC2EBC
INCLUDE EBC2ASC
ENTRY ADABAS
// EXEC LNKEDT,PARM='MSHP'
/*
/&
* $$ EOJ
```

Step 4: Make ADALNK Available to Entire Net-Work

Make sure that the ADALNK phase is placed in a sublibrary available in the Entire Net-work job's LIBDEF search chain.

The (re)linked ADALNK must be made available to Entire Net-Work. If you are calling Adabas version 7 and you do not have the correct LNKUES/ADALNK module, Adabas produces unexpected results: response code 022, 253, etc.

8 Installing The AOS Demo Version

This section describes how to install the Adabas Online System (AOS) demo version. To install AOS on systems that use Software AG's System Maintenance Aid (SMA), refer to the section of this document describing installation of Adabas in your operating environment. For information about SMA, see the *System Maintenance Aid* documentation.

The AOS demo version requires Natural version 3.1 or above.

Note: To install the full version selectable unit AOS, see the *Adabas Online System* documentation.

AOS Demo Installation Procedure

To install the AOS demo version without the System Maintenance Aid

1. For a Com-plete or CICS environment, link the correct object module with the Natural TP nucleus.

   If a split Natural nucleus is to be installed, the AOSASM module must be linked to the shared portion of the nucleus and not to the thread portion.
2. Perform a Natural INPL.

   The tape containing the AOS demo version contains an INPL-formatted dataset in Natural 3.1. The programs for the AOS demo version are stored in library SYSAOS.
3. Load the ADA error messages using the Natural utility ERRLODUS.

   The error messages are stored in an ERRN-formatted dataset included on the tape.

   See the *Natural Utilities* documentation for information about the ERRLODUS utility.
4. Execute the AOS demo version by logging on to the application library SYSAOS and entering the command `DBMENU`.

Installing AOS with Natural Security

Natural Security must be installed before implementing Adabas Online System Security. See the *Adabas Security Manual* for more information. For information about installing Natural Security for use with AOS Security, see the *Natural Security Manual.*

Natural Security Version 3.1 or above includes the ability to automatically close all open databases when the Natural command mode's LOGON function of the AOS demo version is invoked.

Use the following procedure if Natural Security is installed in your environment.

1. Define at least the library SYSAOS to Natural Security

   Software AG recommends you define this library and any others you may define as protected.

2. Specify the startup program for SYSAOS as DBMENU

   Do not specify a startup program name for the other libraries.

Setting the AOS Demo Version Defaults

Parameters that control the operation of the AOS demo version can be set at installation time by changing the defaults in the Natural program AOSEX1. The table below lists the parameters and possible values. Default values are underlined:

Parameter	Valid Values / Default	Function
`AOS-END-MSG`	`Yes (Y) / No (N)`	Display the AOS demo version end-of-session message?
`AOS-LOGO`	`Yes (Y) / No (N)`	Display the AOS demo version logo?

`CPEXLIST`	`No (N): normal list`
	
	`Yes (Y): extended`
		Display extended checkpoint list?

`MAX-AC-IOS`	`0-999999 (150)`	AC read converter block threshold value
`NR-EXT`	`1, 2, 3, 4, 5`	Critical extent threshold for listing file
`STATINTV`	`1-9999 seconds (60)`	Statistics gathering interval

To change the defaults, you must edit the Natural AOSEX1 program and make the changes directly within the program listing in the defaults area, as shown by the following example:

```
DEFINE DATA PARAMETER USING ADVPUEX1
END-DEFINE
*
* SET THE DEFAULTS
*
AOS-END-MSG = 'Y' (Display end-of-session message)
AOS-LOGO = 'Y' (Online System logo display—set to 'N' for no logo display)
CPEXLIST = 'N' (Checkpoint list control: set to 'Y' for extended checkpoint list)
NR-EXT = 4 (Critical extent threshold: 1, 2, 3, 4 or 5)
```

