Adabas Bridge for DL/1

Installation

Version 2.3.1

June 2014

Adabas Bridge for DL/I

This document applies to Adabas Bridge for DL/1 Version 2.3.1.

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

Copyright © 2014 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors..

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

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at http://documentation.softwareag.com/legal/.

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

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

Document ID: ADL-INSTALL-231-20140626

Table of Contents

Installation

This documentation provides an overview of the Installation procedure for the Adabas Bridge for DL/I.

The following topics are covered:

1 Introduction

The Adabas Bridge for DL/I (ADL) is Software AG's tool for the migration of DL/I or IMS/DB databases into Adabas. DL/I applications can continue to run without any change. The migrated data can be manipulated by Natural, Software AG's fourth generation language. The migrated data can be accessed by SQL applications, if the Adabas SQL gateway is available. ADL can also be used to run standard DL/I applications on Adabas sites.

The Adabas Bridge for DL/I runs under z/VSE and z/OS. It can execute concurrently in both batch and online environments supporting CICS and IMS/TP.

ADL consists of the following major functional units:

- The ADL Conversion Utilities allow an automatic conversion of the DL/I data bases into Adabas files. Throughout the ADL documentation, an Adabas file originating from the conversion of a DL/I data base will be referred to as an 'ADL file'. This is to point out the particular properties of these files versus native Adabas files.
- The ADL Directory is an Adabas file, where as a result of the conversion process, the DL/I data base definitions and the related Adabas file layouts are stored. In addition, the ADL Directory contains the ADL error messages and other information.
- With the ADL Online Services the contents of the ADL Directory can be examined and the ADL Interfaces under CICS can be maintained (start, stop, etc.).
- The ADL CALLDLI Interface allows DL/I applications to run against Adabas. It supports assembler, COBOL, PL/1, RPG, FORTRAN, and Natural for DL/I. For programs using the `'EXEC DLI'` interface a special precompiler is available.
- The ADL Consistency Interface allows Natural applications, or programs using Adabas direct calls to manipulate the migrated data.

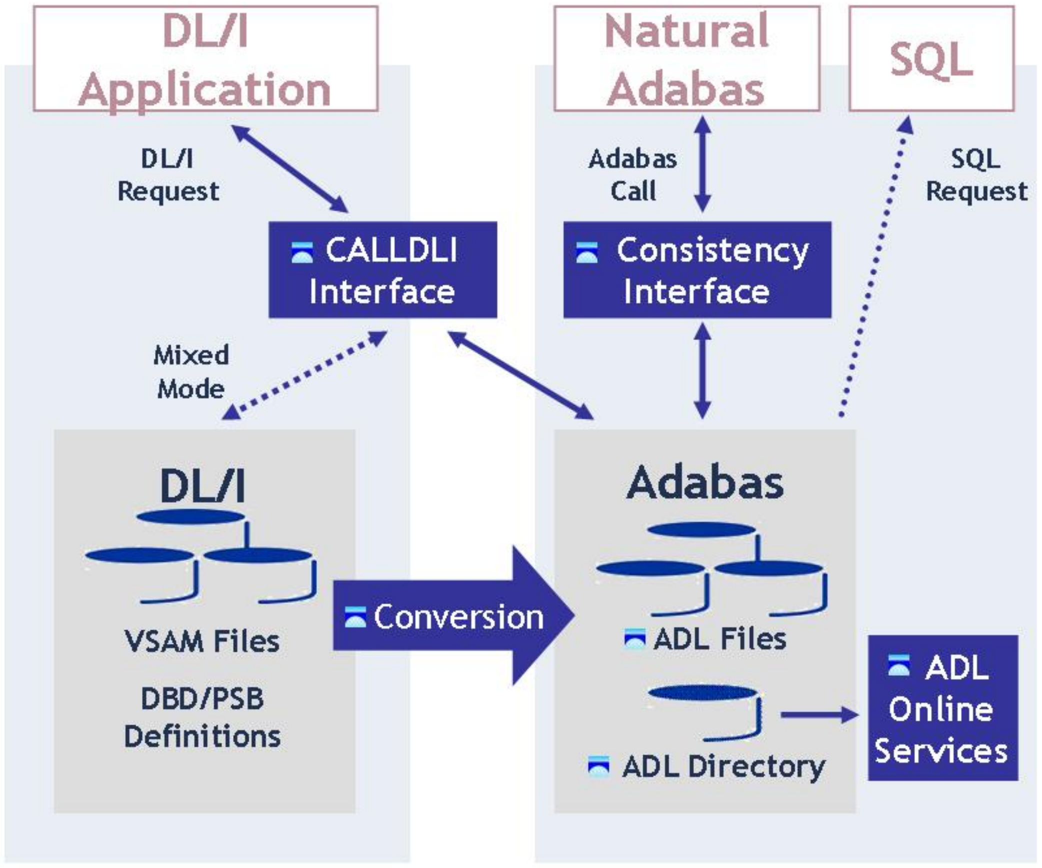


Figure 1

Figure 1 shows the individual functional units of ADL and their interrelation with DL/I, SQL and Natural applications.

The *ADL Installation* documentation describes the installation of the basic components of ADL, namely

- the ADL source library,
- the ADL load library,
- the ADL directory file,
- the ADL parameter module,
- the ADL executable nuclei and
- the ADL Online Services.

Additionally, the **ADL Installation Verification Package** is described in the ADL Installation.

Note: Most parts of the ADL installation can be performed with Software AG's System Maintenance Aid (SMA). Refer to the SMA documentation for more details.

Once this installation process has been completed, you will be prepared

- to convert DL/I DBD and PSB definitions into entries in the ADL directory file
- to transfer data from a DL/I data base into an ADL file
- to install and operate the ADL Interfaces for `CALLDLI` programs and Natural/Adabas applications

The conversion of data definitions and databases is described in the *ADL Conversion* documentation. The installation and operation of the interfaces is covered by the *ADL Interfaces* documentation. In particular, this documentation describes the installation of the ADL Interfaces directly related to the TP monitor in use.

The *ADL Messages and Codes* documentation comprises a list of all error codes and messages issued by ADL together with a glossary of terms related to DL/I, Adabas and ADL.

The *ADL Installation* documentation is intended for the system programmer in charge of the ADL installation. The installation process requires familiarity with the operating system in use. No particular knowledge of either data base system involved is required.

This documentation applies to both z/OS and z/VSE operating systems. References valid for only one operating system are clearly marked as such. The term "DL/I" is used as a generic term for IMS/VS and DL/I DOS/VS.

This chapter covers the following topics:

DL/I Features Supported

The Adabas Bridge for DL/I provides a `CALLDLI` and an `EXEC DLI` interface identical to the DL/I language interface. In special the current version of the Adabas Bridge for DL/I supports the following DL/I features:

- Up to 15 hierarchical levels;
- Secondary indices;
- Index maintenance exit routines;
- Duplicate data fields;
- Variable length segments;
- Inverted structures;
- Bidirectional logical relationships (physically paired and virtually paired);
- Unidirectional logical relationships;

- Fixed intersection data;
- Variable intersection data segments;
- Processing of an index database as a data database;
- Sensitive fields;
- Automatic data type conversion in conjunction with sensitive fields;
- Multiple positioning;
- Path calls;
- The command codes “C”, “F”, “L”, “P”, “N”, “U” and “V”;
- The insert, delete and replace rules “V”, “L”, “P” and “B”;
- Restricted support of GSAM data bases.

DL/I Features not Supported

The current version of the Adabas Bridge for DL/I does *NOT* support the following DL/I features:

- Fast path data bases;
- Sequence fields of root segment types exceeding a length of 253 bytes;
- Sequence fields of dependent segment types exceeding a length of 120 bytes;
- Data edit and compression routines;
- The `INDICES` parameter in the `SENSEG` macro;
- The independent `"AND"` (`"#"`) in conjunction with secondary indices;
- Virtual fields;
- Shared secondary index data bases;
- The `ACCESS=(INDEX,NOPROT)` parameter in the `DBD` macro;
- The use of time stamps as checkpoint identifiers (restart/recovery procedures);
- OS/VS checkpoint requests (restart/recovery procedures);
- `LAST` (restart/recovery procedures under BMP);
- The `DEQ`, `STAT` and `LOG` functions.

Benefits

The Adabas Bridge for DL/I has the following additional features and advantages:

- Administartion for one database only, namely Adabas;
- Pay for one database only;
- Manipulation of the data by full functional Natural, Software AG's fourth generation language;
- Access to migrated data with SQL applications;
- Automatic data compression at field level (by Adabas);
- No corrupted pointers because ADL does not use pointers;
- Deleted data records are released immediately from storage. This is in contrast to DL/I, which simply sets a flag in such records but does not release the storage used by them. With Adabas, the released space can be re-used immediately for new records, there is no requirement to maintain records that are marked as “deleted”, and less re-organization of the data base is required;
- All converted DBDs have full `"HIDAM"` functionality, regardless of the original `ACCESS` method;
- The PSB “language” can be overwritten by batch JCL;
- Increasing the length of a field without unloading and reloading the data;
- Adding segments to the end of a DBD without unloading and reloading the data;
- Trace facilities for online and batch;
- `'EXEC DL/I'` programs can run under IMS/TP;
- `CALLDLI` test program for batch;
- Reduced online system resources, since Adabas (unlike DL/I) is not running in the CICS region/partition.
- Symbolic checkpoint facility under z/VSE.
- HD databases under z/OS.

Migration Planning Checklist

If you are planning to migrate from DL/I or IMS/DB to Adabas using the Adabas Bridge for DL/I the following questions should be answered:

- Which operating system, which TP monitor is used?
- Number of PSBs, how many online, how many in batch?
- Number of physical DBDs (without index or logical DBDs)?

- Number of records (occurrences) in the DBDs?
- Are there packed fields in the DBD definition?
- Are there logical relationships? How many?
- Which `ACCESS` method is used? `SHISAM/GSAM`?
- Are there `FAST PATH` DBDs?
- Are the DBDs, PSBs available as sources?
- Which programming languages are used?
- Are the programs using the `EXEC DLI` or the `CALLDLI` interface?
- Are the programs available as sources (especially the `EXEC DLI` applications)?
- How many applications have to be tested?
- Are there any not-documented DL/I features used?
- Are there any DL/I features used, which are not supported by ADL? See the corresponding list earlier in this chapter.
- Are there any time-critical applications?
- Are there any plans to access the migrated data with Natural?

Other Documentation You May Need

The following Software AG publications may be useful when installing and operating the ADL Interface:

- *Adabas Utilities* documentation
- *Adabas Operations* documentation
- *Adabas Messages and Codes*
- *Adabas Reference Data* and *Adabas DBA Reference Data* documentation.

For a complete list of Software AG documentation, refer to Software AG's **Empower** web site. If you do not have an Empower user ID and password yet, you will find instructions for registering on this site (free for customers with maintenance contracts).

2 z/OS Installation Tape

The Adabas Bridge for DL/I installation tape is a standard-label tape. The Report of Tape Creation that accompanies the installation tape lists the volume serial number, tape density, media type, data sets and sequence numbers. The tape is compatible with Software AG's System Maintenance Aid (SMA). Refer to the SMA documentation for more details.

The ADL installation tape for systems operating under the IBM z/OS Operating System contains the data sets as described in the table below. The "LOAD", "LC23" and "SOURCE" data sets have been unloaded to tape using the `IBM IEBCOPY` utility, and the other data sets have been copied by the `IBM IEBGENER` utility.

In this table, *vrs* indicates the current version, release and system maintenance (SM) level as indicated on the Report of Tape Creation, for example *123* for Version 1, Release 2 and SM 3.

Library	Description
ADL*vrs*.LOAD	This data set contains the ADL Load Library, which comprises the load modules used to link edit executable ADL modules.
ADL*vrs*.LC23	This data set contains the executable ADL modules required for CICS TS 2.3 and below.
ADL*vrs*.SRCE	The ADL Source Library. This data set contains JCL examples for installing the ADL and converting DL/I data bases, as well as macros for reassembling DBDs and PSBs, and source code for sample data base unload programs.
ADL*vrs*.SYSF	An unloaded Adabas file for use as the ADL directory file.
ADL*vrs*.INPL	An unloaded Natural library containing the ADL Online Services and the ADL Installation Verification Package.

The following sections describe the libraries in more detail.

This chapter covers the following topics:

ADL Load Library

When loaded from the installation tape, the ADL Load Library contains executable ADL load modules and load modules used in the installation process to create executable load modules. The precise contents of the library are listed in the following table.

Module	Description
ADLEXITB	Adabas User Exit for the ADL Consistence Interface under CICS/ESA.
DAZAXES	ADL nucleus module containing the logic for generating and issuing Adabas calls.
DAZBDOKE	Batch doorkeeper for the ADL Consistency Interface.
DAZBENT	Batch entry point module.
DAZBIFP	Batch interface program (object module).
DAZBRQH	Batch request handler.

Module	Description
DAZCAPRI	CICS application program interface.
DAZCCGEN	ADL nucleus module for the CBC utility.
DAZCCOUT	ADL nucleus module for the CBC utility.
DAZCCSUB	ADL nucleus module for the CBC utility.
DAZCDOKE	CICS doorkeeper for the ADL Consistency Interface.
DAZCDPOS	ADL nucleus module containing the logic for retrieval calls and for maintaining positional information.
DAZCDUMP	CICS — Write ADL tables on dump file.
DAZCEND	CICS — ADL shut-down program (PLT).
DAZCICS	CICS control program.
DAZCIFP	CICS interface program.
DAZCINF	CICS — Return status of ADL.
DAZCINIT	CICS — ADL start-up program (PLT).
DAZCLUB	CICS – Allocate local user blocks.
DAZCONSI	Environment independent routines of the ADL Consistency Interface.
DAZCRQH	CICS request handler.
DAZCSVC	ADL SVC for CICS.
DAZCTOFF	CICS — Switch off ADL trace.
DAZCTON	CICS — Switch on ADL trace.
DAZDEBUG	ADL nucleus module containing tracing and debugging routines.
DAZDREIN	ADL nucleus module containing the logic for DELETE, REPLACE and INSERT calls.
DAZELORE	Establish Logical Relationships utility.
DAZENTRY	Entry program for IMS/TP application programs.
DAZEXEC	ADL nucleus module containing the logic for the EXEC command precompiler.
DAZIFP	Batch interface program - executable module (AMODE=24).
DAZINICB	ADL nucleus module containing the logic for initializing internal control blocks.
DAZLANP	ADL nucleus module containing the logic for the language processor.
DAZLIBAT	Language Interface for batch mode.
DAZLICI3	Language interface for CICS.
DAZMIX	Batch mixed mode module.
DAZMPL	Pre-load program for IMS/TP.
DAZREFOR	Reformat utility.
DAZSERV	ADL nucleus module containing general purpose service routines.
DAZSTUB	CICS — Stub for the ADL task-related user exit.
DAZSYNC	CICS — ADL task-related user exit.
DAZSYXTB	ADL nucleus module containing the EXEC command syntax.

Module	Description
DAZUEX06	Adabas User Exit 6.
DAZUEXMI	Adabas User Exit 6 for migration to ADL 2.3.
DAZUNDLI	Data Base Unload utility (automated procedure).
DAZZAP	ADL nucleus module containing zaps for nucleus routines.
DAZZLER	Program to test DL/I calls in batch.

ADL Load Library for CICS TS 2.3 and below

The standard ADL load library contains the modules for CICS TS 3.1 and CICS TS 3.2. For CICS releases with changes relevant to ADL, an additional library is delivered containing the ADL modules affected. This library is not a complete ADL load library; it contains only the CICS release dependent ADL modules.

When loaded from the installation tape, the ADL Load Library for CICS TS 2.3 and below contains executable ADL load modules required to operate the ADL under CICS TS 2.3 or below (until CICS 3.2). When you run under any of these CICS releases, concatenate this library in front of the ADL load library when you link the ADL CICS nucleus DAZNUCC, and in the concatenation list of the CICS steplibs.

If your current CICS version is not compatible to the ADL modules, ADL will issue a message ADL0906 at start-up.

The precise contents of the library are listed in the following table.

Module	Description
DAZCICS	CICS control program.
DAZCIFP	CICS interface program.
DAZCLUB	CICS – Allocate local user blocks.
DAZCRQH	CICS request handler.

ADL Source Library

When loaded from the installation tape, the ADL Source Library contains:

- Macros for creating an ADL parameter module;
- Macros for creating Adabas User Exit 6 extensions;
- Macros for creating the CICS runtime control tables;

- Macros for assembling DBDs and PSBs (substitutes for the original `DL/IDBDGEN` and `PSBGEN` macros);
- Source code for sample unload programs;
- Source code for performing ADL functions under CICS;
- Source code for the ADL supplied Adabas link module substitutes;
- Sample JCL.

The following table lists the other macros contained in this library.

Member	Description
DAZLDT	Macro used to create entries for logical `DBID/FNR` assignments in the ADL parameter module. This macro is provided for compatibility with ADL 2.2 only.
DAZTCF	Macro used to create the table of converted files for the ADL Consistency Interface (batch or CICS).
DAZPARM	Macro used to create the ADL parameter module.
ZFNR	Macro used to create Adabas User Exit 6 extensions.
ZPCK	Macro used to create Adabas User Exit 6 extensions.
ZREC	Macro used to create Adabas User Exit 6 extensions.
ZSEG	Macro used to create Adabas User Exit 6 extensions.
ZSEX	Macro used to create Adabas User Exit 6 extensions.
ZVCK	Macro used to create Adabas User Exit 6 extensions.
MGPSTIN	Macro used to create the DAZPSB table.
MGPSTEN	Macro used to create the DAZPSB table.
MGPSTFI	Macro used to create the DAZPSB table.
DFHDLPSB	Macro used to create a DAZPSB table from a CICS PDIR.
BUFMAC	Macro used to create the DAZBUF table.
DBDMAC	Macro used to create the DAZDBD table.
MEXRENDS	DSECT describing an entry in the exit routine table for index maintenance.

