Installation Procedure

This section describes the procedure for Adabas installation in z/VSE environments.

- Installation Checklist
- Contents of the Release Tape
- Preparing to Install Adabas
- Initializing the Adabas Communication Environment
- Installing the Adabas Database
- Migrating an Existing Database
- Logical Unit Requirements
- Job Exit Utility
- Acquiring Storage for the ID Table
- Acquiring Storage for the IIBS Table
- SVC Work Areas
- Displaying Storage Allocation Totals
- Calls from Other Partitions
- Dummy Sequential Files
- Backward Processing of Tapes and Cartridges
- Applying Zaps (Fixes)
- Adabas 8 Adalink Considerations
- Setting Defaults in ADARUN

Installation Checklist

The following is an overview of the steps for installing Adabas on a 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*.

Step	Description	Additional Information
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 *Prepare the Installation Sample JCS for Editing*
8	Customize and run job ADAIOOAL to link the Adabas options table for installation customization.	See section *Modify, Assemble, and Link the Adabas Options Table*
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	See section *Catalog Procedures for Defining Libraries and the Database*

Step	Description	Additional Information
10	Customize and run ADAFRM to allocate and format the Adabas database.	Steps 10-19 require changes to the setup definitions as described in section *Database Installation Steps*
11	Customize and run ADADEF to define global database characteristics.	
12	Customize and run ADALODE, ADALODV, and ADALODM to load the demo files.	
13	Install the product license file.	
14	Customize and run ADANUC to start the Adabas nucleus to test Adabas communications.	
15	Customize and run ADAREP in MULTI mode with the CPLIST parameter to test Adabas partition communication.	
16	Customize and run ADAINPL to load the Adabas Online System, if used.	
17	Terminate the Adabas nucleus.	
18	Customize and run ADASAV to back up the database.	
19	Customize and run DEFAULTS to insert the ADARUN defaults with the ZAP utility.	
20	Install the required TP link routines for Adabas	See section *Installing Adabas With TP Monitors*.

Contents of the Release Tape

The following table describes most of the libraries included on the release tape. Once you have unloaded the libraries from the tape, you can change these names as required by your site, but the following lists the names that are delivered when you purchase Adabas for z/VSE environments.

Library Name	Description
ADA*vrs*.EMPL	The Employees demo file, containing dummy employee data you can use for testing Adabas. The *vrs* in the library name represents the *version* of Adabas.
ADA*vrs*.ERRN	Error messages for the Adabas Triggers and Stored Procedures Facility. These messages can be viewed using the Natural SYSERR utility. The *vrs* in the library name represents the *version* of Adabas.

Library Name	Description
ADA*vrs*.INPL	The code for Adabas Online System, Adabas Caching Facility, Triggers and Stored Procedures Facility, and various add-on demo products. The *vrs* in the library name represents the *version* of Adabas.
ADA*vrs*.LC*nn*	The Adabas library containing character encoding members to support various languages and Unicode. The *nn* letters in the library name represents a number from "00" to "99", assigned by Software AG. The *vrs* in the library name represents the *version* of Adabas.
ADA*vrs*.LIBR	The library for Adabas. The *vrs* in the library name represent the *version* of Adabas.
ADA*vrs*.MISC	The Miscellaneous demo file, containing dummy miscellaneous data you can use for testing Adabas. The *vrs* in the library name represents the *version* of Adabas.
ADA*vrs*.PERL	The LOB demo file storing the LOB data referenced by the Personnel demo file. The *vrs* in the library name represent the *version* of Adabas.
ADA*vrs*.PERS	The Personnel demo file, containing dummy personnel data you can use for testing Adabas. This demo file includes fields that make use of the extended and expanded features of Adabas 8, include large object (LOB) fields. The *vrs* in the library name represents the *version* of Adabas. **Note:** The Personnel demo file must be installed on a UES-enabled database because it includes wide-character format (W) fields.
ADA*vrs*.TZ00	The time zone library for Adabas. The *vrs* in the library name represents the *version* of Adabas. Adabas bases its time zone library on the time zones defined in the public domain tz database, also know as the *zoneinfo* or *Olson* database. For a complete list of the time zones supported by Adabas in any given release, refer to the TZINFO member in this Adabas library.
ADA*vrs*.VEHI	The Vehicles demo file, containing dummy vehicle data you can use for testing Adabas. The *vrs* in the library name represents the *version* of Adabas.
APS*vrs*.L018	A Software AG internal library. The *vrs* in the library name represents the *version* of the internal library code, which is not necessarily the same as the version of Adabas.
APS*vrs*.LIBR	A Software AG internal library. The *vrs* in the library name represents the *version* of the internal library code, which is not necessarily the same as the version of Adabas.