```
MAX-AC-IOS = 150 (AC read converter block threshold)
STATINTV = 60 (Statistic gathering time interval:  range: 1 - 9999)
*
END
```

9 Installing The Recovery Aid (ADARAI)

This section describes how to install the Adabas Recovery Aid (ADARAI).

ADARAI Installation Overview

To install the Adabas Recovery Aid, it is necessary to:

- allocate the recovery log;
- customize the skeleton job streams for your installation (see the *Adabas Operations* documentation for more detailed information);
- update the necessary nucleus run/utility job control to include the Recovery Aid data definition statements;
- install the Adabas/ADARAI utility configuration; and
- run ADARAI PREPARE and a save operation to begin a logging generation.

ADARAI Installation Procedure

Except for customizing the skeleton job stream, the specific installation steps are as follows:

To install the Adabas Recovery Aid:

1. Allocate the recovery logs

   Define and format the RLOGR1 file.

   Use the ADAFRM RLOGFRM function to format the RLOGs.

2. Add data definition statements

   Add an RLOGR1 DLBL statement to the nucleus job stream and to any utilities that update or save the database and thus write to the RLOG files.

   Whenever these utilities are executed while ADARAI is active in the database (that is, after the PREPARE function has been executed), the RLOGR1 DLBL statement must be included.

The following utilities update the database and therefore write to the RLOG:

```
ADAORD (all STORE and REORDER functions)
ADALOD (all functions)
ADAINV (all functions)
ADARES REGENERATE/BACKOUT database
ADASAV RESTORE (all functions) and RESTPLOG
ADADEF NEWWORK
```

The following utilities save the database and therefore write to the RLOG:

```
ADASAV SAVE (all functions)
ADAORD RESTRUCTURE
ADAULD
```

The following utility functions have an impact on recovery and therefore write to the RLOG:

```
ADARES PLCOPY/COPY
ADASAV MERGE
```

Additionally, the Adabas nucleus writes to the RLOG during startup and termination. The nucleus also writes checkpoint information to the RLOG when ADADBS or Adabas Online System functions are processed, ensuring these events are known to ADARAI for recovery processing.

3 Install ADARAI on the database.

Execute the ADARAI PREPARE function. ADARAI PREPARE updates the ASSO GCB to indicate that ADARAI is installed. It also creates a control record on the RLOG file with necessary ADARAI information (number of generations, RLOG size, etc.).

4 Create the first ADARAI generation.

Execute ADASAV SAVE (database) to start the logging of RLOG information. See the *Adabas Utilities* documentation for more information.

Once ADARAI is active in the database, protection logging must always be used.

10 Adabas Dump Formatting Tool (ADAFDP)

This section describes the use of the Adabas dump formatting tool ADAFDP.

ADAFDP Function

ADAFDP is the address space dump formatting module. During abnormal shutdown of the Adabas nucleus, this module receives control to format and display information that should help you analyze the reason for the error.

During a nucleus shutdown, ADAMPM determines the shutdown reason. If the reason is abnormal termination, ADAMPM loads the ADAFDP module into the address space prior to the 20 call to the Adabas SVC. ADAFDP subsequently receives control to format nucleus information.

If ADAFDP cannot be loaded, message ADAF03 is written to the console and abnormal shutdown continues.

ADAFDP Output

Much of the information formatted by ADAFDP is self-explanatory. However, because the type and amount of information depends on the shutdown situation, a summary of ADAFDP output is provided in this section.

ADAFDP Messages

Message	Description
ADAH51 / ADAH52	The message is displayed on the console and written to DDPRINT at the point where the format begins and terminates.
ADAMPM ABEND CODE and PSW	If an Abend code and program status word (PSW) were saved in ADAMPM by the Adabas ESTAE, ADAFDP displays these. In addition, ADAFDP determines the module whose entry point best fits the PSW and calculates the offset within that module. If the ADAMPM abend code and PSW are zero, ADAFDP does not format this information.
ADABAS MODULE LOCATIONS	ADAFDP formats and displays the location of each of the Adabas nucleus modules resident in the address space.
ADDRESS LOCATIONS FOR USER EXITS	ADAFDP formats and displays the location of any user exit loaded with the Adabas nucleus.

Message	Description
ADDRESS LOCATIONS FOR HYPEREXITS	ADAFDP formats and displays the location of any hyperexit loaded with the Adabas nucleus. Hyperexits 10-31 are displayed as A-U, respectively.
ADANC0 STANDARD REGISTER SAVE AREA	Registers 0-7/8-F, which are saved in ADANC0. ADAFDP determines if any of these registers contains an address that points at a nucleus pool in storage. If yes, ADAFDP indicates which pool and snaps storage at that address. If the register is 12 and it points to a user thread, ADAFDP snaps the entire thread.
ADANC0 ABEND SAVE REGISTERS	Registers 0-7/8-F, which are saved in ADANC0 as a result of a user abend. ADAFDP determines if any of these saved registers contains an address that points at a nucleus pool in storage. If yes, ADAFDP indicates which pool and snaps storage at that location. If the saved register is 12 and it points to a user thread, ADAFDP snaps the entire thread.
ADAMPM SAVE REGISTERS	Registers 0-7/8-F, which were saved in ADAMPM by the Adabas ESTAE. These are the same registers displayed with the ADAM99 message. ADAFDP determines if any of these saved registers contains an address that points within a nucleus pool in storage. If yes, ADAFDP indicates which pool and snaps storage at that location.
BEGIN / ENDING ADDRESSES OF POOLS / TABLES	ADAFDP determines begin/ending address locations for pools and tables for the Adabas nucleus. These addresses are presented for easy location in the actual dump. See ***Pool Abbreviations*** for more information.
ADABAS THREADS	ADAFDP formats the physical threads including threads 0, -1, and -2. The number of lines depends on the value of `NT`. The thread that was active at the time of the abnormal termination (if any) is marked by a pointer "-->".
USER THREADS	For any of the threads -2 to `NT` that had assigned work to perform, ADAFDP formats and displays information about the status of that thread. See ***User Threads*** for more information:
FOLLOWING COMMANDS WERE FOUND IN THE CMD QUEUE	ADAFDP scans the command queue and formats information for any command found in the queue. See ***Command Information*** for more information.
POOL INTEGRITY CHECK	ADAFDP check the integrity of several pools within the Adabas nucleus address space. If an error is detected within that pool, ADAFDP indicates which pool and what type of error was encountered. In addition, ADAFDP snaps storage at the location where the error was detected.
FOLLOWING RABNS / FILES ACTIVE IN BUFFER POOL	ADAFDP scans the buffer pool header for RABNs that were active or being updated. See ***RABN Information*** for more information.
ADAIOR REGS FOUND AT OFFSET X'080'	Registers 0-7/8-F found saved in ADAIOR at this offset. If ADAFDP determines that any of these register values is pointing within an Adabas pool, it snaps storage at that location.
ADAIOR REGS FOUND AT OFFSET X'0C0'	Registers 0-7/8-F found saved in ADAIOR at this offset. If ADAFDP determines that any of these register values is pointing within an Adabas pool, it snaps storage at that location.
ICCB POINTED FROM X'A0' IN IOR	The ICCB address to which this offset in ADAIOR points.

Message	Description
ADAI22 ADAIOR TRACE TABLE	Format of ADAIOR trace table; same as that found with the ADAM99 message.

Pool Abbreviations

Pool Abbreviation	Description
LOG	Log area
OPR	Adabas nucleus operator command processing area