The following table lists the source members used during the assembly of the ADL supplied Adabas link module substitute:

Member	Description
DAZLNKO	Source member to be assembled as the ADL supplied Adabas link module substitute in batch.
DAZLNK	Operating system independent part of the ADL supplied Adabas link module substitute in batch.
LNKOS	Operating system dependent part of the ADL supplied Adabas link module substitute in batch.

The following table lists the sample JCL streams for the installation and conversion processes:

Member	Description
ADLINS1	JCL for loading the ADL source and load library from tape.
ADLINS2	JCL for loading the ADL directory file from tape.
ADLINS2A	JCL for updating an existing ADL directory file.
ADLINS3	JCL for an initial program load of the ADL Online Services, the ADL supplied Natural subprograms ADLERROR and ADLACTIV and an initial program load of the Natural programs for the ADL Installation Verification Package.
ADLINS4	JCL for creating the ADL parameter module.
ADLINS5U	JCL for creating an executable ADL CBC utility module.
ADLINS6P	JCL for creating an executable ADL precompiler module.
ADLINS7B	JCL for creating an executable ADL batch module.
ADLINS8A	JCL for creating an executable ADL Consistency Interface module for batch.
ADLINS9C	JCL for creating an executable ADL CICS module.
ADLINS10	JCL for creating an executable ADL Batch Region Controller.
ADLINS11	JCL for assembly and link-edit of the ADL substitute for the Adabas link module in batch.
ADLINS12	JCL for assembly and link-edit of the ADL substitute for the Adabas link module under CICS.
ADLDPC	JCL for running a physical DBD through the DBD conversion process.
ADLDPC1	JCL for assembling and link editing a DBD/PSB.
ADLDPC2	JCL for converting PSBs and logical DBDs.
ADLDPC23	JCL for converting physical DBDs.
ADLDPC4	JCL for assembling the User Exit 6 extension.
ADLDBC4	JCL for unloading a DL/I data base with the `DAZUNDLI` utility
ADLDBC6	JCL for loading a DL/I data base into Adabas (initial load).
ADLDBC7	JCL for loading a DL/I data base to Adabas (mass update).
ADLDBC9	JCL for establishing a logical relationship.
ADLCSD	Sample CICS system definition file.
ADLCTG1	JCL for assembling the `DAZPSB` table.
ADLCTG2	JCL for running the "`DAZSHINE`" utility.
ADLCTG3	JCL for assembling the `DAZBUF` table.
ADLCTG4	JCL for assembling the `DAZDBD` table.
ADLCFCT	JCL for precompile, assembly and link-edit of `DAZCFCT`.
ADLTCF	JCL for assembly of the `DAZTCF` table.
ADLBATCH	JCL for running the `DAZZLER CALLDLI` test program with the trace facility.

Note the naming conventions for the JCL examples:

```
ADLxxxnn  General sample JCL
IVPxxxnn  JCL for the Installation Verification package
```

where *xxx* is

INS	for the installation steps (refer to the *z/OS Installation* documentation;
DPC	for the DBD/PSB conversion steps (section *ADL Conversion Utilities for DBDs and PSBs* in the *ADL Conversion* documentation);
DBC	for the data base conversion steps (section *ADL Data Conversion Utilities* in the *ADL Conversion* documentation);
CTG	for the CICS table generation (section *Generating the Runtime Control Tables*), in the *ADL Interfaces* documentation;
TCF	for the generation of the table of converted Adabas files. (see the section *Batch Installation and Operation* in the *ADL Interfaces* documentation);

and `nn` is the number of the step.

The other members in the Source Library are as follows:

Member	Description
$INFO$	Information about the current ADL release.
DAZUNLOD	Source of the sample unload program.
DAZREFOR	Source of the sample reformatting program.
DAZCFCT	Source of the sample program to perform ADL functions under CICS.
DAZEISTG	`DSECT` used by `DAZCFCT`
ADLEX06	ADACMP User Exit 6 skeleton, for changing the layout of an ADL file.
ADLIMEX	Sample user exit routine for index maintenance.
IVPINFO	Abbreviations used by the ADL Installation Verification Package (IVP).
IVPARUN	`ADARUN` cards for the `ADL IVP`.
IVPCOB	Execute a `COBOL` batch program of the `ADL IVP`.
ADLXPCn	`COBOL` sources for the `ADL IVP`.
ADLXPIn	Input streams for the `COBOL` programs for the `ADL IVP`.
ADLXPAn	Assembler sources for the `ADL IVP` to run under CICS.
ADLXPDn	`DAZZLER` input streams for the `ADL IVP`.
COURSEDB	Physical DBD definitions for the `ADL IVP`.
COURSEL	Logical DBD definitions for the `ADL IVP`.
INSTDB	Physical DBD definitions for the `ADL IVP`.
INSTL	Logical DBD definitions for the `ADL IVP`.
MAINIDX	Primary Index DBD definition for the `ADL IVP`.
INSTIDX	Primary Index DBD definition for the `ADL IVP`.

Member	Description
STUDIDX	Secondary Index DBD definitions for the ADL IVP.
SCHOOL	PSB definitions for the ADL IVP.
COURSUNL	PSB definitions for the ADL IVP (DAZUNDLI utility).
INSTUNL	PSB definitions for the ADL IVP (DAZUNDLI utility).
INSTELO	PSB definitions for the ADL IVP (DAZELORE utility).

ADL Directory File

The unloaded Adabas Directory file on the installation tape was created by the Adabas Unload utility, ADAULD. At installation, the file contains the texts of the ADL error messages. Later, it will also be used as the directory file for storing the DBDs and PSBs for ADL and any checkpoint information.

ADL Natural Programs

This file contains the unloaded Natural programs comprising the ADL Online Services, together with the ADL supplied Natural subprograms ADLERROR and ADLACTIV and the ADL Installation Verification Package. The ADLERROR subprogram may be used by Natural applications to retrieve the comprehensive error messages of the ADL Consistency Interface. The ADLACTIV subprogram may be used by Natural applications to verify whether the ADL Consistency Interface is active or not.

The files were created with the Natural SYSOBJH utility.

3 z/OS Installation

This chapter covers the following topics:

Overview

This chapter describes the steps necessary to install the Adabas Bridge for DL/I (ADL) in a z/OS environment. After performing these steps, you will be able to run the ADL data base conversion utilities, to use the ADL Online Services and to operate the ADL Interfaces for DL/I and Adabas calls.

For easy reference, all installation steps are summarized below. In general, all steps must be performed in all environments. Exceptions are clearly marked as such in the detailed description of the individual steps.

Step	Description
Step 1	Load the ADL libraries to disk.
Step 2	Load the ADL directory file.
Step 3	Load the ADL Online Services and the ADL Installation Verification Package.
Step 4	Create the ADL parameter module.
Step 5	Create the executable ADL data conversion module.
Step 6	Create the executable ADL precompiler module.
Step 7	Create the executable module for the ADL CALLDLI Interface in batch.
Step 8	Create the executable module for the ADL Consistency Interface in batch.
Step 9	Create the executable module for the ADL Interfaces under CICS.
Step 10	Create the executable ADL batch region controller.
Step 11	Create the ADL substitute for the Adabas link module in batch.
Step 12	Link-edit the Adabas link module under CICS with the ADL exit `ADLEXITB`.

Note: If you use Software AG's System Maintenance Aid (SMA), steps 1 to 9 are performed by SMA.

Initial Load of the ADL Libraries (Step 1)

Load the ADL libraries to disk using the JCL given below as an example.

Library Space Requirements (z/OS)

The table below gives an estimate of how much space is needed for various device types (primary cylinders, directory tracks).

Library	3390
Load	3, 10
Source	6, 36
LC23	1, 2

Copying the Tape Contents to a z/OS Disk

If you are using SMA, refer to the *System Maintenance Aid* documentation (included in the current edition of the Natural documentation CD).

If you are *not* using SMA, follow the instructions below.

This section explains how to:

- Copy dataset COPY.JOB from tape to disk.
- Modify this dataset to conform with your local naming conventions.

The JCL in this dataset is then used to copy all datasets from tape to disk.

If the datasets for more than one product are delivered on the tape, the dataset COPY.JOB contains the JCL to unload the datasets for all delivered products from the tape to your disk.

After that, you will have to perform the individual install procedure for each component.

Step 1 - Copy Dataset COPY.JOB from Tape to Disk

The dataset COPY.JOB (Label 2) contains the JCL to unload all other existing datasets from tape to disk. To unload COPY.JOB, use the following sample JCL:

```
//SAGTAPE JOB SAG,CLASS=1,MSGCLASS=X
//* --------------------------------
//COPY EXEC PGM=IEBGENER
//SYSUT1 DD DSN=COPY.JOB,
// DISP=(OLD,PASS),
// UNIT=(CASS,,DEFER),
// VOL=(,RETAIN,SER=<Tnnnnn>),
// LABEL=(2,SL)
//SYSUT2 DD DSN=<hilev>.COPY.JOB,
// DISP=(NEW,CATLG,DELETE),
// UNIT=3390,VOL=SER=<vvvvvv>,
// SPACE=(TRK,(1,1),RLSE),
// DCB=*.SYSUT1
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
// ↩
```

Where:

`<hilev>` is a valid high level qualifier
`<Tnnnnn>` is the tape number
`<vvvvvv>` is the desired volser

Step 2 - Modify COPY.JOB

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

- Set HILEV to a valid high level qualifier.
- Set LOCATION to a storage location.
- Set EXPDT to a valid expiration date.

Step 3 - Submit COPY.JOB

Submit COPY.JOB to unload all other datasets from the tape to your disk.

Initial Load of the ADL Directory File (Step 2)

Load the ADL Directory File. This file is a standard Adabas file unloaded with the `ADAULD` utility of Adabas. You may use the JCL in the ADL source library member `ADLINS2` as an example.

If ADL is already installed at your site, and you want to continue to use the existing ADL directory file, you may simply delete the obsolete error messages from the existing Directory File and load the new messages from the unloaded file provided on the installation tape. You may use the JCL in the ADL source library member `ADLINS2A` as an example.

Initial Program Load of the ADL Natural Programs (Step 3)

Load the ADL Online Services and the `DDM` for the ADL Directory File, `ADB-CONTROL`. The `INPL` file on the installation tape was created with the Natural `SYSOBJH` utility and is compatible with Natural version 4.1 and upwards. You may use the JCL in the ADL source library member `ADLINS3` as an example.

Note that the `INPL` file on the ADL installation tape contains a `DDM` for the ADL Directory File and the object modules of the library `SYSADL` with the ADL-supplied subprograms `ADLERROR` and `ADLACTIV`. These subprograms may be used by Natural applications operating under the ADL Consistency Interface.

The same step loads also the ADL Installation Verification Package (`SYSADLIV`) and the related DDMs.

Creating the ADL Parameter Module (Step 4)

Create the ADL parameter module (`DAZPARM`) by assembling the `DAZPARM` macro. This procedure is common to both z/OS and z/VSE and is described in the chapter ***ADL Parameter Module*** in this documentation.

You may use the JCL in the ADL Source Library member `ADLINS4` as an example.

Creating the ADL Executable Load Modules (Steps 5 - 10)

The following steps are performed:

- Step 5
- Step 6
- Step 7
- Step 8
- Step 9

- Step 10

Step 5

Create the executable ADL CBC utility module by running the link editor and specifying the following link edit directives:

```
ORDER DAZPARM
INCLUDE ADLLOAD(DAZPARM)
INCLUDE ADLLOAD(DAZCCGEN)
INCLUDE ADLLOAD(DAZCCOUT)
INCLUDE ADLLOAD(DAZCCSUB)
INCLUDE ADLLOAD(DAZAXES)
INCLUDE ADLLOAD(DAZSERV)
INCLUDE ADLLOAD(DAZDEBUG)
INCLUDE ADLLOAD(DAZZAP)
NAME DAZNUCU(R)
```

You may use the JCL in the ADL Source Library member `ADLINS5U` as an example.

Step 6

This step need only be performed where one or more of the application programs to be converted uses the `HLPI`.

Create the executable ADL precompiler module by running the link editor and specifying the following link edit directives:

```
ORDER DAZPARM
INCLUDE ADLLOAD(DAZPARM)
INCLUDE ADLLOAD(DAZEXEC)
INCLUDE ADLLOAD(DAZSERV)
INCLUDE ADLLOAD(DAZLANP)
INCLUDE ADLLOAD(DAZSYXTB)
INCLUDE ADLLOAD(DAZDEBUG)
INCLUDE ADLLOAD(DAZZAP)
NAME DAZNUCP(R)
```

You may use the JCL in the ADL Source Library member `ADLINS6P` as an example.

Step 7

Create the executable ADL CALLDLI Interface batch module by running the link editor and specifying the following link edit directives. This step must also be performed when installing under IMS/TP:

```
ORDER DAZPARM
INCLUDE ADLLOAD(DAZPARM)
INCLUDE ADLLOAD(DAZBENT)
INCLUDE ADLLOAD(DAZAXES)
INCLUDE ADLLOAD(DAZSERV)
INCLUDE ADLLOAD(DAZINICB)
INCLUDE ADLLOAD(DAZCDPOS)
INCLUDE ADLLOAD(DAZDREIN)
INCLUDE ADLLOAD(DAZDEBUG)
INCLUDE ADLLOAD(DAZZAP)
NAME DAZNUCB(R)
```

You may use the JCL in the ADL source library member ADLINS7B as an example.

Step 8

Create the executable ADL Consistency Interface module for batch by running the linkage-editor and specifying the following link-edit directives:

```
ORDER DAZPARM
INCLUDE ADLLOAD(DAZPARM)
INCLUDE ADLLOAD(DAZAXES)
INCLUDE ADLLOAD(DAZBDOKE)
INCLUDE ADLLOAD(DAZBRQH)
INCLUDE ADALOAD(ADAUSER)
INCLUDE ADLLOAD(DAZCDPOS)
INCLUDE ADLLOAD(DAZCONSI)
INCLUDE ADLLOAD(DAZDEBUG)
INCLUDE ADLLOAD(DAZDREIN)
INCLUDE ADLLOAD(DAZINICB)
INCLUDE ADLLOAD(DAZSERV)
INCLUDE ADLLOAD(DAZZAP)
NAME DAZNUCA(R)
```

You may use the JCL in the ADL source library member ADLINS8A as an example.

The ADAUSER object module must be included from a valid Adabas load library.

Note: Whenever you want the user written user exit (DAZUEX01) to become active in batch you must, in addition, include your user exit DAZUEX01 within the ADL nucleus DAZNUCA. This may be achieved by specifying an extra statement for the linkage-editor input:

```
INCLUDE USRLOAD(DAZUEX01)
```

See the section *ADL User Exit DAZUEX01* later in this documentation for more details on the purpose and conventions for the user exit `DAZUEX01`.

Step 9

Create the executable ADL Interfaces CICS module by running the link editor and specifying the following link edit directives:

```
INCLUDE CICLOAD(DFHEAI)
INCLUDE CICLOAD(DFHEAI0)
INCLUDE ADLLOAD(DAZPARM)
INCLUDE ADLLOAD(DAZCAPRI)
INCLUDE ADLLOAD(DAZCIFP)
INCLUDE ADLLOAD(DAZCLUB)
INCLUDE ADLLOAD(DAZCRQH)
INCLUDE ADLLOAD(DAZCDOKE)
INCLUDE ADLLOAD(DAZCONSI)
INCLUDE ADLLOAD(DAZAXES)
INCLUDE ADLLOAD(DAZSERV)
INCLUDE ADLLOAD(DAZINICB)
INCLUDE ADLLOAD(DAZCDPOS)
INCLUDE ADLLOAD(DAZDREIN)
INCLUDE ADLLOAD(DAZDEBUG)
INCLUDE ADLLOAD(DAZSTUB)
INCLUDE ADLLOAD(DAZZAP)
ENTRY   DAZPARM
NAME DAZNUCC(R)
```

You may use the JCL in the ADL source library member `ADLINS9C` as an example.

Note: The ADL load library contains modules which are CICS release dependent. The members in the standard load library are for CICS TS 3.1 and CICS TS 3.2. When you run under CICS TS 2.3 or below, the ADL.LC23 library must be concatenated in front of the ADL load library.

Note: Whenever you want the user written user exit (`DAZUEX01`) to become active under CICS, you must, in addition, include your user exit `DAZUEX01` within the ADL Interfaces CICS nucleus, `DAZNUCC`. This may be achieved by specifying an extra statement for the linkage-editor input:

```
INCLUDE USRLOAD(DAZUEX01)
```

See the section ***ADL User Exit DAZUEX01*** later in this documentation for more details on the purpose and conventions for the user exit `DAZUEX01`.

Step 10

Create the executable ADL batch region controller by running the linkage-editor and specifying the following link-edit directives:

```
ENTRY DAZBIFP
INCLUDE ADLLOAD(DAZBIFP)
INCLUDE ADLLOAD(DAZBRQH)
INCLUDE ADALOAD(ADAUSER)
INCLUDE USRLOAD(DAZUEX01)
NAME DAZIFP(R)
```

You may use the JCL in the ADL source library member `ADLINS10` as an example. The ADL load library, the Adabas load library and the user load library must be defined with the file names `ADLLOAD, ADALOAD` and `USRLOAD`, respectively.

Note: This step has to be performed only if you want a user written user exit (`DAZUEX01`) to become active in batch. See the section ***ADL User Exit DAZUEX01*** later in this documentation for more details on the purpose and conventions for this user exit.