Library Name	Description
MLC*vrs*.LIBJ	The sample job library for Software AG's common mainframe license check software. The *vrs* in the library name represents the *version* of the license check software, which is not necessarily the same as the version of Adabas.
MLC*vrs*.LIBR	The load library for Software AG's common mainframe license check software. The *vrs* in the library name represents the *version* of the license check software, which is not necessarily the same as the version of Adabas.
WAL*vrs*.LIBR	The library for Adabas components shared by Adabas and other Software AG products, such as Entire Net-Work. The *vrs* in the library name represents the *version* of Adabas.
WCA*vrs*.LIBR	The library for Entire Net-Work Administration, used by some of the Adabas add-on products. The *vrs* in the library name represents the *version* of Entire Net-Work Administration.

Adabas is shipped with the code for Entire Net-Work Client (open systems software) and Entire Net-Work Administration (mainframe software). Entire Net-Work Client and Entire Net-Work Administration are Software AG middleware packages used for communication between Adabas or Event Replicator Servers on the mainframe and open systems software packages such as Adabas Manager (including the Adabas Manager demo) or Event Replicator Administration. Entire Net-Work Administration is a limited version of Entire Net-Work for mainframes and includes the Simple Connection Line Driver.

Note:
Entire Net-Work Client requires a license key. A limited license is shipped with your Adabas software to support the Adabas Manager demo. If you purchase a full version of Adabas Manager, you will need a full license of Entire Net-Work Client.

If appropriate Entire Net-Work mainframe and client products are not already installed on your system, install Entire Net-Work Administration on the mainframe and Entire Net-Work Clienton the client side. For complete information on these products, read the Entire Net-Work Administration documentation and Entire Net-Work Client Administration.

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

- Disk Space Requirements for the Database
- Data Sets Required for UES Support
- Disk Space Requirements for Internal Product Data Sets
- Adabas Nucleus Partition/Address Space Requirements
- Defining the Library
- Restoring the ADAvrs LIBR File
- Using the ADAvrs LIBR File

Disk Space Requirements for Libraries

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

Library	3390 Tracks
Adabas Library	600

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 3390 disk space requirements are:

Database Component	3390 Cylinders Required	3390 Tracks Required
ASSOR1 (Associator)	20	300
DATAR1 (Data Storage)	60	900
WORKR1 (Work space)	15	225
TEMPR1 (temporary work space)	15	225
SORTR1 (sort work space)	15	225

Data Sets Required for UES Support

The Software AG internal product libraries (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 LIBDEF concatenation:

```
APSvrs.LIBR
APSvrs.L0nn
```

where *vrs* is the *version* of the library provided on the most recent tape for these components and *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.

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

```
ADAvrsCS.LIBR
```

For information about setting up connections to UES-enabled databases, see section *Enabling Universal Encoding Support (UES) for Your Adabas Nucleus*.