CQ	Address of the command queue, which is formatted later by ADAFDP
ICQ	Internal command queue
TT	Thread table
IA1	Software AG internal area 1
SFT	Session file table
FU	File usage table
FUP	File update table
IOT	I/O table for asynchronous buffer flushing
PL2	PLOG area for asynchronous buffer flushing
PET	Table of posted ETs
TPT	Tpost
TPL	Tplatz
UQP	Unique descriptor pool
UHQ	Upper hold queue
HQ	Hold queue
UUQ	Upper user queue
UQ	User queue
FP	Format pool
FHF	File HILF element
PA	Protection area
TBI	Table of ISNs
TBQ	Table of sequential searches
WK3	Work part 3 space allocation table
IA2	Software AG internal area 2
WK2	Work part 2 space allocation table
VOL	VOLSER table
WIO	Work block I/O area
FST	Free space table work area

Pool Abbreviation	Description
UT	User threads
WP	Work pool
AW2	Work block asynchronous I/O area
IOP	I/O pool related to asynchronous buffer flush
IU2	Buffer pool importance header upper 2
IU1	Buffer pool importance header upper 1
BU2	Buffer pool upper header 2
BU1	Buffer pool upper header 1
BH	Address location of the buffer pool header, information from the buffer pool header is formatted later by ADAFDP
BP	Address location of the physical start of the buffer pool

User Threads

Information	Description
Thread Number	-2 to `NT`
Status	Indicates the current status of the thread: ■ *Active*: the currently active thread ■ In Use: thread has been assigned work ■ Waiting For I/O: waiting for a block not in buffer pool ■ Waiting For RABN: waiting for a RABN already in use ■ Waiting For Work-2 Area Block: similar to waiting for I/O ■ Waiting Workpool Space: provides number of bytes in decimal ■ Ready To Run: waiting to be selected for execution
CMD	The Adabas command being executed
Response Code	Response code (if any)
File Number	File number for this command
ISN	Internal sequence number for this command
Sub. Rsp	Subroutine response code (if any)
Last RABN for I/O	Last RABN required by command processing, in decimal
Type	Last RABN type (A - ASSO, D - DATA)
CQE Addr	Command queue element address for this command
User Jobname	Job name for user who executed this command
ITID	Internal Adabas ID for user who executed this command
User	User ID for user who executed this command

Information	Description
Unique global ID	28-byte ID for user who owns this command
Buffer Addresses	buffer addresses for: control block, format buffer, search buffer, value buffer, ISN buffer
Buffer Lengths	FL: format buffer length RL: record buffer length SL: search buffer length VL: value buffer length IL: ISN buffer length
Snap Thread	The first 144 bytes of the user thread are snapped

Command Information

Information	Description
CQE Address	The address location of this CQE
F	Command queue flag bytes: ■ First Byte: General Purpose Flag ■ X’80’: User buffers in service partition, region, address space ■ X’40’: ET command waiting for 12 call ■ X’20’: Waiting for 16 call ■ X’10’: 16 call required ■ X’08’: Attached buffer ■ X’04’: Attached buffer required ■ X’02’: X-memory lock held (MVS only) ■ Second Byte: Selection Flag ■ X’80’: In process ■ X’40’: Ready to be selected ■ X’20’: Search for UQE done ■ X’10’: UQE found ■ X’08’: Not selectable during BSS=x’80’ status ■ X’04’: Not selectable during ET-SYNC ■ X’02’: Waiting for space ■ X’01’: Waiting for ISN in HQ
CMD	The command type
File Number	The file number for this command
Job Name	Job name for the user
Addr User	UQE Address of users UQE, if searched for and found
Addr User ASCB	Address location of user’s ASCB

Information	Description
Addr ECB	Address location of user's ECB (in user's address space)
Addr User UB	Address of users UB (in user's address space)
Addr User PAL	Address location of user's parameter address list
CQE ACA	ACA field of CQE.
CQE RQST	RQST field of CQE
Abuf/Pal	Address of the attached buffer/parameter address list (PAL) for CMD
Comm Id	28-byte unique user ID for this command

RABN Information

Information	Description
RABN Number	The RABN number in decimal
Type	Type of block (A - ASSO, D - DATA)
Flag	BP header element flag byte: ■ AKZ X'40': Active indicator ■ UKZ X'20': Update indicator ■ RKZ X'10': Read indicator ■ XKZ X'04': Access is waiting for block ■ YKZ X'02': Update is waiting for block ■ SKZ X'01': Write indicator
File	File number that owns this block
Address	Address location of block in storage.

11 Maintaining A Separate Test Environment

This section describes a method to set up a temporary test copy of phases updated by a program fix. The method described is intended as an example. Its relevance depends on the installation standards you use for library maintenance.

The example scenario uses MSHP in a single z/VSE machine to control both the standard production Adabas library and an additional testing library or sublibrary used to validate recently applied program fixes.

After restoring the standard Adabas library and defining it to MSHP, an additional test library or sublibrary can be defined.

Object modules can then be copied from the standard library as required, and controlled with MSHP using a different VSE system history file. Using the same component ID as for the standard environment (9001-ADA-00-*vrs*) ensures that the ZAP source remains common to both environments.

The test version of a phase is then invoked by placing the test library or sublibrary at the head of the LIBDEF PHASE search chain.

The setup jobs required to implement this environment are described in detail below. Note that the first three steps form part of the standard installation process.

to setup the separate test environment:

1 Define standard Adabas library.

   For a sample job, see the section *Installing the Adabas Release Tape*.

2 Restore standard Adabas library.

   For a sample job, see the section *Installing the Adabas Release Tape*.

3 Define standard Adabas to MSHP.

Note: This job uses the history file identified by the IJSYSHF label in the VSE standard label area.