Creating the Consistency Front-Ends (Steps 11 - 12)

Step 11

Note: This step is only required if you plan to use the ADL Consistency Interface in batch.

Assemble and link-edit the ADL substitute for the Adabas link module in batch, `ADALNK`. You may use the JCL in the ADL source library member `ADLINS11` as an example. You need to perform this step only if you plan to use the ADL Consistency Interface. See the section *Batch Installation and Operation* in the *ADL Interfaces* documentation for more information on how to install the ADL Consistency Interface.

The ADL source library member to be assembled is `DAZLNKO`. Before the assembly, you will have to customize the assembler local variable `&ADALNK` in the ADL source library member `DAZLNKO`. `&ADALNK` specifies the new name chosen for the original Adabas link module which must be renamed. It is defaulted to `"ADAOLK"`.

Note that the ADL source library, the Adabas source library and the system macro libraries must be concatenated in the given order.

If you want to make use of the table of converted Adabas files (`DAZTCF`) you have to perform the steps described in section *Batch Installation and Operation* in the *ADL Interfaces* documentation. Assemble `DAZTCF` before link-editing `ADALNK`. The assembly of `DAZTCF` is included in the sample `JCL ADLINS11`. Add the following statement to the linkage-editor input for the Adabas batch link module:

```
INCLUDE ADLLOAD(DAZTCF)
```

For Adabas version 8 and above, you must additionally link the Adabas link-globals-table LNKGBLS to the ADL substitute:

```
INCLUDE ADALOAD(LNKGBLS)
```

Step 12

This step is only required if you plan to use the ADL Consistency Interface under CICS. See the section *CICS Installation and Operation* in the *ADL Interfaces* documentation for more information on how to install the ADL Consistency Interface.

Add the following statement to the linkage-editor input for the Adabas link module under CICS or, for Adabas version 8, to the linkage-editor input for the Adabas globals module:

```
INCLUDE ADLLOAD(ADLEXITB)
```

If you want to use the table of converted Adabas files, assemble the `DAZTCF` table and include this also:

```
INCLUDE ADLLOAD(DAZTCF)
```

Note: The parameter `LUSAVE` in the Adabas link source member `LNKOLSC` must have a value of at least 72 bytes. For Adabas Version 8 and above set the “ADL” parameter in the LGBLSET macro to “YES”.

4 z/VSE Installation Tape

The Adabas Bridge for DL/I installation tape is a standard-label tape. The Report of Tape Creation that accompanies the installation tape lists the volume serial number, tape density, media type, data sets and sequence numbers. The tape is compatible with Software AG's System Maintenance Aid (SMA). Refer to the SMA documentation for more details.

The ADL installation tape for systems operating under the IBM z/VSE Operating System contains four files as described in the table below. The first file has been unloaded to tape using the `IBM LIBR BACKUP` utility, and the other files have been copied by the `IBM IEBGENER` utility.

In this table, *vrs* indicates the current version, release and system maintenance (SM) level as indicated on the Report of Tape Creation, for example *123* for Version 1, Release 2 and SM 3.

Data Set	DSN	Description
1	ADL*vrs*.LIBR	The ADL Core Image Library, containing the ADL core image load modules. The ADL Relocatable Library, containing ADL relocatable load modules used to link edit executable ADL modules. The ADL Source Library, containing JCS examples for installing the ADL and converting DL/I data bases, as well as macros for reassembling DBDs and PSBs, and source code for sample data base unload programs.
2	ADL*vrs*.SYSF	An unloaded Adabas file for use as the ADL directory file.
3	ADL*vrs*.INPL	An unloaded Natural library containing the ADL Online Services and the ADL Installation Verification Package.

The following sections describe the libraries in more detail.

This chapter covers the following topics:

ADL Libraries

The following libraries are described in more detail below:

- ADL Core Image Library

- ADL Relocatable Library

ADL Core Image Library

When loaded from the installation tape, the ADL Core Image Library contains executable ADL load modules. The table below gives the names and descriptions of the modules.

Module	Description
DAZCDUMP	CICS - Write ADL tables on dump file.
DAZCEND	CICS - ADL shut-down program (PLT).
DAZCICS	CICS control program.
DAZCINF	CICS - Return status of ADL.
DAZCINIT	CICS - ADL start-up program (PLT).
DAZCTOFF	CICS - Switch off ADL trace.
DAZCTON	CICS - Switch on ADL trace.
DAZELORE	Establish Logical Relationships utility.
DAZIFP	Batch interface program.
DAZMIX	Batch mixed mode module.
DAZREFOR	Reformat utility.
DAZSYNC	CICS - ADL task-related user exit.
DAZUNDLI	Data Base Unload utility (automated procedure).
DAZZLER	Program to test DL/I calls in batch.

ADL Relocatable Library

When loaded from the installation tape, the ADL Relocatable Library contains the relocatable load modules given in the table below. These are used during installation to create executable ADL load modules.

Module	Description
DAZAXES	ADL nucleus module containing the logic for generating and issuing Adabas calls.
DAZBDOKE	Batch doorkeeper for the ADL Consistency Interface.
DAZBENT	Batch entry point module.
DAZBIFP	Batch interface program (object module).
DAZBRQH	Batch request handler.
DAZCAPRI	CICS application program interface.
DAZCCGEN	ADL nucleus module for the CBC utility.
DAZCCOUT	ADL nucleus module for the CBC utility.
DAZCCSUB	ADL nucleus module for the CBC utility.

DAZCDOKE	CICS doorkeeper for the ADL Consistency Interface.
DAZCDPOS	ADL nucleus module containing the logic for retrieval calls and for maintaining positional information.
DAZCIFP	CICS interface program.
DAZCLUB	CICS – Allocate local user blocks.
DAZCONSI	Environment independent routines of the ADL Consistency Interface.
DAZCRQH	CICS request handler.
DAZDEBUG	ADL nucleus module containing tracing and debugging routines.
DAZDREIN	ADL nucleus module containing the logic for `DELETE, REPLACE` and `INSERT` calls.
DAZEXEC	ADL nucleus module containing the logic for the `EXEC` command precompiler.
DAZINICB	ADL nucleus module containing the logic for initializing internal control blocks.
DAZLANP	ADL nucleus module containing the logic for the language processor.
DAZLIBAT	Language interface for batch mode.
DAZLICID	Language interface for CICS.
DAZSERV	ADL nucleus module containing general purpose service routines.
DAZSTUB	CICS — Stub for the ADL task-related user exit.
DAZSYXTB	ADL nucleus module containing the `EXEC` command syntax.
DAZUEX06	Adabas User Exit 6.
DAZUEXMI	Adabas User Exit 6 for migration to ADL 2.3.
DAZZAP	ADL nucleus module containing zaps for nucleus routines.

ADL Source Library

When loaded from the installation tape, the ADL Source Library contains:

- Macros for creating an ADL parameter module;
- Macros for creating Adabas User Exit 6 extensions;
- Macros for creating the CICS runtime control tables;
- Macros for assembling DBDs and PSBs (substitutes for the original `DL/IDBDGEN` and `PSBGEN` macros);
- Macros for assembling the application control table (substitute for the original `DL/I DLZACT` macro);
- Source code for sample unload programs;
- Source code for performing ADL functions under CICS;
- Source code for the ADL supplied Adabas link module substitutes;
- Sample JCS.

The table below lists the other macros contained in this library.

Member	Description
DAZLDT	Macro used to create entries for logical `DBID/FNR` assignments in the ADL parameter module. This macro is provided for compatibility with ADL 2.2 only.
DAZTCF	Macro used to create the table of converted files for the Adabas link module substitute in batch.
DAZPARM	Macro used to create the ADL parameter module.
ZFNR	Macro used to create Adabas User Exit 6 extensions.
ZPCK	Macro used to create Adabas User Exit 6 extensions.
ZREC	Macro used to create Adabas User Exit 6 extensions.
ZSEG	Macro used to create Adabas User Exit 6 extensions.
ZSEX	Macro used to create Adabas User Exit 6 extensions.
ZVCK	Macro used to create Adabas User Exit 6 extensions.
MGPSTIN	Macro used to create the `DAZPSB` table.
MGPSTEN	Macro used to create the `DAZPSB` table.
MGPSTFI	Macro used to create the `DAZPSB` table.
DLZACT	Macro used to create a `DAZPSB` or `DAZACT` table.
DBDMAC	Macro used to create the `DAZDBD` table.
BUFMAC	Macro used to create the `DAZBUF` table.

The following table lists the source members used during the assembly of the ADL supplied Adabas link module substitutes:

Member	Description
DAZLNKD	Source member to be assembled as the ADL supplied Adabas link module substitute in batch.
DAZLNK	Operating system independent part of the ADL supplied Adabas link module substitute in batch.
LNKDOS	Operating system dependent part of the ADL supplied Adabas link module substitute in batch.

The following table lists the sample JCS streams for the installation and conversion processes:

Member	Description
ADLINS1.J	JCS for loading the ADL libraries from tape.
ADLINS2.J	JCS for loading the ADL directory file from tape.
ADLINS2A.J	JCS for updating an existing ADL directory file.
ADLINS3.J	JCS for an initial program load of the ADL Online Services and the ADL supplied Natural subprograms `ADLERROR` and `ADLACTIV` and an initial program load of the Natural programs for the ADL Installation Verification Package .

Member	Description
ADLINS4.J	JCS for creating an ADL parameter module.
ADLINS5U.J	JCS for creating an executable `ADL CBC` utility.
ADLINS6P.J	JCS for creating an executable ADL precompiler module.
ADLINS7B.J	JCS for creating an executable ADL batch module.
ADLINS8A.J	JCS for creating an executable ADL Consistency Interface module for batch.
ADLINS9C.J	JCS for creating an executable `ADL CICS` module.
ADLCSD	Sample CICS system definition file.
ADLCTG1.J	JCS for assembling the `DAZPSB` table.
ADLCTG2.J	JCS for running the `DAZSHINE` utility.
ADLCTG3.J	JCS for assembling the `DAZBUF` table.
ADLCTG4.J	JCS for assembling the `DAZDBD` table.
ADLCFCT.J	JCL for precompile, assembly and link-edit of `DAZCFCT`.
ADLTCF.J	JCS for assembly of the `DAZTCF` table.

Note the naming conventions for the JCL examples:

```
ADLxxxnn    General sample JCL
IVPxxxnn    JCL for the Installation Verification package
```

where *xxx* represents one of the following suffixes

INS	for the installation steps (section ***z/VSE Installation*** in this documentation);
DPC	for the `DBD/PSB` conversion steps (section *ADL Conversion Utilities for DBDs and PSBs* in the *ADL Conversion* documentation ;
DBC	for the data base conversion steps (section *ADL Data Conversion Utilities* in the *ADL Conversion* documentation;
CTG	for the CICS table generation (section *Generating the Runtime Control Tables* in the *ADL Interfaces* documentation);
TCF	for the generation of the converted Adabas file table (see the section *Batch Installation and Operation* in the *ADL Interfaces* documentation);

and *nn* is the number of the step.

The following table lists the remaining members in the Source Library.

Member	Description
$INFO$	Information about the current ADL release.
ADLEX06	ADACMP User Exit 6 skeleton, for changing the layout of an ADL file.
ADLIMEX	Sample user exit routine for index maintenance.
ADLXPCn	COBOL sources for the ADL IVP.
ADLXPIn	Input streams for the COBOL programs for the ADL IVP.
ADLXPAn	Assembler sources for the ADL IVP to run under CICS.
ADLXPDn	DAZZLER input streams for the ADL IVP.
COURSEDB	Physical DBD definitions for the ADL IVP.
COURSEL	Logical DBD definitions for the ADL IVP.
COURSUNL	PSB definitions for the ADL IVP (DAZUNDLI utility).
DAZUNLOD	Source of the sample unload program.
DAZREFOR	Source of the sample reformat program.
DAZCFCT	Source of the sample program to perform ADL functions under CICS.
DAZEISTG	DSECT used by DAZCFCT
INSTDB	Physical DBD definitions for the ADL IVP.
INSTIDX	Primary Index DBD definition for the ADL IVP.
INSTL	Logical DBD definitions for the ADL IVP.
INSTUNL	PSB definitions for the ADL IVP (DAZUNDLI utility).
INSTELO	PSB definitions for the ADL IVP (DAZELORE utility).
IVPINFO	Abbreviations used by the ADL Installation Verification Package (IVP).
IVPARUN	ADARUN cards for the ADL IVP.
IVPCOB	Execute a COBOL batch program of the ADL IVP.
MAINIDX	Primary Index DBD definition for the ADL IVP.
SCHOOL	PSB definitions for the ADL IVP.
STUDIDX	Secondary Index DBD definition for the ADL IVP.

ADL Directory File

The unloaded Adabas Directory file on the installation tape was created by the Adabas Unload utility, ADAULD. At initialization, the file contains the texts of the ADL error messages. Later, it will also be used for storing the DBDs and PSBs for ADL and any checkpoint information.

ADL Natural Programs

This file contains the unloaded Natural programs comprising the ADL Online Services, together with the ADL supplied Natural subprograms `ADLERROR` and `ADLACTIV` and the ADL Installation Verification Package. The `ADLERROR` subprogram may be used by Natural applications to retrieve the comprehensive error messages of the ADL Consistency Interface. The `ADLACTIV` subprogram may be used by Natural applications to verify whether the ADL Consistency Interface is active or not.

The files were created with the Natural Object Handler (SYSOBJH).

5 z/VSE Installation

This chapter covers the following topics:

Overview

This chapter describes the steps necessary to install the Adabas Bridge for DL/I (`ADL`) in a z/VSE environment. After performing these steps, you will be able to run the ADL data base conversion utilities, to use the ADL Online Services and to operate the ADL Interfaces for DL/I and Adabas calls.

For easy reference, all installation steps are summarized below. In general, all steps must be performed in all environments. Exceptions are clearly marked as such in the detailed description of the individual steps.

Step	Description
Step 1	Load the ADL libraries to disk.
Step 2	Load the ADL directory file.
Step 3	Load the ADL Online Services and the ADL Installation Verification Package.
Step 4	Create the ADL parameter module.
Step 5	Create the executable ADL data conversion module.
Step 6	Create the executable ADL precompiler module.
Step 7	Create the executable module for the ADL CALLDLI Interface in batch.
Step 8	Create the executable module for the ADL Consistency Interface in batch.
Step 9	Create the executable module for the ADL Interfaces under CICS.
Step 10	Create the executable ADL batch region controller.
Step 11	Create the ADL substitute for the Adabas link module in batch.
Step 12	Link-edit the Adabas link module under CICS with the ADL exit `ADLEXITB`.

Note: If you use Software AG's System Maintenance Aid (SMA), steps 1 to 9 are performed by SMA.

Initial Load of the ADL Libraries (Step 1)

Load the ADL libraries to disk using the JCS given as an example below.

Library Space Requirements (z/VSE)

The following table gives an estimate of how much space is needed for various device types (primary tracks, directory tracks).

Library	3390
SL	40, 1
RL	80, 1
CL	80, 1

Copying the Tape Contents to a z/VSE Disk

If you are using SMA, refer to the *System Maintenance Aid* documentation (included in the current edition of the Natural documentation CD).

If you are *not* using SMA, follow the instructions below.

This section explains how to:

- Copy dataset COPY.JOB from tape to disk.
- Modify this dataset to conform with your local naming conventions.

The JCL in this member is then used to copy all datasets from tape to disk.

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

After that, you will have to perform the individual install procedure for each component.