Disk Space Requirements for Internal Product Data Sets

The minimum disk space requirements on a 3390 disk for the internal product libraries delivered with Adabas Version 8 are as follows:

Library	3390 Cylinders	3390 Tracks
ADA*vrs*CS.LIBR	32	480
APS*vrs*.LIBR	8	120
APS*vrs*.L0*nn*	5	75

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 V8 ADABAS LIBRARY
// OPTION LOG
// EXEC IDCAMS,SIZE=AUTO
DEFINE CLUSTER -
```

```
(NAME(ADABAS.ADAvrs.LIBRARY) -
VOLUME(vvvvvv vvvvvv) -
NONINDEXED -
RECORDFORMAT(NOCIFORMAT) -
SHR(2) -
TRK(nnnnnn)) -
DATA (NAME(ADABAS.ADAvrs.LIBRARY.DATA))
/*
// OPTION STDLABEL=ADD
// DLBL SAGLIB,'ADABAS.ADAvrs.LIBRARY',,VSAM
// EXEC IESVCLUP,SIZE=AUTO
ADABAS.ADAvrs.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.
```

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 V8 ADABAS LIBRARY
// OPTION LOG
// DLBL SAGLIB,'ADABAS.ADAvrs.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.
```

Restoring the ADA*vrs* 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 Online System (AOS)	AOS*vrs*.LIBR	SAGLIB.AOS*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 ADA*vrs* 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 CDLOAD macro must be used. Therefore, the zap to change the ADAUSER CDLOAD to the LOAD macro cannot be used.

Initializing the Adabas Communication Environment

Communication between the Adabas nucleus residing in a z/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 z/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
- ADASIP Processing
- Running ADASIP
- Finding an Unused SVC
- Loading a Secondary Adabas SVC
- ADASIP Execution Parameters
- ADASIP Runtime Display

Installing the Adabas SVC with Turbo Dispatcher Support

The Adabas SVC module supports the IBM z/VSE Turbo Dispatcher environment.

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 z/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.

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 de-activated 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 z/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*

Without Turbo Dispatcher Support

The following sample is available in member ADASIP.X:

```
// DLBL SAGLIB,'ADABAS.ADAvrs.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.ADAvrs.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 z/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 z/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 z/VSE SVC table. The table contains a four-byte entry for each SVC between 0 and 150 (depending on the z/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, where *xx* is the two-byte suffix of the new SVC:

```
// OPTION SYSPARM='svc,xx'
```

ADASIP Execution Parameters

This section describes the ADASIP execution parameters.

- OPTION SYSPARM= Statement
- UPSI Statement
- NRIDTES PARM= Option
- REPLACE PARM= Option
- DMPDBID PARM= Option

Runtime Display

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 z/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 z/VSE SVC (see the section *Loading a Secondary Adabas SVC)*..

UPSI Statement

```
// UPSI DSxTVOGx
```

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 z/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 z/VSE SVC table and the z/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.
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*.

REPLACE PARM= Option

Specifying REPLACE=N or NO will cause warning messages ADASIP80 and ADASIP81 to appear if the SVC has been previously installed. Specifying REPLACE=Y or YES replaces the current SVC regardless of any active targets. The default value is REPLACE=NO. No abbreviation of the REPLACE keyword is supported.

Warning:
Setting the REPLACE parameter to YES should be done carefully. Replacing an SVC while your targets are running can produce unpredictable results.

If both the NRIDTES and REPLACE keywords are specified, they must be separated by a comma. For example:

```
//EXEC ADASIP,PARM='NRIDTES=10,REPLACE=YES'
```

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'
```

ADASIP Runtime Display

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

```
ADASIP00 ...ADABAS V8 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.

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

Installing the Release Tape

Note:
If you are using System Maintenance Aid (SMA), refer to the *System Maintenance Aid* documentation. If you are not using SMA, follow the instructions below.

This section explains how to copy the data sets `.LIBJ`, `.LIBR` and `.LICS` from tape to disk. All other data sets can be installed directly from the tape.

You will then need to perform the individual installation procedure for each component to be installed.

- 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` contains the JCL required to copy the data sets `.LIBJ`, `.LIBR` and `.LICS` from tape to disk. Copy `COPYTAPE.JOB` to your disk by using the following sample JCL:

```
* $$ JOB JNM=LIBRCAT,CLASS=0,                                          +
* $$ DISP=D,LDEST=(*,UID),SYSID=1
* $$ LST CLASS=A,DISP=D
// JOB LIBRCAT
* ***************************************
```