```
// EXEC MSHP
ARCHIVE ADAvrs
COMPRISES 9001-ADA-00
RESOLVES 'SOFTWARE AG - ADABAS Vvrs'
ARCHIVE 9001-ADA-00-vrs
RESIDENCE PRODUCT=ADAvrs -
PRODUCTION=SAGLIB.adannn -
GENERATION=SAGLIB.adannn
/*
—where
vrs is the Adabas version, revision, and system maintenance (SM) level
adannn is the sublibrary name for standard Adabas
```

4 Create test sublibrary and copy object modules to it.

```
// DLBL SAGLIB,'adabas.vnnn.library'
// EXTENT SYS010
// ASSGN SYS010,DISK,VOL=volser,SHR
// EXEC LIBR
DEFINE SUBLIB=SAGLIB.adatst
CONNECT SAGLIB.adannn:SAGLIB.adatst
COPY *.OBJ LIST=Y REPLACE=Y
/*
—where

adabas.vnnn.library is the physical name of the standard Adabas library
volser is the volume on which library resides
adannn is the sublibrary name for standard Adabas
adatst is the sublibrary name for testing Adabas
```

5 Create additional system history file for test environment and define test Adabas to it.

```
// ASSGN SYS020,DISK,VOL=volhis,SHR
// EXEC MSHP
CREATE HISTORY SYSTEM
DEFINE HISTORY SYSTEM EXTENT=start:numtrks -
UNIT=SYS020 -
ID='sag.test.system.history.file'
ARCHIVE ADAvrs
COMPRISES 9001-ADA-00
RESOLVES 'SOFTWARE AG - ADABAS Vvrs'
ARCHIVE 9001-ADA-00-vrs
RESIDENCE PRODUCT=ADAvrs -
PRODUCTION=SAGLIB.adatst -
GENERATION=SAGLIB.adatst
/*
—where
```

```
volhis is the volume on which test system history file resides
start is the start of extent on which test system history file resides
numtrks is the length of extent on which test system history file resides
sag.test.system.history.file is the physical name of test system history file
vrs is the Adabas version, revision, and system maintenance level
adatst is the sublibrary name for testing Adabas
```

6 Apply zap to test environment.

```
// DLBL IJSYSHF,'sag.test.system.history.file'
// EXTENT SYS020,,,,start,numtrks
// ASSGN SYS020,DISK,VOL=volhis,SHR
// DLBL SAGLIB,'adabas.vnnn.library'
// EXTENT SYS010
// ASSGN SYS010,DISK,VOL=volser,SHR
// EXEC MSHP
CORRECT 9001-ADA-00-vrs : ADnnnnn
AFFECTS MODULE=modname
ALTER offset hexold : hexnew
INVOLVES LINK=lnkname
/*
—where
sag.test.system.history.file is the physical name of test system history file
start is the start of extent on which test system history file resides
numtrks is the length of extent on which test system history file resides
volhis is the volume on which test system history file resides
adabas.vnnn.library is the physical name of the standard Adabas library
volser is the volume on which library resides
vrs is the Adabas version, revision, and system maintenance level
nnnnn is the Adabas fix number
modname is the Adabas object module to be zapped and then relinked
offset is the hexadecimal offset to the beginning of the zap
hexold is the verify data for the zap
hexnew is the replace data for the zap
lnkname is the link book for the phase affected
```

7 Invoke updated test phase.

```
// DLBL SAGLIB,'adabas.vnnn.library'
// EXTENT SYS010
// ASSGN SYS010,DISK,VOL=volser,SHR
// LIBDEF PHASE,SEARCH=(SAGLIB.adatst,SAGLIB.adannn,...)
...
—where
adabas.vnnn.library is the physical name of the standard Adabas library
volser is the volume on which library resides
adatst is the sublibrary name for testing Adabas
adannn is the sublibrary name for standard Adabas
```

8 Apply zap to standard environment.

Note: This job uses the history file identified by the IJSYSHF label in the VSE standard label area.

```
// DLBL SAGLIB,'adabas.vnnn.library'
// EXTENT SYS010
// ASSGN SYS010,DISK,VOL=volser,SHR
// EXEC MSHP
CORRECT 9001-ADA-00-vrs : ADnnnnn
AFFECTS MODULE=modname
ALTER offset hexold : hexnew
INVOLVES LINK=lnkname
/*
—where
adabas.vnnn.library is the physical name of the standard Adabas library
volser is the volume on which library resides
vrs is the Adabas version, revision, and system maintenance (SM) level
nnnnn is the Adabas fix number
modname is the Adabas object module to be zapped and then relinked
offset is the hexadecimal offset to the beginning of the zap
hexold is the verify data for the zap
hexnew is the replace data for the zap
lnkname is the link book for the phase affected
```

9 Invoke standard phase.