Step 1 - Copy Dataset COPYTAPE.JOB from Tape to Disk

The dataset COPYTAPE.JOB (File 5) contains the JCL to unload all other existing datasets 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
/*
/&amp;
* $$ EOJ                                                                ↩
```

Where:

NNN is the tape address
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 datasets from the tape to your disk.

Initial Load of the ADL Directory File (Step 2)

Load the ADL Directory File. This file is a standard Adabas file unloaded with the `ADAULD` utility of Adabas. You may use the JCL in the ADL source library member `ADLINS2.J` as an example.

If ADL is already installed at your site and you want to continue to use the existing ADL directory file, you may simply delete the obsolete error messages from the existing Directory File and load the new messages from the unloaded file provided on the installation tape. You may use the JCL in the ADL source library member `ADLINS2A.J` as an example.

Initial Program Load of the ADL Natural Programs (Step 3)

Load the ADL Online Services and the DDM for the ADL Directory File, ADB-CONTROL. The INPL file on the installation tape was created with the Natural SYSOBJH utility and is compatible with Natural version 4.1 and upwards. You may use the JCL in the ADL source library member ADLINS3.J as an example.

Note that the INPL file on the ADL installation tape contains a DDM for the ADL Directory File and the object modules of the library SYSADL with the ADL-supplied subprograms ADLERROR and ADLACTIV. These subprograms may be used by Natural applications operating under the ADL Consistency Interface.

The same step loads also the ADL Installation Verification Package (SYSADLIV) and the related DDMs.

Creating the ADL Parameter Module (Step 4)

Create the ADL parameter module (DAZPARM) by assembling the DAZPARM macro. This procedure is common to both z/OS and z/VSE and is described in the section *The ADL Parameter Module* .

You may use the JCS in the ADL Source Library member ADLINS4.J as an example.

Creating the ADL Executable Modules (Steps 5 - 10)

Step 5

Create the executable ADL CBC utility module by running the link editor and specifying the following link edit directives:

```
PHASE DAZNUCU,*,NOAUTO
INCLUDE DAZPARM
INCLUDE DAZCCGEN
INCLUDE DAZCCOUT
INCLUDE DAZCCSUB
INCLUDE DAZAXES
INCLUDE DAZSERV
INCLUDE DAZDEBUG
INCLUDE DAZZAP
ENTRY DAZPARM
```

You may use the JCS in the ADL Source Library member ADLINS5U.J as an example.

Step 6

This step need only be performed where one or more of the application programs to be converted uses the `HLPI`.

Create the executable ADL precompiler module by running the link editor and specifying the following link edit directives:

```
PHASE DAZNUCP,*,NOAUTO
INCLUDE DAZPARM
INCLUDE DAZEXEC
INCLUDE DAZSERV
INCLUDE DAZLANP
INCLUDE DAZSYXTB
INCLUDE DAZDEBUG
INCLUDE DAZZAP
ENTRY DAZPARM
```

You may use the JCS in the ADL Source Library member `ADLINS6P.J` as an example.

Step 7

Create the executable `ADL CALLDLI` Interface batch module by running the link editor and specifying the following link edit directives:

```
PHASE DAZNUCB,*,NOAUTO
INCLUDE DAZPARM
INCLUDE DAZBENT
INCLUDE DAZAXES
INCLUDE DAZSERV
INCLUDE DAZINICB
INCLUDE DAZCDPOS
INCLUDE DAZDREIN
INCLUDE DAZDEBUG
INCLUDE DAZZAP
ENTRY DAZPARM
```

You may use the JCS in the ADL Source Library member `ADLINS7B.J` as an example.

Step 8

Create the executable ADL Consistency Interface module for batch by running the linkage-editor and specifying the following link-edit directives:

```
PHASE DAZNUCA,*,NOAUTO
INCLUDE DAZPARM
INCLUDE DAZAXES
INCLUDE DAZBDOKE
INCLUDE DAZBRQH
INCLUDE ADAUSER
INCLUDE DAZCDPOS
INCLUDE DAZCONSI
INCLUDE DAZDEBUG
INCLUDE DAZDREIN
INCLUDE DAZINICB
INCLUDE DAZSERV
INCLUDE DAZZAP
ENTRY DAZPARM
```

The `ADAUSER` object module must be included from a valid Adabas relocatable library.

Note: When you want the user written user exit (`DAZUEX01`) to become active in batch, you must, in addition, include your user exit `DAZUEX01` within the ADL nucleus `DAZNUCA`. This may be achieved by specifying an extra statement for the linkage-editor input:

```
INCLUDE DAZUEX01
```

See the section ***ADL User Exit DAZUEX01*** later in this documentation for more details on the purpose and conventions for the user exit `DAZUEX01`.

You may use the JCS in the ADL source library member `ADLINS8A.J` as an example on how to link-edit the ADL nucleus `DAZNUCA`.

Step 9

Create the executable ADL Interfaces CICS module by running the link editor and specifying the following link edit directives:

```
PHASE DAZNUCC,*,NOAUTO
INCLUDE DFHEAI
INCLUDE DFHEAI0
INCLUDE DAZPARM
INCLUDE DAZCAPRI
INCLUDE DAZCIFP
INCLUDE DAZCRQH
INCLUDE DAZCDOKE
INCLUDE DAZCLUB
INCLUDE DAZCONSI
INCLUDE DAZAXES
INCLUDE DAZSERV
INCLUDE DAZINICB
INCLUDE DAZCDPOS
INCLUDE DAZDREIN
INCLUDE DAZDEBUG
INCLUDE DAZSTUB
INCLUDE DAZZAP
ENTRY DAZPARM
```

You may use the JCS in the ADL Source Library member `ADLINS9C.J` as an example.

Note: When you want the user written user exit (`DAZUEX01`) to become active under CICS, you must, in addition, include your user exit `DAZUEX01` within the ADL Interfaces CICS nucleus, `DAZNUCC`. This may be achieved by specifying an extra statement for the linkage-editor input:

```
INCLUDE DAZUEX01
```

See the section ***ADL User Exit DAZUEX01*** later in this documentation for more details on the purpose and conventions for the user exit `DAZUEX01`.

Step 10

Create the executable ADL batch region controller by running the linkage-editor and specifying the following link-edit directives:

```
ACTION CLEAR
PHASE DAZIFP,*,NOAUTO
INCLUDE DAZBIFP
INCLUDE DAZBRQH
INCLUDE ADAUSER
INCLUDE DAZUEX01
ENTRY DAZBIFP
```

You may use the JCS in the ADL source library member `ADLINS10.J` as an example. The ADL load library, the Adabas load library and the user load library must be defined in a "`LIBDEF PHASE,SEARCH=`" statement in the given order.

It is only necessary to perform this step, if you want a user written user exit (`DAZUEX01`) to become active in batch. See the section ***ADL User Exit DAZUEX01*** later in this documentation for more details on the purpose and conventions for this user exit.

Creating the Consistency Front-Ends (Steps 11 - 12)

Step 11

This step is only required if you plan to use the ADL Consistency Interface in batch.

Assemble and link-edit the ADL substitute for the Adabas link-module in batch, `ADALNK`. You may use the JCS in the ADL source library member `ADLINS11.J` as an example. You need to perform this step only, if you plan to use the ADL Consistency Interface. See the section *Batch Installation and Operation* in the *ADL Interfaces* documentation for more information on how to install the ADL Consistency Interface.

The ADL source library member to be assembled is `DAZLNKD`. Before the assembly, you will have to customize the assembler local variable `&ADALNK` in the ADL source library member `DAZLNKD`. `&ADALNK` specifies the new name chosen for the original Adabas link module. It is defaulted to `"ADAOLK"`.

Note that the ADL source library, the Adabas source library and the system macro libraries must be concatenated in the given order.

If you want to make use of the table of converted Adabas files (`DAZTCF`) you have to perform the steps described in section *Batch Installation and Operation* in the *ADL Interfaces* documentation. Assemble `DAZTCF` before link-editing of `ADALNK`. The assembly of `DAZTCF` is included in the sample JCL `ADLINS11`.

Add the following statement to the linkage-editor input for the Adabas batch link module:

```
INCLUDE DAZTCF
```

For Adabas version 8 and above, you must additionally link the Adabas link-globals-table LNKGBLS to the ADL substitute:

```
INCLUDE LNKGBLS
```

Step 12

This step is only required if you plan to use the ADL Consistency Interface under CICS. See the section *CICS Installation and Operation* in the *ADL Interfaces* documentation for more information on how to install the ADL Consistency Interface.

Add the following statement to the linkage-editor input for the Adabas link module under CICS:

```
INCLUDE ADLEXITB
```

If you want to use the table of converted Adabas files, assemble the `DAZTCF` table and include this also:

```
INCLUDE DAZTCF
```

Note: The parameter `LUSAVE` in the Adabas link source member `LNKOLSC` must have a value of at least 72 bytes.

6 ADL Parameter Module

This chapter covers the following topics:

Overview

The Adabas Bridge for DL/I requires certain installation-dependent information. This information is basically supplied to ADL by means of a parameter module. Most of the parameters are called static, that is, they can be defined in the ADL parameter module only. In addition, for batch mode, the possibility exists to define certain parameters dynamically to the ADL batch region controller, `DAZIFP`, and to the ADL substitute for the Adabas link module in batch, `ADALNK`.

The ADL parameter module is created by the assembly and link-edit of the `DAZPARM` macro. All parameters for this macro are keyword parameters.

Finally, the ADL parameter module will be link-edited together with one of the five ADL nuclei (`DAZNUCA, DAZNUCB, DAZNUCC, DAZNUCP` and `DAZNUCU`). You may, for each individual nucleus, define a different ADL parameter module depending on the requirements.

Note: With ADL 2.3 the logical file numbers have been replaced by logical IDs. Therefore the `DAZLDT` macro which was part of the ADL 2.2 parameter module has become obsolete. For compatibility reasons DBDs converted with ADL 2.2 or before may still use this macro. Its meaning and usage can be found in the ADL 2.2 documentation.

List of Parameters for the ADL Parameter Module

Keyword	Explanation	Possible values	Default
ACTSF	(z/VSE only) A two-character suffix for the name of the ACT table module.		Blank
ADANAME	(CICS only) The name of the Adabas link module. If you want to use the ADL Consistency Interface, this parameter must be the same as the Natural parameter `ADANAME`. Refer to the section *CICS Installation and Operation* in the *ADL Interfaces* documentation for more details.		ADABAS
ADAUSR	(CICS only) Specifies whether the `ADL CALLDLI` interface should generate a special Adabas User ID under CICS. The Adabas user Id is only generated when the ADL Consistency interface is active. For more information see the section CICS Installation and Operation in the *ADL Interfaces* documentation.	YES or NO	YES
BUFSF	A two-character suffix for the name of the buffer table module. See the section *Generating the Runtime Control Tables* in the *ADL Interfaces* documentation for details.		Blank

Keyword	Explanation	Possible values	Default
CHKPMSG	Specifies where the checkpoint message is to be written, if at all.	1 - Message is written to DAZOUT1 2 - Message is written to DAZOUT2 NO - No checkpoint message is written	1
CPID	(DAZIFP parameter only). The checkpoint ID from which the application is to be restarted (for programs using symbolic checkpoints only).	Any 8-character string.	None
CRSIZ	The size (in bytes) of the area used by ADL for retrieving segment occurrences when processing cascaded deletes. The size of this area can be determined as follows: (n + 1) * 8 where "n" is the maximum number of levels for which the delete applies. The default size should be sufficient in most cases.	0 - 999	512 bytes
DBD	The size (in kilobytes) of the ADL's DBD ICB buffer. This buffer is only allocated and used in application batch runs, when it stores the DBD internal control blocks.	0 - 999	16 KB
DBDSF	A two-character suffix for the name of the DBD table module. See the section *Generating the Runtime Control Tables* in the *ADL Interfaces* documentation for details.		Blank
DBID	Adabas data base ID for the ADL directory file. This parameter is mandatory.	1 - 32767	
DUO	(z/OS batch only) Support of CA-DUO under z/OS batch operation. DUO=YES has to be specified for programs which are link-edited by the CA-DUO linkage editor. Note that the program has to be linked with the ADL language interface module DAZLIBAT and not with DUODLZLI. Use the 'NCAL' parameter for DUOLINK. The entry point for COBOL programs is DLITCBL. The application runs under the control of DUO, which itself is called by DAZIFP. DUO=NO has to be specified in any other case. For further information see the CA documentation in the *CA-DUO USER GUIDE*.	YES or NO	NO
EBUF	The size (in kilobytes) of ADL's ECB buffer. The size allocated for this area depends on the size of the PSB used and the DBDs referenced.	0 - 99	8 KB
ET	The number of times a different root segment occurrence may be accessed before an Adabas ET command is issued. This parameter is only of interest for batch programs running as BMP,	1 - 999999, NO	1

Keyword	Explanation	Possible values	Default
	MPS or SDB jobs. ET=NO may be specified in cases where no automatic ETs should be issued. For further information, see the section *Recovery and Restart Procedures* in the *ADL Interfaces* documentation. Under CICS, ADL enqueues every accessed DB record. If no update has taken place, ADL releases the record as soon as the next one is accessed. The ET parameter specifies the record number at which this release starts. Thus, ET allows the number of Adabas 'RI' calls to be decreased. Note that the records which have been accessed before the specified number is reached remain in hold status until the next explicit SYNCPOINT call or until the end of the task. If ET=NO is specified under CICS, ADL will not enqueue a record as long as it is accessed without hold (i.e. with GU, GN, GNP). As soon as it is accessed with a hold command (i.e. with GHU, GHN, GHNP) ADL enqueues the record and treats it as if ET=1 was specified.		
FBSIZ	The size (in bytes) of the area allocated to the Adabas format buffer. The maximum length of the format buffers created by ADL depends on the depth of the hierarchy and the number of secondary indices defined for a particular source segment. The default value is generally sufficient for a segment at level 15 having 20 secondary indices.	0 - 999	128 bytes
FDT	(Batch only). The size (in kilobytes) of the file description table used internally by the ADL Consistency Interface.	4 - 999	4 KB
FNR	Adabas file number for the ADL directory file. This parameter is mandatory.	1 - 32767	
FSTAC	The size (in bytes) for the format buffer stack used internally by the ADL Consistency Interface.	16 - 9999	800 bytes
FX	(z/VSE only) Used to specify the input data set for the Print utility for printing the Trace routine or for the ADL precompiler. The syntax of the parameter is as follows: `FX=([x],[y],[z])` where x is Logical unit y is Record length in bytes z is Block size in bytes	x: 1 - 99 y: 0 - 99999 z: n * y ≤ 99999 where n is any positive integer.	x: 14 y: 132 bytes z: 1320 bytes
IBSIZ	The size (in bytes) of the area allocated to the Adabas ISN buffer. This is needed for Adabas calls using PREFETCH, i.e. for segments for which no Z0 field is available (see the section *DBD/PSB Conversion*) and for which either an "INSERT LAST" or a "GET NEXT LAST" DL/I call is issued. The number of Adabas calls needed to retrieve the last segment occurrence can be influenced by changing the size of the ISN buffer. When automatic ETs are to be issued by ADL (see the ET parameter), a so-called retain	0 - 32767	1024 bytes

Keyword	Explanation	Possible values	Default
	ISN list is created for every PCB. It contains the file numbers and ISNs of the last accessed root segment and of those dependencies, which should be kept in hold status. The size of one retain ISN list area is: `IBSIZ / (number-of-PCBs - 1)`		
IMSY	(IMS/TP only) Indicates whether or not IMS/TP sync point/Adabas ET synchronization is to be done. Every GU call on the first I/O PCB triggers an Adabas ET call and an IMS/TP sync point (in this order). As there is a gap between the two synchronization points, a synchronization problem may occur between the data stored in the Adabas data base and the IMS/TP message queue. When this parameter is set to "YES", such a situation is recognized and the application will be terminated.	YES or NO	NO
LANG	(DAZIFP parameter only). Specifies the language of the application program to be executed. If this parameter is not given, the language defined in the PSB is used.	ASSEM, ASM, ASSEMBLER, COBOL, CBL, FORTRAN, PL/I, PL/1, PLI, PL1, RPG, NATURAL, NDL.	Language specified in the PSB.
LCS	(Batch only). The size (in kilobytes) of the “last call save area” (LCS). Refer to the section *Performance Considerations* in the *ADL Interfaces* documentation for more details. If "NO" is specified, the LCS will not be used.	0-999, NO	NO
LOAD	(Batch only). Indicates, how an ISRT call against a PCB with PROCOPT=L is to be treated. When this parameter is set to "DIRECT", the ISRT is translated into an Adabas 'N1' call, the data is directly inserted into the Adabas file. When it is set to "UTILITY" the data is written into the sequential file DAZOUT3 (DAZOT3D) for z/OS (z/VSE). This file has the same layout as the one produced by the ADL utilities DAZUNDLI or DAZREFOR. The data can be loaded to Adabas by an initial load as described in the section *Converting Data - Load* in *ADL Data Conversion Utilities* in the *ADL Conversion* documentation. For more information on the LOAD parameter see the section *Performance Considerations* in the *ADL Interfaces* documentation.	DIRECT or UTILITY	DIRECT
MFT	(DAZIFP parameter only). Specifies the Multifetch Table (MFT). This parameter should only be used when the Adabas Multifetch facility is active. It defines the ISN lower limit value for L3 calls against specific PCB/SENSEG combinations. Thus it specifies, how many records should be returned by Multifetch. A value '0' means, that the maximum number of records is returned. This number is determined by the size of the ISN buffer and the record buffer, i.e., by the ADARUN parameters PREFSBL and	range for value of the ISN lower limit field: 1 - 32767	

Keyword	Explanation	Possible values	Default
	PREFTBL. '0' is the default value for all not specified PCB/SENSEG combinations. A value '1' means, that the multifetching of the corresponding PCB/SENSEG combination will be minimized, i.e., 2 records per read activity. Refer to section *Performance Considerations* in the *ADL Interfaces* documentation for more details. Note, that the MFT and RBE parameters are mutually exclusive. If you specify the MFT parameter, you must not use the ADARUN parameter PREFNREC. The syntax of the parameter is as follows: MFT=(MFT-entry1,MFT-entry2,...). A maximum number of 128 entries is allowed. An empty list 'MFT=( )' is possible also. A MFT-entry has the following layout: (pcb,senseg,value) where: pcb is the number of the PCB senseg is the number of the sensitive segment and value is the value of the ISN lower limit field.		
NUMEXR	(Batch only). The maximum number of secondary indices for which an index maintenance exit routine is supplied (EXTRTN keyword in the DBD definition).	0-910	0
NUMLR	The maximum number of logical relationships in which the DBDs referenced by a PSB are involved. This parameter is used to reserve working space during initialization of the DBDs and PSBs. Note that all logical relationships must be included in the count, even though not all of them may be referenced by a particular PSB.	1-99	16
NUMQS	The maximum number of qualification statements to be expected for a single DL/I call. This parameter is used to calculate the length of the buffer used to store the internal representation of the qualification statements of a DL/I call. The default value should be sufficient in most cases, otherwise a status code “AV” is received.	0-999	32
OPENRQ	Under CICS, specifies whether or not an Adabas "OP" call is to be issued on a scheduling call. This parameter is related to the OPENRQ parameter for ADARUN. See the *Adabas Operations* documentation for a full explanation of this parameter. OPENRQ=YES must be specified for ADL if it was specified for ADARUN.	YES - Adabas "OP" required. NO - Adabas "OP" not required.	NO
PARM	Whether or not dynamic overwrite parameter will be read from the file DAZIN1 (z/OS) or the logical unit SYSIPT(z/VSE) during initialization of the ADL substitute for the Adabas link module in batch.	YES - read dynamic parameters. NO - do not read dynamic parameters.	NO

Keyword	Explanation	Possible values	Default
PASSWRD	The Adabas password. When specified, this password will be used by the ADL for every Adabas call.	1 - 8 characters.	
PBUF	The size (in kilobytes) of the ADL buffer used by the precompiler. The size allocated for this area depends on the complexity of the `EXEC` commands used.	0 - 99	8 KB
PLILE	(z/VSE batch only). This parameter is used for batch z/VSE programs written in PL/1.	YES - PL/1 is using the LE/VSE language environment. NO - The PL/1 LE/VSE language environment is not used.	YES
PLINTWA	(CICS only). Determines whether or not the Adabas call parameter list is passed on to the `CICS TWA`.	YES - parameter list is in TWA. NO - parameter list is pointed to by R1.	YES
PR	(z/VSE only). The number of logical printers available. The CBC utility produces two separate printer output files. You may send these to two different logical printers, if these are available, by setting this parameter to `PR=2`. Specifying `PR=1` will cause the second printer output file to be stored on an intermediate disk file.	1,2	1
PSB	The size (in kilobytes) of ADL's `PSB ICB` buffer. This buffer is only allocated and used in application batch runs, when it stores the `PSB` internal control blocks.	0 - 999	16 KB
PSBSF	A two-character suffix for the name of the `PSB` table module used under CICS.		Blank
RBE	(`DAZIFP` parameter only). Specifies the record buffer extension (`RBE`) list. It allows increasing the record buffer length (`RBL`) for `L3` calls against specific `PCB/SENSEG` combinations. The increase of the `RBL` does not mean that the `RB` area is increased. Thus you will find garbage in the ADL trace after the 'real' end of the record buffer. This parameter should be used only, if the Adabas Prefetch facility is active. Refer to the section *Performance Considerations* in the *ADL Interfaces* documentation for more details. Note that the `RBE` and `MFT` parameters are mutually exclusive. The syntax of the parameter is as follows: `RBE=(RBE-entry1,RBE-entry2,...)` A maximum number of 128 entries is allowed. An empty list `` `RBE=()' `` is possible also. An `RBE`-entry has the layout	range for size by which the RBL will be increased: 0 - 32767	

Keyword	Explanation	Possible values	Default
	(pcb,senseg,size) where pcb is the number of the PCB senseg is the number of the sensitive segment and size is the size by which the RBL will be increased.		
RBSIZ	The size (in bytes) for the record buffer used internally by the ADL Consistency Interface and by the DAZSHINE utility.	3 - 9999	1024 bytes
RETRY	The number of times ADL tries to put into the hold status a record which is already held by another user. After the last try ADL will abend with the message ADL0145. RETRY=WAIT may be specified to let Adabas wait until the record has been released or a timeout occurs.	1 - 65535, WAIT	WAIT
SBSIZ	The size (in bytes) of the area allocated to the Adabas search buffer. The Adabas Bridge for DL/I never creates a search buffer greater than the default length given.	0 - 999	32 bytes
SDT	(Batch only) The size (in kilobytes) of the segment description table used internally by the ADL Consistency Interface.	4 - 999	4 KB
SQ	(z/VSE only) Used to specify the in/output data set for the Unload utility. The syntax of the parameter is as follows: SQ=([x],[y]) where x - logical unit y - Block size in bytes	x: 1 - 999 y: 0 - 99999	x: 13 y: 8196 bytes
STACK	The size (in kilobytes) of the ADL internal subroutine stack. The size allocated for this area depends largely on the type of DL/I calls issued. The default size should be sufficient in most cases.	1 - 18	5 KB
SVC	z/OS CICS ADL Installation SVC number.	200 – 255	none
TRACE	Activates the Trace facility and specifies what is to be traced. The syntax of the parameter is as follows: TRACE = ([t],[s],[a],[n],[m],[c],[u],[b]) The operands for this parameter and further details are explained in the section *Debugging Aids - ADL Trace Facility* in the *ADL Interfaces* documentation.		
UTI	Specifies the CBC utility work area. This buffer is only allocated and used in CBC utility runs. The syntax of the parameter is as follows: UTI=([x],[y]) where x - The size (in kilobytes) of the CBC utility work area.	x: 8 - 999 y: Y (yes) N (no)	x: 32 KB y: Y (yes)

Keyword	Explanation	Possible values	Default
	y - Specifies whether or not output control statements are to be generated by the CBC utility.		
VBSIZ	The size (in bytes) of the area allocated to the Adabas value buffer. ADL never creates a value buffer greater than the default length given.	0 - 999	256 bytes

Dynamic Overwrite Parameters

When running the ADL Interfaces in batch, some of the parameters specified for the `DAZPARM` macro may be dynamically overwritten. The dynamic overwrite parameters are set during the initialization of the ADL batch region controller, `DAZIFP`, or the ADL substitute for the Adabas link module in batch.

More details on how to specify these dynamic overwrite parameters are given in the section *Batch Installation and Operation* in the *ADL Interfaces* documentation . Also, there you will find a complete list of dynamic overwrite parameters for the `ADL CALLDLI` and the ADL Consistency Interface. The syntax and the meaning of the dynamic overwrite parameters is exactly the same as for the `DAZPARM` macro.

7 ADL Installation Verification Package

This chapter covers the following topics:

Introduction

The ADL Installation Verification Package (IVP) provides you with a full DL/I application environment. It can be used to verify the successful installation of the ADL. Moreover, when running through the steps outlined below, you will gain experience in the ADL concepts and the various ADL tools. By the way, if you do not yet have DL/I or Adabas knowledge, you will learn about the most important terms of the both database systems, and how ADL connect the both. If you are interested in more detailed information about these database systems, refer to the corresponding IBM or Software AG documentation.

The ADL IVP consists of the following parts:

- DBD and PSB definitions of the example database
- Sample JCL
- COBOL batch programs with input files
- Assembler online programs
- DAZZLER input streams
- Natural program sources
- DDM definitions

The DDM definitions are loaded into the DDM file during the ADL installation. At the same time, the Natural programs are loaded into the Natural library SYSADLIV. All other parts are in the ADL source library.

DL/I Terms

In DL/I, the database layout is described in a so-called *DBD* (*database description*). For each information type (like ‘COURSE’) there is one *SEGMENT (type)* definition, describing the corresponding data layout. Single data information (like 'Mathematics') is named ‘*SEGMENT occurrence*’. *FIELD* definitions can be used, to describe a part of the segment data.

DL/I is a hierarchical database system. This means that the relation between the segments in a database is a *parent to child* (1:n) relationship. The first level segment is named ‘*root*’. For each segment type, you can define a *sequence field*. This specifies in which sequence you will retrieve the data. For dependent segment types (child types), only the data which belongs to one specific parent occurrence is sequenced. The concatenated data of the sequence fields of all parent segments together with the current sequence field value is named ‘*concatenated key*’ (CCK). It describes the current position in the database. Under ADL, one specific sequence field value from the CCK is denoted as *partial concatenated key* (PCK).

Alternate keys are called '*secondary indices*'. A secondary index on a dependent segment type also defines an alternate entry into the database (besides the root). Additionally to the definitions in the physical DBD, you need a *secondary index DBD* for each secondary index.

To reduce data replication, pointers can be defined between two segment types from different DBDs, so-called '*logical child (LC) segments*'. An LC segment contains the concatenated key of the segment, where it is pointing to (the *destination paren*t), and if desired, some more information (*intersection data*). With the help of the LC segments, you can build logical databases, which contain segment types from both connected *physical* databases.

The view from the application to the DBDs is described in a *PSB* (*program specification block*). A PSB is build up by one or more *PCBs* (*program communication block*), each PCB corresponds to one DBD. For each PCB, the *sensitive segments* (*SENSEG*) describe, which segments can be accessed by the application.

The application program communicates with DL/I with a *user PCB*. To access specific data from the database, it can specify a *segment search argument* (*SSA*). The data of the segment is returned in the *I/O area*. A status code, which indicates whether the call was successful, is put into the user PCB.

Example Database

Let's assume you have to build up a database system for a school. The school offers various courses, each course consisting of one or more classes. You want to maintain the courses, the classes and the students taking the classes. On the other hand, there are the instructors teaching the classes. You want to collect information about their salary, skills, and address.

The COURSEDB and INSTDB members on the ADL source library are the DL/I DBD definitions for these databases. The COURSEDB contains the information about the courses, classes, and students; the INSTDB about the instructors, their salary, skill and address. The figure below shows the hierarchical structure of the two databases.

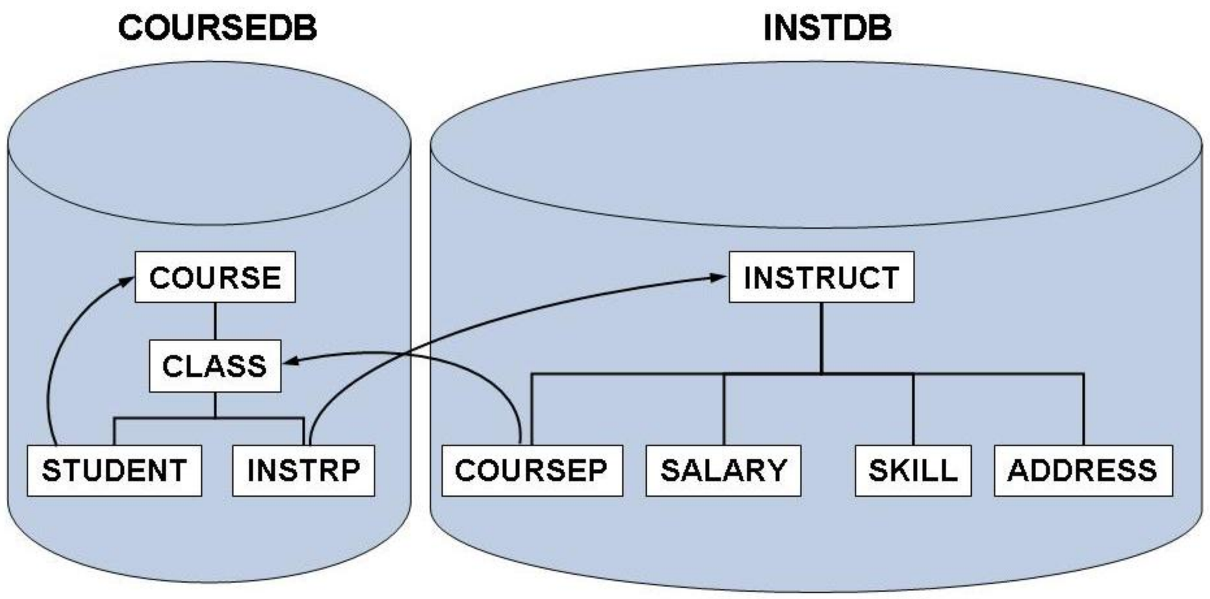


Figure 1: Example Database

The two databases are connected by a logical relationship. In the INSTDB database, there is the segment COURSEP, which points to the class, taught by the instructor. This is indicated by the second parent of the COURSEP segment. Note that in this case, the segment data contains the concatenated key of the CLASS segment. Additionally, it contains a YEAR field as intersection data. In the COURSEDB database, there is the segment INSTRP, which points to the instructor teaching the class. Since both logical child segments contain the same information ('instructor is teaching class'), the corresponding data is kept only in one of them, in the COURSEP segment. The SOURCE keyword at the INSTRP statement indicates that the data is kept at the paired logical child. The COURSEP segment is named the '*real logical child*' (*RLC*), and the INSTRP segment is named the '*virtual logical child*' (*VLC*).

There are two logical databases defined for the logical relationship between the COURSEDB and the INSTDB databases. The COURSEL database starts from the COURSE segment in the COURSEDB database. From here you can access not only the CLASS and STUDENT segments, but also the INSTRUCT segment in the INSTDB database, and all its dependents. The INSTL database allows you to access the COURSE, CLASS and STUDENT segments in the COURSEDB database, when starting from the INSTRUCT segment in the INSTDB database.

In the COURSEDB database, a secondary index is defined. This is indicated by the 'LCHILD POINTER=INDX' statement, followed by an XDFLD statement. It sorts the COURSE segment in the sequence of the student names. This allows you to give a fast answer to questions like 'Which courses is student 'Smith' taking?' The corresponding secondary index DBD is named STUDIDX.

The main PSB describing the two databases is the PSB 'SCHOOL'. The other PSBs on the ADL source library (COURSUNL, INSTUNL, and INSTELO) are required for ADL utilities. Note that the ADL source library also contains the primary index DBD definitions MAINIDX and INSTIDX, which are not used by ADL.

Adabas Terms

With Adabas the data of the same layout is collected in a file, similar to a table of a relational database system. Each file belonging to one database is identified by a *unique file number* (*FNR*), whereas each database is identified by a *unique database Id* (*DBID*). The single piece of data information in a file is named ‘record’, which is identified by a *unique ISN* (*internal sequence number*). A file is build up by one or more fields. Key fields are named ‘*descriptors*’. Multiple fields can be combined into a superdescriptor key field. A field can be defined with the *null-value suppression* (*NU option*), which helps to save data storage. For descriptors, the NU option has the effect that a search with this descriptor will not return a record if the corresponding descriptor value is null.

Series of consecutive fields can be combined in a group. So-called *multiple value fields* (*MU option*) can contain more than one value in a single record. If a descriptor is defined with the MU option, a search will return a record if any of the descriptor values matches the search.

The detailed layout of a file is described in the *file description table* (*FDT*). The FDT of an existing file can be outlined with the Adabas Online Services or the Adabas Manager.

A *DDM* (*Data Definition Module*) is the logical definition of a physical database file referenced by Natural programming objects. For a Natural program, the *user view* describes which fields from a specific file can be accessed.

An application corresponds with Adabas with the *ACB* (*Adabas Control Block*). The *search/value buffer* combination describes the value the application is looking for. The data is returned in the *record buffer*, while the *format buffer* describes which field values should be put into the record buffer. The *response code* in the ACB indicates, whether the call was successful.

DL/I terms versus Adabas terms

The following table gives you a rough correspondence of the DL/I and Adabas terms. Note that this correspondence is on a higher level, for example the DL/I status code and the Adabas response code both return the information how successful the call was, but the detailed codes are by far not the same. For some terms there is no corresponding term in the other database system.

DL/I	Adabas
	Database
Database	File, related files
Segment type	File, part of a file, group
Root segment	
Parent segment	
Child segment	
	Group
Field	Field
Sequence field (root segment)	Descriptor
Sequence field (dependent segment)	
Concatenated key	
Secondary index	Descriptor
Segment occurrence	Record
	ISN
DBD	FDT
PSB	
PCB	(Natural) view
User PCB	Adabas control block
SSA	Search/value buffer
Sensitive field	Format buffer
IO-area	Record buffer
Status code	Response code

How ADL converts the DL/I definitions into Adabas is described in *Adabas File Layout* of the section *Conversion of the Data Structure - General Considerations* of the *ADL Conversion* documentation.

IVP Sample JCL

The ADL source library contains the following sample JCL members for the ADL Installation Verification Package:

IVPCOB	Run a COBOL batch program (ADLXPC1) against ADL.
IVPDBC4C	Unload of the migrated COURSEDB.
IVPDBC4I	Unload of the migrated INSTDB.
IVPDBC6A	Initial load of the course/class data into Adabas.
IVPDBC6B	Initial load of the student data into Adabas.
IVPDBC6C	Initial load of the instructor data into Adabas.
IVPDBC9	Establish the logical relationship.
IVPDPC1	Assemble, link-edit of the DBD and PSB sources.
IVPDPC2	Conversion of the PSBs and logical DBDs.
IVPDPC23	Conversion of the INSTDB and COURSEDB databases.
IVPDPC4A	Assemble, link-edit of the user exit 6 for COURSEDB.
IVPDPC4C	Assemble, link-edit of the user exit 6 for INSTDB.
IVPINVA	Create additional descriptors for the course and class data.
IVPINVB	Create additional descriptors for the student data.
IVPINVC	Create additional descriptors for the instructor data.
IVPZLER	Run a DAZZLER stream (ADLXPD1) against ADL.
IVPZLERT	Run a DAZZLER stream (ADLXPD4) against ADL in shared database mode with the ADL traces facility.

Note: The member IVPINFO contains all abbreviations used in the sample JCL. Before you submit any job, you must replace the abbreviation with the real values, such as ADL.LOAD with the name of the ADL load library. Under z/OS you must additionally edit the member IVPARUN, which contains the ADARUN cards, and adapt it to your requirements.

Conversion of the Example Database

Before you start the conversion, you must consider on which Adabas files the data should be stored. You need three files for the example database. The instructor data (INSTDB) should be stored in one file (FNR=*c*), while the data of the COURSEDB is split up into two files: file *a* for the course and class data, and file *b* for the student data.

The conversion steps are described in more detail in the *ADL Conversion* documentation, sections ADL Conversion Utilities for DBDs and PSBs and ADL Data Conversion Utilities.

Run the sample jobs in the following sequence:

1. IVPDPC1: Assemble and link-edit all DBD and PSB sources. Only the primary index DBD definitions are not assembled, because ADL does not need them.
2. IVPDPC2: Convert the PSBs and the logical DBDs. The DBID and FNR of the ADL directory on which the definition is stored should have already been defined during the ADL Installation.

Note that a logical DBD or a PSB can be converted, even if the corresponding physical DBD is not yet converted.

3. IVPDPC23: Convert the databases INSTDB and COURSEDB. The GENSEG statement for the STUDENT segment is used to store the student data in a different file than the course and class data. This job generates the ADACMP cards for all three files and the macro cards for the Adabas User Exit 6.
4. IVPDPC4A/C: Assemble and link-edit the User Exit 6 cards for COURSEDB and INSTDB respectively, which have been generated in the previous step. Note that there is one User Exit 6 per database, even if the segments are distributed over several files.
5. IVPDBC6A/B/C: Initial load of the Adabas files *a*, *b*, and *c*, respectively. Each job consists of the following steps: Delete the file (if it already exists), compress the data, and load the file. Since we have not unloaded any data before, we now load an empty file. In this case, User Exit 6 is not required. Moreover, we do not need to establish logical relationships in an empty file, which is normally done by running the DAZELORE utility.

   The ADL Conversion utility has generated the following Adabas structures for the example database:

DL/I	Adabas	Description
COURSEDB	**File a**	**Segments COURSE and CLASS of COURSEDB**
address pointers	Z1 - Z8	Pointers to reflect the hierarchy.
address pointers	PC	PCK for the COURSE segment.
COURSE	SA	Course data segment / group.
COURSENO	AA	Course number field (descriptor).
	AB	Filler field for remaining course data.
CLASS	SB	Class data segment / group.
CLASSNO	AC	Class number field.
	AD	Filler field for remaining class data.
COURSEDB	**File b**	**Segment STUDENT of COURSEDB**
address pointers	Z0 - Z8	Pointers to reflect the hierarchy.
address pointers	PC, PB	PCK for the COURSE and CLASS segment.
STUKEY	XA	Secondary index field / descriptor.
STUDENT	SA	Student data segment / group.
SURNAME	AA	Student name field.
	AB	Filler field for remaining student data.
INSTDB	**File c**	**All segments of INSTDB**
address pointers	Z0 - Z8	Pointers to reflect the hierarchy.
address pointers	PA	PCK for the INSTRUCT segment.
INSTRUCT	SA	Instructor data segment / group.

DL/I	Adabas	Description
INSTNAME	AA	Instructor name field (descriptor).
	AB	Filler field for remaining instructor data.
COURSNO	PC	PCK of the COURSE segment as part of INSTRP.
CLASSO	PB	PCK of the CLASS segment as part of INSTRP.
COURSEP	SB	Course pointer intersection data segment / group.
YEAR	AC	Course pointer intersection data field.
SALARY	SC	Salary data segment / group.
DATE	AD	Date field.
AMOUNT	AE	Amount field.
SKILL	SD	Skill data segment / group.
	AF	Filler field for skill data.
ADDRESS	SE	Address data segment / group.
ZIPCODE	AG	Zip code field.
CITY	AH	City field.
STREET	AI	Street field.

After the steps mentioned above have been performed, DL/I applications can run against the example database. But before we continue, we allocated additional descriptors and super-descriptors, which we will need for Natural. Alternatively to the Adabas invert utility, it would also be possible to modify the ADACMP cards generated in step 3, before the initial load.

6. IVPINVA: Create additionally descriptors for file a. We make the field AC (CLASSNO) a descriptor and create a superdescriptor S1 (CCK-CLASS), build up by the fields PC (PCK-COURSENO) and AC (CLASSNO). With the help of this superdescriptor we can easy read the class data in the hierarchical sequence.
7. IVPINVB: Create additionally descriptors for file b. We make the field AA (SURNAME) a descriptor and create a superdescriptor S1 (CCK-CLASS), build up by the fields PC (PCK-COURSE) and PB (CLASSNO). With the help of this superdescriptor we can easy read the student data in the hierarchical sequence.
8. IVPINVC: Create additionally descriptors for file c. For each of the dependent segments SALARY, SKILL and ADDRESS we create a superdescriptor (S2 - S4), which is build up by the field PA (PCK-INSTRUCT) and a part of the segment data. With the help of these superdescriptors we can easy read the segments data in the hierarchical sequence. Additionally we create the super-descriptors S1 (CCK-COURSEP) and S5 (CCK-INSTRP) to reflect the hierarchical view of the COURSEP and the INSTRP segments.

DL/I Applications for the Installation Verification Package

First we want to populate our databases with data. This is performed by the DAZZLER stream ADLXPD1, which inserts courses, classes and students, as well as the related instructors. The DAZZLER program is described in detail in the *ADL Interfaces* documentation, section CALLDLI Test Program - DAZZLER. You can use the sample job IVPZLER to run the DAZZLER.

Perform also the other DAZZLER streams, by modifying the 'CFILE' card in the job IVPZLER.

- Stream ADLXPD2 gives you a summary of all PCBs in the PSB SCHOOL.
- Stream ADLXPD3 reads all students of one specific class.
- Stream ADLXPD4 inserts, modifies and deletes some data. At the end it makes a BACKOUT, which brings the database back into its original status. Use the sample job IVPZLERT for this stream. In this case, ADL is running in mode SDB with `ET=NO`, which enables to use the BACKOUT function. Additionally it starts the ADL trace facility, which is described in the *ADL Interfaces* documentation, section Debugging Aids - ADL Trace Facility. The *Routine Trace* lists all ADL routine names where ADL is running through when it performs the requested function. In the *Database Call Trace*, you can see the DL/I calls as well as the resulting Adabas calls.

The next task is to run some COBOL batch programs against the example database. Compile and link-edit the COBOL programs ADLXPC1/2/3. These COBOL programs use as input streams the members ADLXPI1/2/3, respectively. Use the sample job IVPCOB to submit the programs.

- ADLXPC1 lists the students of one specific course/class.
- ADLXPC2 lists the courses and classes which are visited by one specific student
- ADLXPC3 lists the students which are taught by one specific instructor.

Now we want to run some assembler programs under CICS against the example database. First you must add the PSB SCHOOL to the ADL PSB table DAZPSB, and generate the ADL CICS tables as described in *Generating the Runtime Control Tables* in section *CICS Installation and Operation* of the *ADL Interfaces* documentation.

Assemble and link-edit the assembler programs ADLXPA1 and ADLXPA3. Make an entry in the DFHCSDUP table for each of the programs. If possible, choose as TRANSID the names ADL1 and ADL3, respectively. You may use the member IVPCSD in the ADL source library as input to DF-HCSDUP.

The program ADLXPA1 makes a scheduling call against the PSB SCHOOL and reads some data. The program ADLXPA3 issues a checkpoint, after it has read and replaced some data.

DDMs for the Installation Verification Package

For each segment of the Example Database there is a DDM (Natural data definition module) defined. It is named in the following way:

```
DBDname-segmentname
```

The DDMs contain all the fields described in the DL/I DBD source, the PCK fields for the hierarchical access, and the additionally defined superdescriptors.

For each of these DDMs there is one local data area defined in the SYSADLIV library. The name of the local data area is the same as the corresponding segment name.

The view INSTDB-ALL contains all fields of the INSTDB file c. This includes the ADL internal fields. The corresponding local data area is named INST-ALL.

Before you can run any Natural program for the IVP, you must perform the Natural SYSDDM utility. Re-catalog all views of the IVP with your actual used DBID / FNR combination. Use FNR *a* for the COURSE and CLASS views, FNR *b* for the STUDENT view, and FNR *c* for the others.

Example Database Application

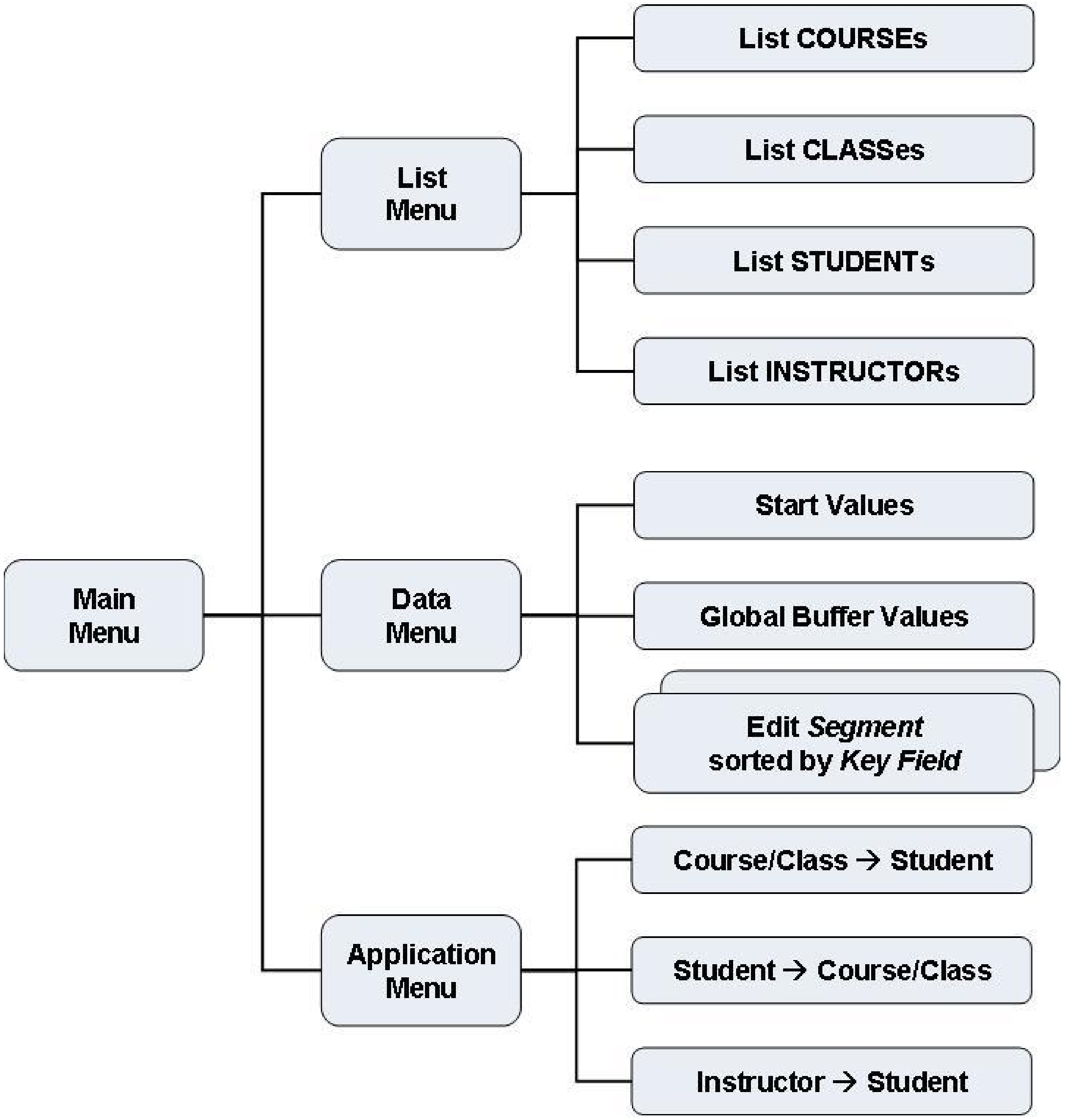
Logon to the Natural library SYSADLIV and catalog all sources by using the Natural CATALL utility.

Except for the MENU program, the members of the Example Database Application are named in the following way:

```
ADBXPntm
```

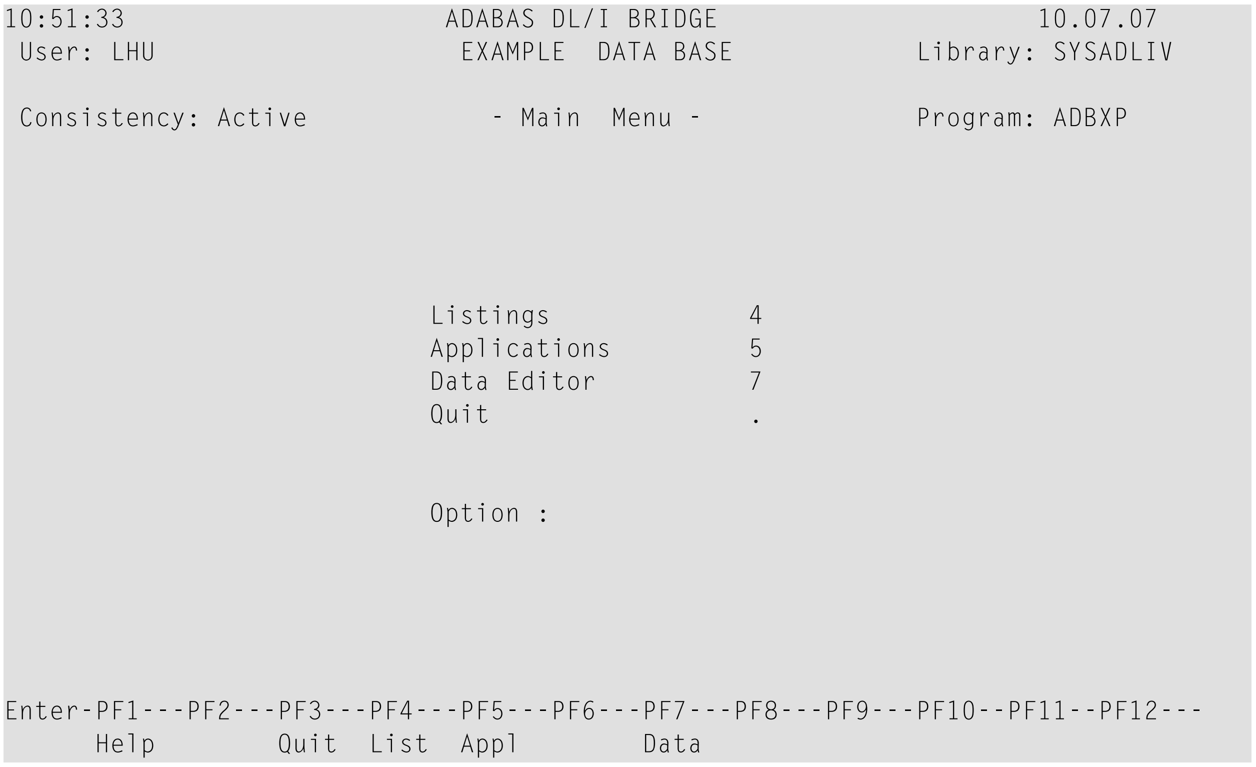
where

n	is the identification of the program (blank, A-N, 1-3),
t	is the type of the object (blank=program, G=global data, L=local data, M=map, S=subroutine), and
m	is the identification of the object.



The Example Database Application SYSADLIV

SYSADLIV Main Menu



```
10:51:33                    ADABAS DL/I BRIDGE                    10.07.07
 User: LHU                   EXAMPLE  DATA BASE           Library: SYSADLIV

 Consistency: Active            - Main  Menu -            Program: ADBXP





                            Listings            4
                            Applications        5
                            Data Editor         7
                            Quit                .


                            Option :






Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Quit  List  Appl        Data
```

The Example Database Application is started with the command 'menu'. The first map displayed is the 'Main Menu'. Here, as in the other screens of the Example Database Application, the user Id, the current Natural library name, and the current active program name are displayed. Additionally it is indicated, whether the ADL Consistency is active or not. For this check, the ADLACTIV subprogram is called, which can also be used by your own applications.

The Example Database Application does not preserve the referential integrity, as described in the *ADL Interfaces* documentation, section Using ADL Files with Natural/Adabas. This enables you to test Consistency error situations when the Consistency is active, as well as to destroy the referential integrity if the Consistency is not active. Note that your own Natural programs should never run against migrated data when the ADL Consistency is inactive.

From the Main Menu you can select three sub-menus: the 'List Menu', the 'Applications Menu' and the 'Data Menu' by choosing the option '4', '5', or '7', respectively, or by pressing the corresponding PF-key. When you choose the option '.' (dot) or press PF3, you will leave the Example Database Application.

SYSADLIV List Menu

```
10:57:03                    ADABAS DL/I BRIDGE                    10.07.07
 User: LHU                   EXAMPLE  DATA BASE            Library: SYSADLIV

                               - List  Menu -              Program: ADBXPA



                           PFK  !  Function
                          ------+-------------------
                           PF3  !  Main Menu
                           PF4  !  List Courses
                           PF5  !  List Classes
                           PF6  !  List Students
                           PF7  !  List Instructors




 Press a PF-KEY!