```
*     STORE COPYTAPE.JOB IN LIBRARY
* ************************************
// ASSGN SYS004,nnn
// MTC REW,SYS004
// MTC FSF,SYS004,4
ASSGN SYSIPT,SYS004
// TLBL IJSYSIN,'COPYTAPE.JOB'
// EXEC LIBR,PARM='MSHP; ACC S=lib.sublib'
/*
// MTC REW,SYS004
ASSGN SYSIPT,FEC
/*
/&
* $$ EOJ
```

where:

nnn is the tape address
lib.sublib is the library and sublibrary in which COPYTAPE.JOB is to be stored

Step 2: Modify COPYTAPE.JOB

Modify COPYTAPE.JOB according to your local naming conventions and set the disk space parameters.

Step 3: Submit COPYTAPE.JOB

Submit COPYTAPE.JOB to copy the data sets .LIBJ, .LIBR and .LICS from tape to your disk.

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.ADAvrs.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 ADALODP.X /* LOAD DEMO FILES PERSONNEL & LOB */
PUNCH ADANUC.X /* SAMPLE NUCLEUS STARTUP */
PUNCH ADAREP.X /* SAMPLE ADAREP JOB */
```

```
PUNCH ADAINPL.X /* SAMPLE NATINPL TO INSTALL AOS */
/*
/&
* $$ EOJ
```

where *p* is the output class for punch, *volume* is the specified volume for the Adabas library, and *vrs* is the Adabas *version*.

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
- IORDOSO Macro Parameters

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 z/VSE disk space managers such as DYNAM/D;
- Interfaces to z/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 z/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

The following parameters can be set in using the IORDOSO macro.

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).

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 z/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 3390, 9345 and FBA.

DISKMAN

Parameter	Description
`DISKMAN={ NO | YES }`	Indicates to Adabas that a z/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 z/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 z/VSE disk ASSGN statements. Such statements are sometimes not needed with a z/VSE disk manager facility.

RAITASG

Parameter	Description
RAITASG={ NO | YES }	RAITASG=YES specifies that the Adabas Recovery Aid (ADARAI) is to create z/VSE tape ASSGN statements. Such statements are sometimes not needed with a z/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 2).

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 z/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 tape device types are 2400, 3410, 3420, 3480 and 8809. For device types 3480, 3490, 3490E or 3590, specify TAPEDEV=3480.

TAPEMAN

Parameter	Description
`TAPEMAN={ NO | YES }`	Indicates that a z/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). There is no default value.

Note:
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 z/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.

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).

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 z/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 3390, 9345 and FBA.

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). There is no default value.

Note:
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 z/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.

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.

Database Installation Steps

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

- Step 1. Allocate and format the DEMO database.
- Step 2. Define the global database characteristics.
- Step 3. Load the demonstration (demo) files.
- Step 4. Install the product license file.
- Step 5. Start the Adabas nucleus and test the Adabas communications.
- Step 6. Test Adabas partition communications.
- Step 7. Load the Adabas Online System, if used.
- Step 8. Terminate the Adabas nucleus.
- Step 9. Back up the database.
- Step 10. Insert the ADARUN defaults.
- Step 11. Install the required TP link routines for Adabas.

Notes:

1. For information about running ADADEF, ADAFRM 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, monitoring, controlling, and terminating the nucleus, see the *Adabas Operations* documentation.

Step 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 data sets for each ADAFRM statement.

Step 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.

Step 3. Load the demonstration (demo) files.

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

Customize and run the job

- ADALODE to load the sample demo file EMPL;
- ADALODV to load the sample demo file VEHI;
- ADALODM to load the sample demo file MISC; and
- ADALODP to load the sample demo file PERS and its associated LOB demo file, PERL.

  Note:
  The Personnel demo file must be installed on a UES-enabled database because it includes wide-character format (W) fields.

For each job, the following items must be customized:

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

Step 4. Install the product license file.

The product license file is supplied on the individual customer installation tape or separately via an e-mail attachment. If the license file is provided on an installation tape, you can follow the instructions in this step to install the license file. If the license file is supplied via an e-mail attachment, you must first transfer the license to z/VSE, as described in *Transferring a License File from PC to a z/VSE Host Using FTP* and then you can install it, as described in this step.