```
// DLBL SAGLIB,'adabas.vnnn.library'
// EXTENT SYS010
// ASSGN SYS010,DISK,VOL=volser,SHR
// LIBDEF PHASE,SEARCH=(SAGLIB.adannn,...)
...
—where
adabas.vnnn.library is the physical name of the standard Adabas library
volser is the volume on which library resides
adannn is the sublibrary name for standard Adabas
```

12 Translation Tables

This section describes the translation tables which are supplied by Adabas.

Adabas EBCDIC to ASCII and ASCII to EBCDIC

```
cUES2ASC DS OF
c* .0.1.2.3.4.5.6.7.8.9.A.B.C.D.E.F
c DC x’000102033F093F7F3F3F3F0B0C0D0E0F’ 0.
c DC x’101112133F3F083F18193F3F3F1D3F1F’ 1.
c DC x’3F3F1C3F3F0A171B3F3F3F3F3F050607’ 2.
c DC x’3F3F163F3F1E3F043F3F3F3F14153F1A’ 3.
c DC x’203F3F3F3F3F3F3F3F3F3F2E3C282B3F’ 4.
c DC x’263F3F3F3F3F3F3F3F3F21242A293B5E’ 5.
c DC x’2D2F3F3F3F3F3F3F3F3F7C2C255F3E3F’ 6.
c DC x’3F3F3F3F3F3F3F3F3F603A2340273D22’ 7.
c DC x’3F616263646566676869 3F3F3F3F3F3F’ 8.
c DC x’3F6A6B6C6D6E6F7071723F3F3F3F3F3F’ 9.
c DC x’3F7E737475767778797A3F3F3F5B3F3F’ A.
c DC x’3F3F3F3F3F3F3F3F3F3F3F3F3F5D3F3F’ B.
c DC x’7B414243444546474849 3F3F3F3F3F3F’ C.
c DC x’7D4A4B4C4D4E4F5051523F3F3F3F3F3F’ D.
c DC x’5C3F535455565758595A3F3F3F3F3F3F’ E.
c DC x’303132333435363738393F3F3F3F3F3F’ F.
c* .0.1.2.3.4.5.6.7.8.9.A.B.C.D.E.F
END

cUES2EBC DS OF
c* .0.1.2.3.4.5.6.7.8.9.A.B.C.D.E.F
c DC x’00010203372D2E2F1605250B0C0D0E0F’ 0.
c DC x’101112133C3D322618193F27221D351F’ 1.
c DC x’405A7F7B5B6C507D4D5D5C4E6B604B61’ 2.
c DC x’F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F’ 3.
c DC x’7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6’ 4.
c DC x’D7D8D9E2E3E4E5E6E7E8E9ADE0BD5F6D’ 5.
c DC x’79818283848586878889919293949596’ 6.
c DC x’979899A2A3A4A5A6A7A8A9C06AD0A107’ 7.
c DC x’6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F’ 8.
c DC x’6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F’ 9.
c DC x’6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F’ A.
c DC x’6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F’ B.
c DC x’6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F’ C.
c DC x’6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F’ D.
c DC x’6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F’ E.
c DC x’6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F’ F.
c* .0.1.2.3.4.5.6.7.8.9.A.B.C.D.E.F
END
```

Entire Net-Work EBCDIC to ASCII and ASCII to EBCDIC

```
NW2ASC DS OF
* .0.1.2.3.4.5.6.7.8.9.A.B.C.D.E.F
DC X'000102030405060708090A0B0C0D0E0F' 0.
DC X'101112131415161718191A1B1C1D1E1F' 1.
DC X'00000000000000000000000000000000' 2.
DC X'00000000000000000000000000000000' 3.
DC X'200000000000000000005B2E3C282B5D' 4.
DC X'26000000000000000000021242A293B5E' 5.
DC X'2D2F000000000000000007C2C255F3E3F' 6.
DC X'000000000000000000603A2340273D22' 7.
DC X'00616263646566676869000000000000' 8.
DC X'006A6B6C6D6E6F707172000000000000' 9.
DC X'007E737475767778797A00005B000000' A.
DC X'000000000000000000000000005D0000' B.
DC X'7B414243444546474849000000000000' C.
DC X'7D4A4B4C4D4E4F505152000000000000' D.
DC X'5C7E535455565758595A000000000000' E.
DC X'303132333435363738397C00000000FF' F.
* .0.1.2.3.4.5.6.7.8.9.A.B.C.D.E.F