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Menu  Cour  Clas  Stud  Inst
```

The following functions are available in the ‘List Menu’ by pressing a PF-key:

PF-key	Function
PF3	Redisplay the Main Menu.
PF4	List all courses sorted by the COURSENO field.
PF5	List all classes sorted by the CLASSNO field. This function uses the fact, that the CLASSNO field has been defined as a descriptor (job IVPINVA).
PF6	List all students sorted by the SURNAME field. This function uses the fact, that the SURNAME field has been defined as a descriptor (job IVPINVB).
PF7	List all instructors sorted by the INSTNAME field.

SYSADLIV Application Menu

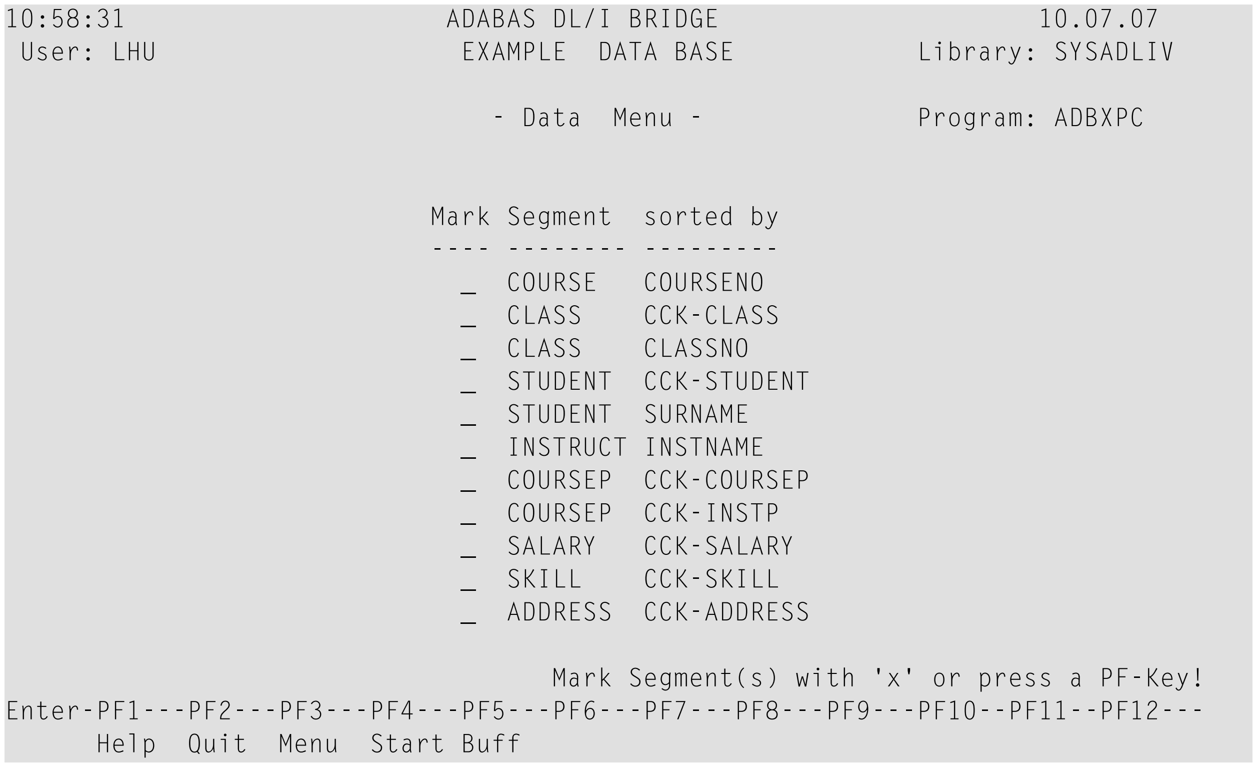
```
10:58:00                     ADABAS DL/I BRIDGE                     10.07.07
 User: LHU                    EXAMPLE  DATA BASE           Library: SYSADLIV

                            - Applications  Menu -         Program: ADBXPB



              PFK  !  Function
           --------+-----------------------------------------------
              PF3  !  Main Menu
              PF4  !  What Students are in a Course / Class?
              PF5  !  What Courses / Classes visits a Student?
              PF6  !  What Students instructs an Instructor?






 Press a PF-KEY!


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Menu  CC>S  S>CC  I>S
```

The Applications Menu provides you with the following functions:

PF-key	Program	Function
PF3		Redisplay the Main Menu.
PF4	ADBXP1	List the students of one specific course/class.
PF5	ADBXP2	List the courses and classes which are taken by one specific student.
PF6	ADLXP3	List the students, which are taught by one specific instructor.

These Natural programs perform exactly the same functions as the COBOL programs ADLXPC1/2/3, described above. Take the time to compare the corresponding sources. The Natural programs are shorter, i.e. faster written, and easier to understand, which means, less bugs and less maintenance. Moreover the Natural programs can run in batch and CICS, while making the COBOL programs able to run online would mean much more programming effort and a more complicated code.

SYSADLIV Data Menu



```
10:58:31                    ADABAS DL/I BRIDGE                    10.07.07
 User: LHU                   EXAMPLE  DATA BASE           Library: SYSADLIV

                               - Data  Menu -             Program: ADBXPC


                            Mark Segment  sorted by
                            ---- -------- ---------
                              _  COURSE   COURSENO
                              _  CLASS    CCK-CLASS
                              _  CLASS    CLASSNO
                              _  STUDENT  CCK-STUDENT
                              _  STUDENT  SURNAME
                              _  INSTRUCT INSTNAME
                              _  COURSEP  CCK-COURSEP
                              _  COURSEP  CCK-INSTP
                              _  SALARY   CCK-SALARY
                              _  SKILL    CCK-SKILL
                              _  ADDRESS  CCK-ADDRESS

                                    Mark Segment(s) with 'x' or press a PF-Key!
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Quit  Menu  Start Buff
```

From the Data Menu, you can edit the data of all segments of the Example Database. Mark a line with ‘x’ to edit the corresponding data. The ‘sorted by’ field indicates which descriptor is used to sort the data for the editor. For some segments, more than one sequence is possible, for example the CLASS segment can be edited in the hierarchical sequence with the CCK-CLASS key, or directly in the sequence of the CLASSNO field. The logical child segment COURSEP can be viewed like the COURSEP segment by the CCK-COURSEP key (sequence: INSTNAME), or like the INSTRP segment by the CCK-INSTP key (sequence: COURSENO/CLASSNO).

When you edit the data of a segment, the list begins at the start-value of the sort-field. When you press PF4 in the Data Menu, the start-values for the key fields are displayed, and can be modified.

When you press PF5 in the Data Menu, the ‘Global Buffer Values’ are displayed and can be modified. There is one global buffer value for each descriptor. The global buffer values are used at the ‘yank’ and ‘put’ commands in the editor, as described later.

When you press the PF3 key in the Data Menu, the Main Menu is redisplayed.

SYSADLIV Example Database Editor

```
Segment:   COURSE            ADABAS DL/I BRIDGE            User:    LHU
 sorted by: COURSENO          EXAMPLE  DATA BASE            Library: SYSADLIV
 Start:     BIOLOGY300              EDITOR                  Program: ADBXPD
 -------------------------------- DATA  AREA ----------------------------------
 S M COURSENO   COURSENAME
   _ BIOLOGY300 BIOLOGY_____________
   _ EDV    800 EDP_________________
   _ ENGLISH620 ENGLISH_____________
   _ GERMAN 610 GERMAN______________
   _ GREEK  400 GREEK_______________
   _ HISTORY500 HISTORY_____________
   _ MATHEMA200 MATHEMATICS_________
   _ PHILOSO100 PHILOSOPHY__________
   _ RINGKNO700 KNOWLEDGE OF RING___
   _ __________ ____________________
 -------------------------------- INPUT AREA ----------------------------------
   _ __________ ____________________
   _ __________ ____________________
   _ __________ ____________________
   _ __________ ____________________
   _ __________ ____________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  STOP  Menu  Segm  Start Save  Top   Next  Input       Let   Undo
```

When you have marked any segment in the Data Menu, you come into the Example Database Editor. At the top of the screen, the current segment name, the name of the sort key, and the start-value are displayed. The main part of the screen is spitted into two areas: the 'Data Area' and the 'Input Area'. The Data Area displays the data of the segment. You can modify the data by overtyping it. Modifications in the editor data does not automatically result in modifications of the database data, unless you have saved it. In the Input Area you can specify the data, which should be inserted into the database.

There are two special columns in front of the data, the status column (‘S’) and the line-command column (‘M’). The status column ‘S’ indicates the actual status of the data in the line:

Value	Area	Description
D	Data	Data is marked for deletion.
M	Data	Data is marked for modification.
I	Input	Data is marked for insertion.

The 'M' column can be marked with the following line commands:

Value	Area	Description
D	D+I	Delete the line.
C	D+I	Copy the line to the Input Area.
L	D+I	Undo the changes in the line since last Enter or PF-key pressing.
U	Data	Undo the changes in the line since last Save request.
Y	Data	‘Yank’, i.e. copy the key values from the current line into the global buffer.
P	Input	Put the key values from the global buffer into the current line.

The PF-keys provide the following functions:

PF-key	Name	Function
Enter		All changes are performed on the screen, but there is no access to the database. The line commands are executed and the status column is set.
PF1	Help	Display the help text.
PF2	STOP	Redisplay the Data Menu. All modifications since the last Save / Input are lost.
PF3	Menu	Redisplay the Data Menu. The modifications of the data area are saved and the data from the Input area is inserted into the database, i.e. it includes the 'Save' and the ‘Input’ functions.
PF4	Segm	Change the segment and the sort-key names. If you specify an incorrect segment / sort-key combination, the Data Menu is displayed. This function includes the 'Save' and the ‘Input’ function.
PF5	Start	Modify the start field value. The list will start at the value greater than or equal to the specified one. This function includes the 'Save' function.
PF6	Save	The modifications of the data area are saved.
PF7	Top	The list is started from the top. This function includes the ‘Save’ function.
PF8	Next	The next page is listed. This function includes the ‘Save’ function.
PF9	Input	Insert the data from the Input Area into the database. This function includes the ‘Save’ function.
PF11	Let	Undo all changes since the last Enter or PF-key pressing.
PF12	Undo	Undo all changes in the Data Area since the last Save request.

With the Example Database Editor you can easily test the ADL Consistency rules. What happens if you delete a COURSE, which has dependents? Can you insert a STUDENT with a not-existing CLASSNO? Which fields can be modified? By 'playing' through questions like this, you will get a better feeling for the Consistency rules. Note that it is recommended that your own applications never violate the Consistency rules, i.e. they should never receive an error message from the ADL Consistency. The section Using ADL Files with Natural/Adabas in the *ADL Interfaces* documentation describes how to archive this.

Other Natural Objects of the Installation Verification Package

On the SYSADLIV library there are some programs which show how to respect the referential integrity.

- The program DEL-COUR deletes a COURSE segment occurrence. Before it performs the deletion, it checks, whether the COURSE has dependent segments.
- The program INS-STUD inserts a new student record. Before the insert, it verifies whether the chosen COURSE/CLASS path exists.
- The programs UPD-COUR and UPD-STUD update a COURSE and a STUDENT record. These programs modify neither the sequence field nor the PCK fields. The UPD-STUD program updates the secondary index source fields, while the secondary index descriptor field (XA) is handled by the Consistency.

When you delete a parent segment type under DL/I, all dependent segment occurrences are deleted automatically. This is named 'hierarchical cascaded deletion'. The Consistency does not perform a cascaded deletion. It deletes only the current record, or if this record has dependents, it returns a response code. Thus you must code your own hierarchical cascaded deletion if you want to perform such a task. In the SYSADLIV library there are some examples for a hierarchical cascaded delete.

- The subprograms CASDELST and CASDELIP delete all students and instructor pointers, belonging to a given PCK-COURSE/PCK-CLASS combination. Since these segments do not have dependent segments, the deletion can be performed without any further validation.
- The subprogram CASDELCL deletes all classes belonging to a given PCK-COURSE. Before it deletes a class, it deletes all dependent STUDENT and INSTRP occurrences by calling the subprograms CASDELST and CASDELIP.
- The programs CDELCOUR and CDELCLAS ask for one course or class number, for which it will perform a hierarchical cascaded deletion. They use the subprograms mentioned above to delete the dependent segment occurrences, before they delete the COURSE or CLASS record itself.

The program READ-Z uses the view INSTDB-ALL to read the data of the INSTDB database in the sequence of the ADL internal pointer field Z1. Additionally it selects some specific data. You can use the ADL internal fields for specific purposes, like validation of the data, but you should keep in mind, that these fields will no longer be supplied if the ADL Consistency has become obsolete.

Finally there are some programs and subprograms in the SYSADLIV library, which can be used by your applications. You can copy the source programs into your application library. Use them as delivered, or adapt them to your requirements.

- The subprogram ADLACTIV verifies whether the ADL Consistency is active. It returns a 2-bytes integer response code. For a more detailed description, see *Availability of the Consistency Interface* in the section *Using ADL Files with Natural/Adabas* of the *ADL Interfaces* documentation.
- The subprogram ADLACTIM verifies whether the ADL Consistency is active. It returns the same 2-bytes integer response code as the subprogram ADLACTIV. Additionally it returns an 80-bytes character message telling the status of the ADL Consistency.
- The program ADLCONSI shows how to use the ADLACTIV subprogram.
- The subprogram ADLERROR returns the last Consistency error message in an 80-bytes character field. For a more detailed description see *Error Situations and Consistency Response Codes* in the section *Using ADL Files with Adabas* of the *ADL Interfaces* documentation.
- The subprogram ADLFNR returns the DBID and FNR of the ADL directory as defined with the Natural `LFILE` parameter. Both values are numeric fields of length 5. Additionally it returns a 2-bytes integer response code. If an LFILE setting for the ADL directory file is defined, the response code is zero.
- The program LFILE sets the Natural `LFILE` parameter for the ADL directory. It reads the new DBID and FNR from the input.
- The subprogram SETLFILE sets the Natural `LFILE` parameter for the ADL directory. Use function=3 and specify the DBID and FNR parameters (each 5 byte numeric) as required. The subprogram returns a 2 byte integer response.

Tuning the ADL Installation Verification Package

The tuning of applications which run against the ADL is described in general in the section Performance Considerations in the *ADL Interfaces* documentation. Here we take a closer look to three possibilities, which can increase the performance:

Hierarchical Sequence

Originally the data is loaded randomly, i.e. in the sequence how the inserts have been issued. The hierarchical sequence can be established by performing a logical unload with DAZUNDLI, a reload with the Adabas utilities ADACMP and ADALOD, and an establishing of the logical relationships with DAZELORE, as described in the section Managing ADL Files in the *ADL Interfaces* documentation.

You can use the sample jobs IVPDBC4C and IVPDBC4I to unload the COURSEDB and INSTDB databases, respectively. These jobs use the unload PSBs COURSUNL and INSTUNL. The data can be reloaded into Adabas with the sample jobs IVPDBC6A, IVPDBC6B and IVPDBC6C. If you reload the data in this way, you must re-create the additionally descriptors with the jobs IVPINVA, IVPINVB and IVPINVC. Alternatively you can reload the data by refreshing the three files and loading the data. In this case, you must modify the IVPDBC6x jobs, so that they use the ADALOD UPDATE function. Finally the logical relationship must be re-established by running the DAZELORE utility. This is handled by the sample job IVPDBC9, which uses the PSB INSTELO.

Last Call Savearea (LCS)

The LCS is switched on by specifying the LCS parameter for DAZIFP. We specify the value 'LCS=7'. Note that the LCS does not save data of logical child segment types (here INSTRP and COURSEP).

ADARUN Multifetch Feature

ADARUN Multifetch is activated by specifying the PREFETCH parameter for ADARUN. We specify the value