Installing the license file.

In z/VSE environments, the product license file can be installed either as a load module or as a library member.

To install the product license file as a module, complete the following steps:

1. Verify that the license file is stored in an Adabas source library or sequential data set (with RECFM=F or FB and LRECL=80), taking care to preserve its format as ASCII.
2. If you loaded your Adabas license file to a library, review and modify sample JCS job ASMLICAL.X, adjusting the library and volume specifications as appropriate for your site. If you loaded your Adabas license file to a data set, use sample ASMLICAV.X instead.

   Note:
   In sample jobs ASMLICAL.X and ASMLICAV.X, the standard label area is assumed to contain label information for library USERLIB. You can change this as appropriate for your library.
3. Submit modified sample job ASMLICAL.X or ASMLICAV.X.

   These sample jobs generate your Adabas license in ADALIC.PHASE. They assume that ADALIC.PHASE will be in a user sublibrary. If a user sublibrary is chosen for ADALIC.PHASE, this sublibrary must be included in the LIBDEF search chain in your Adabas nucleus startup JCS. You may find it more convenient to place ADALIC.PHASE directly into the Adabas ADA*vrs* sublibrary, to avoid the need to define additional libraries. During initial testing, Software AG recommends using a user sublibrary.

To install the product license file as a library member, complete the following steps:

1. Verify that the license file is stored in an Adabas source library (with RECFM=F or FB and LRECL=80), taking care to preserve its format as ASCII.
2. Make sure any previously created ADALIC load module is inaccessible in the Adabas load library being used by the nucleus jobs. Adabas first tries to load ADALIC and if unsuccessful it reads from DDLIC.
3. Provide all Adabas nucleus startup jobs with a DLBL statement in the following format:

   ```
   // DLBL DDLIC,'/libname/sublb/memname.memtype'
   ```

   where *libname* is the Librarian name of the library, *sublib* is the name of the sublibrary, *memname* is the license member name, and *memtype* is the license member type.

To install the product license file as a sequential data set, complete the following steps:

1. Verify that the license file is stored in a sequential file (with RECFM=F or FB and LRECL=80), taking care to preserve its format as ASCII.
2. Make sure any previously created ADALIC load module is inaccessible in the Adabas load library being used by the nucleus jobs. Adabas first tries to load ADALIC and, if unsuccessful, it reads from DDLIC.

3. Provide all Adabas nucleus startup jobs with DLBL, EXTENT and ASSGN statements in the following format:

```
// DLBL DDLIC,'adabas.license.file'
// EXTENT SYSnnn
// ASSGN SYSnnn,DISK,VOL=volser,SHR
```

   where *adabas.license.file* is the physical file name, *nnn* is an unused logical unit, and *volser* is the volume serial on which the license file resides.

Step 5. Start the Adabas nucleus and test the Adabas communications.

Note:
Sample JCS is available in ADANUC.X.

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

- The Adabas SVC number, the database ID, and device type(s);

  Note:
  Be sure to include appropriate LIBDEF references for user sublibraries, especially the library containing the ADALIC license file. The licensing component MLC*vrs* must also be added to the LIBDEF SEARCH chain for load modules. These additional sublibraries can be added via the ADAV*v*FIL procedure, as required.

- ADANUC parameters.

Step 6. 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.

Step 7. Load the Adabas Online System, if used.

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

Customize and run the job ADAINPL 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.

Step 8. 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.

Step 9. 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.

Step 10. 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).

Step 11. Install the required TP link routines for Adabas.

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

Migrating 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.

This section covers the following topics:

- Installation and Initialization
- Librarian Input Override Processing
- Activating Adabas Use of Job Exit Processing
- Using the Job Exit Utility for ADARAI JCS Capture
- Job Exit Storage Requirements

- Optional Console or Printer Messages
- Diagnostic Functions

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*.

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 z/VSE on z/VSE 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, link-edit 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.