NW2EBC DS OF
* .0.1.2.3.4.5.6.7.8.9.A.B.C.D.E.F
DC X'000102030405060708090A0B0C0D0E0F' 0.
DC X'101112131415161718191A1B1C1D1E1F' 1.
DC X'405A7F7B5B6C507D4D5D5C4E6B604B61' 2.
DC X'F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F' 3.
DC X'7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6' 4.
DC X'D7D8D9E2E3E4E5E6E7E8E9ADE0BD5F6D' 5.
DC X'79818283848586878889919293949596' 6.
DC X'979899A2A3A4A5A6A7A8A9C06AD0A100' 7.
DC X'00000000000000000000000000000000' 8.
DC X'00000000000000000000000000000000' 9.
DC X'00000000000000000000000000000000' A.
DC X'00000000000000000000000000000000' B.
DC X'00000000000000000000000000000000' C.
DC X'00000000000000000000000000000000' D.
DC X'00000000000000000000000000000000' E.
DC X'000000000000000000000000000000FF' F.
* .0.1.2.3.4.5.6.7.8.9.A.B.C.D.E.F
END
```

13 Glossary of Installation-Related Terms

Adalink

The teleprocessing-monitor-dependent interface module that connects the application/user to Adabas. The actual module name depends on the environment being used; for example, the module name for linking to a batch or TSO program is ADALNK, and for CICS, the module name is ADALNC. The term “Adalink” refers to the module appropriate for the given environment.

address converter

Adabas stores each database record in a Data Storage block having a relative Adabas block number (RABN). This RABN location is kept in a table called the address converter. The address converters, one for each database file, are stored in the Associator. Address converter entries are in ISN order (that is, the first entry tells the RABN location of data for ISN 1, the 15th entry holds the RABN location of data for ISN 15, and so on).

address space

The storage area assigned to a program task/work unit. In MVS, an address space is a region; in VSE, a partition; and in BS2000, a task. In this documentation, the term region is used as a synonym for partition and task.

communicator

A routine for communicating between operating systems, making remote targets accessible. Entire Net-work is a communicator.

database administrator

Controls and manages the database resources. Tasks include defining database distribution structure and resources, creating and maintaining programming and operation standards, ensuring high performance, resolving user problems, user training, controlling database access and security, and planning for growth and the integration of new database resource applications and system upgrades. Also known as the database analyst.

ID

An abbreviation of "target ID", a unique identifier used for directing Adabas calls to their targets.

ID table

A reference data list maintained for all active targets within the boundaries of one operating system. The ID table is located in commonly addressable storage.

IIBS

The "isolated ID bit string", a 256-bit (32-byte) string contained in the ID table header. Each bit corresponds in ascending order to a logical ID. If the bit has the value 1, the corresponding ID is isolated.

isolated ID

The ID of an isolated target, which can be specified by the user as a logical ID. An isolated ID must be greater than zero and less than 256. The isolated ID is interpreted as a physical ID for addressing the target.

isolated target

A target called directly by a user.

logical ID

A user's identifier of target(s) to which a message is directed. It must be greater than 0 and less than 256 (either explicitly or implicitly, the content of the first byte of ACBFNR is a logical ID).

non-DB target

A target that is not an Adabas nucleus. Access and X-COM are non-DB targets.

physical ID

The identifier of a target. It must be greater than 0 and less than 65,536. A database ID (DBID) is a physical ID.

pseudo-cylinder

The logical cylinder on an fixed-block-addressed (FBA) device that has no actual DASD cylinder.

reset

A flag bit is said to be reset when it contains 0.

router

A central routine for communication within the boundaries of one operating system. The routine is called by users with Adalink routines, and by targets with ADAMPM. The router's main purpose is to transfer information between the Adalink and Adabas. The router also maintains the ID table. VM/ESA, z/VM, and BS2000 environments divide router functions among Adalink or other Adabas functions. The Adabas SVCs in OS/390, z/OS, and z/VSE are examples of routers.

service

A processor of Adabas calls and issuer of replies. An Adabas nucleus is an example of a service (see also target).

set

A flag bit is said to be set when it contains 1.

target

A receiver of Adabas calls. A target maintains a command queue, and communicates with routers using ADAMPM. A target is also classified as a service (see definition). The Adabas nucleus is a target.

user

A batch or online application program that generates Adabas calls and uses an Adalink for communication.

Index