```
PREFETCH=YES,PREFSBL=32767,PREFTBL=294903
```

This means that the highest single buffer length (PREFSBL=32767) is used and one single buffer for each of the 9 segments (PREFTBL=9*32767=294903).

Performance Test Streams

We use two different test streams. The streams are named 'ADLXPD5' and 'ADLXPD6' and reside on the ADL source library. In the first stream we read all segments of the DBD COURSEDB and all segments of the DBD INSTDB sequentially, i.e. we make unqualified GN calls, until the end of the database is reached. In the second stream we read the student data together with their COURSE and CLASS information, i.e. we make GN path calls to the STUDENT segment.

At the beginning of every stream there are seven L3-calls against the ADL directory. These seven calls use all the value-start option (OP2=V). In the following we count only the number of L3-calls against the data files.

Test 1) Read all Segments of COURSEDB Sequentially (Stream ADLXPD5)

Number of DL/I calls: 426

JobNo.	1	2	3	4	5
hierarchical sequence	N	N	Y	Y	Y
LCS	N	Y	N	Y	Y
ADARUN Multifetch	N	N	N	N	Y
Number of L3's with value start	99	76	99	50	50
Number of L3, no value start	424	424	424	424	74
Total number of L3 calls	523	500	523	474	124

In the first job, we read the data in the original sequence, without using any additional feature. As you can see, the number of L3-calls is much higher than the number of DL/I calls. This is because ADL makes one unsuccessful L3-call each time that the end of a twin chain is reached.

In the second job the number of value-starts is reduced by using the last-call-savearea. In this run, the LCS can only help randomly, because the data is not in the hierarchical sequence.

In the third job, we have sorted the data in the hierarchical sequence. The number of L3-calls is still the same as in the first run. This is because we must satisfy the same DL/I calls, and we make the same unsuccessful L3 calls at the end of a twin chain as in the first job. Nevertheless, re-establishing the hierarchical sequence can decrease the number of I/Os because successive records can be found on one block. In our example it makes no sense to look to at the number of I/Os because the amount of data is too small.

In the fourth job the LCS can work optimally, because the data is found in the hierarchical sequence. Nevertheless there are relatively many L3-calls with value-start option, because the LCS doesn't work on logical child segment types. In this case there are 42 value-starts against the INSTRP and COURSEP segments, and only 8 value-start calls against all the other segments.

Now we can use with the fifth job the ADARUN Multifetch feature, since the number of value-start calls is minimized. This reduces the number of Adabas L3 calls, which do not use the 'V' option, considerably.

Test 2) Read all Students in COURSEDB Sequentially (Stream ADLXPD6)

Number of DL/I calls: 311

JobNo.	1	2	3	4	5
hierarchical sequence	N	N	Y	Y	Y
LCS	N	Y	N	Y	Y
ADARUN Multifetch	N	N	N	N	Y
Number of L3’s with value start	30	17	30	3	3
Number of L3, no value start	338	338	338	338	7
Total number of L3 calls	368	355	368	341	10

In this stream we do not access any logical child segment. Therefore the LCS can work optimal. In case the data is in the hierarchical sequence (job 4 and 5) we need exactly one value-start call for each of the three segments. Together with the ADARUN Multifetch feature, ADL can satisfy the 311 DL/I calls, by issuing 10 Adabas calls.

8 Migration to ADL 2.3 and Backward Migration

This section covers the following topics:

Migrating to ADL 2.3

The Adabas Bridge for DL/I uses several internal fields to reflect the DL/I hierarchy. These fields are the Z-field (`Z0, Z1, Z2` etc.) and the secondary index fields (their Adabas names start usually with `'X'`). The Z-fields are stored with each dependent record, the secondary index fields with each secondary index source record.

Some of the internal fields contain Adabas ISNs; with ADL 2.2 the `Z0` and `Z1` fields additionally contained a DBID and file number. The layout of ADL internal fields has been changed with ADL 2.3 to be able to handle 4-byte ISNs and 2-byte DBIDs and file numbers. The new layout (see the section *Performance Considerations* in the *ADL Interfaces* documentation for more information) is not compatible with the ADL 2.2 layout, i.e. ADL 2.3 does not support the ADL 2.2 layout and vice versa. Therefore all data which have been converted with ADL 2.2 or below must be migrated to the new ADL 2.3 layout before you can work with ADL 2.3.

Although ADL 2.3 is able to run with ADL 2.2 directory entries, physical DBD definitions must be reconverted to generate the new ADACMP and DAZUEX06 cards. It is not required to re-convert PSB definitions or logical DBD definitions.

For the migration, all ADL files (i.e. files converted from DL/I) must be unloaded and reloaded with Adabas utilities. At the reload the Adabas User Exit 6 DAZUEXMI maps the old layout to the new layout.

The following steps must be performed for the migration of the data to the new layout:

- Step 1: Preparation
- Step 2: Install ADL 2.3
- Step 3: Run the DBD definition through the ADL CBC utility
- Step 4: Create the Adabas User Exit 6 for migration
- Step 5: Modify the ADACMP cards
- Step 6: Unload and reload the data with Adabas utilities

After these steps have been performed for all ADL files, you can run your applications against ADL 2.3.

For emergency, ADL offers a backward migration from ADL 2.3 to ADL 2.2. See section ***Backward Migration*** for details.

The following sections describe the migration steps in more details.

Step 1: Preparation

Copy the ADL directory file with standard Adabas utilities. The new file will be used as ADL 2.3 directory.

Note: Although this step is not required, it allows to convert the DBD definitions to the new directory (step 3) while ADL 2.2 is still running.

Save the ADL 2.2 User Exit 6 extension (if available) and the old ADACMP cards. The old User Exit 6 extension simplifies the data migration (step 6) because you do not need to specify the SEQ parameter. The old ADACMP cards will be checked later against the new ADACMP card to identify manual changes (step 5)

Step 2: Install ADL 2.3

The installation of ADL 2.3 is described in ***Upgrading to New ADL Releases*** in the section *Miscellaneous*.

Load the new error messages into the new ADL 2.3 directory file.

If you have used logical file numbers with ADL 2.2, you can delete the DAZLDT entries from the ADL 2.3 parameter module.

Step 3: Run the DBD definition through the ADL CBC utility

The CBC utility is described in details in the *ADL Conversion* documentation, section *ADL Conversion Utilities for DBDs and PSBs*. Consider the following:

- There is no need to re-assemble the DBD definition (step 1 of the CBC utility).
- The `DBID` and `FNR` parameters of the GENDBD/GENSEG functions reflect the physical file, i.e. the Adabas file where the data is stored (with ADL 2.2 it reflected the logical file).
- If the DBID of the data file is the same as the DBID of the ADL directory, omit the `DBID` parameter for the GENDBD function. This eases the creation of mirror databases.
- Specify as `LOGID` the previous value of `FNR` for every GENDBD/GENSEG function where the `FNR` parameter was specified.
- Specify all other parameters as with ADL 2.2. Especially if you have used the `BACKW` parameter, it must use the same value as before.
- The member with the ADACMP cards is named W*`fffff`*, where *`fffff`* is the file number. Use several libraries to save ADACMP cards of several DBIDs.
- The member with the User Exit 6 extension is named I*`fffff`*, where *`fffff`* is the file number.

Step 4: Create the Adabas User Exit 6 for migration

Assemble the Adabas User Exit 6 extension and link-edit this with `DAZUEXMI`. The Adabas User Exit 6 extension is the output of step 3 of the control block conversion.

If available, use the extension generated by ADL 2.2, otherwise the extension generated by ADL 2.3.

Step 5: Modify the ADACMP cards

If you had adjusted the Adabas FDT (additional user fields, super descriptors, etc.) with previous ADL versions, you must apply the changes to the ADACMP cards accordingly.

Step 6: Unload and reload the data with Adabas utilities

Unload each Adabas file used to store the converted data using the standard Adabas utility ADACMP DECOMPRESS. Specify the option ISN.

The data of each file is loaded individually using the standard Adabas utilities ADACMP COMPRESS and ADALOD.

The sequential file produced by the ADACMP DECOMPRESS is taken as input for the ADACMP COMPRESS step, as are the ADACMP statements generated by the CBC utility.

Each Adabas file used must be loaded with the option USERISN. This applies to both the ADACMP and the ADALOD steps (for ADACMP it is already generated by the CBC utility).

Adabas User Exit 6

The ADACMP COMPRESS step uses Adabas User Exit 6. This exit consists of two parts which were linked together in step 4:

1. **Fixed Part**
   The fixed part consists of the `DAZUEXMI` module, which maps the ADL 2.2 layout to the ADL 2.3 layout.
2. **User Exit 6 Extension**
   The User Exit 6 extension is generated by the `CBC` utility and contains information on the structure of the DBD being migrated, and the default record layouts of the Adabas file(s) used to store the data. You can use the User Exit 6 Extension generated by ADL 2.2 as well as the extension generated by ADL 2.3.

Adabas User Exit 6 needs a control statement to indicate which Adabas file should be migrated. The syntax of this control statement is as follows:

```
FNR=nnnnn,MODE=MIGR,SEQ=sss,DBID=nnn
```

Parameter	Description	Possible values	Default
FNR	Specifies the file number to be processed. The value must be the same as the `FNR` specified at the CBC utility run, i.e. the logical file number if the extension was generated with ADL 2.2 or the physical file number if it was generated with ADL 2.3.	1 - 65534	
MODE	Specifies whether the data should be migrated from ADL 2.2 to ADL 2.3 or vice versa.	MIGR -The data is migrated from ADL 2.2 to ADL 2.3 BACK - The data is migrated back from ADL 2.3 to ADL 2.2	MIGR
SEQ	Specifies the ADL 2.2 processing sequence of the file. This parameter must only be specified if the extension was generated with ADL 2.3.	SEG - Sequence is “segment”. This was the default for the ADL 2.2 conversion. PAR - Sequence is “parent”. This is the setting for all DBDs converted with ADL 2.1 or before.	SEG
DBID	Specifies the logical DBID to be inserted to the ADL internal field. The value must be the same as the DBID specified at the ADL 2.2 `CBC` utility run. This parameter is mandatory for the backward migration (`MODE=BACK`), otherwise it is not required.	1 - 255	

Backward Migration

For emergency, ADL offers a backward migration from ADL 2.3 to ADL 2.2. The following steps must be performed for a backward migration:

- Install ADL 2.2
- Restore the ADL directory

- Unload and reload the data with Adabas utilities

Install ADL 2.2

The ADL 2.2 installation is described in the ADL 2.2 documentation. If the old ADL load library is still available, you can omit this step.

Restore the ADL directory

Use the original ADL 2.2 directory (see step 1 of the migration).

Alternatively you can convert the DBD definition with ADL 2.2. The `DBID` and `FNR` parameters of the ADL 2.2 GENDBD/GENSEG functions reflect the logical file which is described in details in the ADL 2.2 documentation.

If the logical file numbers differ from the physical file numbers, DAZLDT entries must be added to the ADL parameter module.

Unload and reload the data with Adabas utilities

Unload each Adabas file used to store the converted data using the standard Adabas utility `ADACMP DECOMPRESS`. Specify the option `ISN`.

The data of each file is loaded individually using the standard Adabas utilities `ADACMP COMPRESS` and `ADALOD`. The sequential file produced by the `ADACMP DECOMPRESS` is taken as input for the `ADACMP COMPRESS` step. Use the `ADACMP` statements generated by the ADL 2.2 `CBC` utility run. Each Adabas file used must be loaded with the option USERISN. This applies to both the `ADACMP` and the `ADALOD` steps.

The `ADACMP COMPRESS` step uses Adabas User Exit 6. For the backward migration use the same Adabas User Exit 6 which was used for the (forward-) migration of the file to ADL 2.3.

The parameters of the Adabas User Exit 6 are described in details in the section above. For the backward migration you must specify `MODE=BACK`. Additionally to the parameters used for the migration, you must specify the `DBID` parameter, reflecting the logical DBID used at the ADL 2.2 CBC utility run.

JCL Requirements

The JCL requirements for the migration and backward migration are the same as for the initial load. Refer to *z/OS JCL Requirements* or *z/VSE JCS Requirements* in the section *ADL Data Conversion Utilities* of the *ADL Conversion* documentation.

Other Changes

z/OS CICS application programs which have been linked with the ADL language interface DAZ-LICI2, must be re-linked with the new language interface DAZLICI3. This is especially true for all precompiled EXEC DLI programs. See the *ADL Interfaces documentation*, section *CICS Installation and Operation* for further information.

9 Miscellaneous

This chapter covers the following topics:

User Exit DAZUEX01

Before ADL issues an Adabas call, it passes control to a user-written user exit. The purpose of this user exit is to allow any modifications to Adabas parameters, for example, assignment of Adabas command IDs, to conform to user site standards. Also, this user exit might be useful for monitoring programs as well as access control programs.

Note, however, that the user is responsible for the integrity of the Adabas call parameters. This is in particular true for the Adabas command ID. ADL uses specific Adabas command IDs which are determined according to an internal algorithm. Should the user intend to modify these command IDs, care should be taken that identical Adabas command IDs used by ADL are reflected by identical command IDs in the user exit.

The entry point name of the user exit must be `DAZUEX01`. On entry to the user exit the following register conventions apply in batch and online:

R1	points to the Adabas parameter list
R13	points to a 18-fullword save area
R14	return address
R15	entry point address of DAZUEX01

In addition, under CICS, R8 points to the `CSA` and `R12` points to the user `TCA`.

The contents of those registers not mentioned above are undefined. On return, all registers must be restored to their original values with the exception of `R15`. The save area pointed to by `R13` may be used to save the registers of ADL.

The user exit must be re-enterable. When running under z/OS/XA, the user exit must eventually be able to run in `AMODE 31`, depending on how your environment has been configured.

In order to become active, the user exit must be link-edited with the ADL batch region controller, `DAZIFP`, with the executable ADL Consistency Interface module for batch, `DAZNUCA`, and with the executable ADL Interfaces module for CICS, `DAZNUCC`. See the section ***z/OS Installation*** or ***z/VSE Installation*** for more details on how to link-edit these modules.

User-supplied Index Maintenance Exit Routines

With the `EXTRTN` keyword of the `XDFLD` statement in the `DBD` definition, the name of a user-supplied index maintenance exit routine can be specified. This exit routine allows the suppression of indexing for certain data base records.

On entry to the exit routine, the following register conventions apply:

Register	Usage
R2	points to the proposed index pointer segment
R3	points to the exit routine table entry described below
R4	points to the index source segment
R13	points to an 18-fullword save area
R14	return address
R15	entry point address of the exit routine

The contents of registers not mentioned above are undefined. On return, all registers must be restored to their original values with the exception of `R15`, which must contain a return code of either 0 or 4. A return code of 0 means that the index is to be allocated, whereas a return code of 4 means that the indexing of the data base record is suppressed.

Control is passed to the exit routine for insert and replace calls which affect the corresponding secondary index. Unlike DL/I, the exit routine is not called by ADL for delete requests, since the descriptor value corresponding to this index is deleted by Adabas anyway. Therefore, logical error situations cannot occur.

An exit routine table entry has the following layout (see the source library member `MEXRENDS`):

```
EXRENDS  DSECT
EXRENTGN DS    CL8                 NAME OF TARGET SEGMENT
EXRENXDN DS    CL8                 NAME OF XDFLD
EXRENNAM DS    CL8                 NAME OF USER EXIT ROUTINE
EXRENADR DS    A                   ENTRY POINT OF USER EXIT ROUTINE
EXRENDBD DS    A                   ADDRESS OF DBD ICB
EXRENSEX DS    H                   O.T. SEC. INDEX ICB (WRT. DBDIC)
EXREN$FL DS    X                   FLAG:
EXREN#AD EQU   B'00000001'           'USER EXIT ALREADY DEFINED'
EXREN#NF EQU   B'00000010'           'USER EXIT NOT FOUND'
         DS    0F                  ALIGN
EXRENLLL EQU   *-EXRENDS
```

In order to become active, the exit routine must be an executable module. The module name must be the same as specified with the `EXTRTN` keyword. It must be present on the corresponding load library. Under CICS, an entry in the DFHCSDUP table must be defined for each exit routine.

A sample exit routine is supplied in the source library member `ADLIMEX`.

Application/Integration of Software Corrections

Unless otherwise stated, all corrections distributed for ADL will be valid for both z/OS and z/VSE operating systems.

Source changes are to be applied directly to the affected members in the ADL source library. Under z/VSE, source changes to macros must be followed by an `EDECK` assembly of the affected macros.

Fixes to ADL load modules (`ZAP`s) are distributed in the `IMASPZAP` format for the z/OS operating system, and in `MSHP` format for the z/VSE operating system.

When applying corrections to load modules, it is recommended to follow the guidelines given in the *Adabas Installation* documentation.

It is strongly recommended to apply all `ZAP`s to the affected object modules directly and to repeat the link-edit of the affected ADL executable load modules afterwards. This is to ensure that the corrections remain active, even if an executable module should be re-linked.

Upgrading to New ADL Releases

In general, an upgrade to a new ADL release will require the following steps to be performed:

1. Load the ADL libraries to disk. (Step 1 of the original installation procedure.)
2. Update the error messages on the ADL directory file. (Step 2 of the original installation procedure.)
3. Reload the ADL Online Services. (Step 3 of the original installation procedure.)
4. Regenerate the ADL parameter module. (Step 4 of the original installation procedure.)
5. Create the new ADL executable load modules. (Steps 5 to 9 of the original installation procedure.)
6. Create the new Consistency front-ends. (Steps 10 to 11 of the original installation procedure.)
7. Regenerate the CICS Runtime Control Tables.

Steps 1 through 6 are described in detail in the sections ***z/OS Installation*** and ***z/VSE Installation***. Step 7 is described in the *ADL Interfaces* documentation.

Steps 1 through 5 can be performed with Software AG's System Maintenance Aid (SMA).

When upgrading to a new ADL release, you will usually not have to reconvert your `DBD` definition or to repeat the database conversion procedure. The upgrade to ADL 2.3 requires a data migration as described in the section ***Migration to ADL 2.3 and Backward Migration***.

10 Appendix A - z/OS Dataset Usage

The following table provides a complete list of all the z/OS input and output data sets used by the ADL batch monitor, `DAZIFP` and the ADL utilities.

DDname	BLKSIZE	LRECL	RECFM	DSORG	Storage Medium
DAZIN1		80	F	PS	READER
DAZIN2		80	F	PS	READER
DAZOUT1		132	F	PS	PRINTER
DAZOUT2		132	F	PS	PRINTER
DAZIN3	user-defined	user-defined	VB	PS	TAPE/DISK*
DAZOUT3	user-defined	user-defined	VB	PS	TAPE/DISK*
DAZIN4	8196	8192	VB	PS	TAPE/DISK
DAZIN5	user-defined	132	FB	PS	TAPE/DISK*
DAZOUT5	8196	8192	VB	PS	TAPE/DISK
DAZOUT4		80	F	PS	TAPE/DISK

* The block size and/or record format may be changed by specifying a `DCB` for the data set in the `DD` statement.

11 Appendix B - z/VSE Dataset Usage

The following table provides a complete list of all the z/VSE input and output files used by the ADL batch monitor, `DAZIFP` and the ADL utilities.

The following table provides a complete list of all the z/VSE input and output files used by the ADL Interfaces.

DTF	Logical Unit	BLKSIZE	LRECL	RECFM	DSORG	Storage Medium
DAZIN1	SYSIPT		80	F	PS	READER
DAZIN2	SYSIPT		80	F	PS	READER
DAZOUT1	SYSLST		132	F	PS	PRINTER
DAZOUT2	SYS011		132	F	PS	PRINTER *
DAZIN3D	SYS013	8196	8192	VB	PS	DISK **
DAZIN3T	SYS013	8196	8192	VB	PS	TAPE **
DAZOT3D	SYS013	8196	8192	VB	PS	DISK **
DAZOT3T	SYS013	8196	8192	VB	PS	TAPE **
DAZIN4D	SYS012	8196	8192	VB	PS	DISK ***
DAZIN4T	SYS012	8196	8192	VB	PS	TAPE ***
DAZIN5D	SYS014	1320	132	FB	PS	DISK ****
DAZIN5T	SYS014	1320	132	FB	PS	TAPE ****
DAZOT5D	SYS012	8196	8192	VB	PS	DISK ***
DAZOT5T	SYS012	8196	8192	VB	PS	TAPE ***
DAZOUT4	SYSxxx		80	F	PS	DISK *****

* This printer output may be routed to a second logical printer, if one is available, by specifying `PR=2` in the ADL parameter module or as a dynamic parameter. If no second logical printer is available, the printer output is written to a temporary file using `DAZOT3D` as output and `DAZIN3D` as input, and printed on `SYSLST` at the end of the job.

** The logical unit and block size may be modified by specifying `SQ=(logical unit,block size)` in the ADL parameter module or as a dynamic parameter. The values given in the table are the default values.

*** The logical unit and block size may be modified by specifying `TRACE=(,,,,,,logical unit,block size)` in the ADL parameter module or as a dynamic parameter. The values given in the table are the default values.

**** The logical unit and block size may be modified by specifying `FX=(logical unit, blocksize)` in the ADL parameter module or as a dynamic parameter. The values given in the table are the default values.

***** The logical unit must be assigned using an `EXTENT` statement.