SVC Work Areas

For each Adabas SVC installed, a number of 384-byte work areas are reserved. The number of work areas reserved is calculated as four times the number of IDTEs (`4 x IDTE-count`). The maximum number of work areas allocated is 128; the minimum is 24. The SVC work areas therefore occupy between 9K and 48K of storage. The default value of 10 IDTEs results in 15K of SYSTEM GETVIS being allocated.

Displaying Storage Allocation Totals

Specifying `// UPSI` *xxxxxxx*G*x* during the ADASIP execution (see UPSI byte description in *ADASIP Execution Parameters*, earlier in this guide,) 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 Partitions

In order for an Adabas nucleus to accept calls from other partitions, storage is acquired in the SVA GETVIS area for any required attached buffers. The buffers hold data moved between the nucleus and users in other partitions.

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.

- Applying Fixes Using MSHP PATCH
- Applying Fixes Using MSHP CORRECT
- Link Book Update Requirements for Secondary SVC
- 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
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

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 z/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
saglib is the Adabas library name in procedure ADAVvFIL
```

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
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

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 V*vr* history file resides
numtrks is the length of the extent on which the Adabas V*vr* history file resides
adabas.new.version.history.file is the physical name of the Adabas V*vr* history file

MSHP CORRECT

The MSHP CORRECT and UNDO jobs use the history file identified by label IJSYSHF in the z/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 V*vr* history file resides
nnn is the user-defined SYS number
adabas.new.version.history.file is the physical name of the Adabas V*vr* 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
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*, *x* is the Adabas component (for example, N for nucleus), and *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" and the member type is "OBJ".

No link book is provided for module ADAOPD or for any other programs distributed in source form. Programs distributed in source form continue to be modified using assembly and link jobs.

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 z/VSE system history file. See *Maintaining a Separate Test Environment in z/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 8 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 ADASVC81 is LNKSVC.OBJ. It contains the following:

```
PHASE ADASVC81,*,NOAUTO,SVA
MODE AMODE(31),RMODE(24)
INCLUDE SVCVSE
INCLUDE SVCCLU
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 SVCCLU
ENTRY ADASVC
/+
/*

—where
vrs is the Adabas version
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.

Adabas 8 Adalink Considerations

- Link Routine User Exit 1 (Pre-Command) and User Exit 2 (Post-Command)
- LNKUES for Data Conversion
- ADAUSER Considerations

Link Routine User Exit 1 (Pre-Command) and User Exit 2 (Post-Command)

A pre-command user exit and a post-command user exit may be linked with an Adalink routine:

- Link routine user exit 1, LUEXIT1, 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 (ACBX). LUEXIT1 must check this byte and return if it is set to X‘04’. Be sure to reset R15 to zero on return.

- Link routine user exit 2, LUEXIT2, 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 an 18-word format 1 register save area
13	For CICS, on entry to the link user exit, R13 points to the CICS DFHEISTG work area at *xxxxxxxxx*. For batch/TSO, R13 points to the link routine’s work area.
14	Return address
15	Entry point address: LUEXIT1 or LUEXIT2

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 LUEXIT1, 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 ACBXRSP to a non-zero value to indicate to the calling program that it has suppressed the command: response code 216 (ADARSP216) is reserved for this purpose.

The LUEXIT1 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 LUEXIT2 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 ACBXRSP:

Response Code	Description
213 (ADARSP213)	No ID table
216 (ADARSP216)	LUEXIT1 suppressed the command
218 (ADARSP218)	No UB available

LNKUES for Data Conversion

The Adabas 8 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.

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 LUEXIT1.
- For replies, LNKUES receives control after LUEXIT2.

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 ADALNK,*
MODE AMODE(31),RMODE(24)
INCLUDE LNKVSE8
INCLUDE LINKIND
INCLUDE LINKGBLS
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 8 and you do not have the correct LNKUES/ADALNK module, Adabas produces unexpected results: response code 022 (ADARSP022), 253 (ADARSP253), 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 non-reentrant Adalink modules. To load a reentrant batch link routine, use the ADARUN parameter `PROGRAM=RENTUSER`. 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	3390
SVC number	45
Database ID	1