

Adabas Bridge for DL/1

Interfaces

Version 2.3.2

November 2016

This document applies to Adabas Bridge for DL/1 Version 2.3.2.

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

Copyright © 2016 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://softwareag.com/licenses>.

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://softwareag.com/licenses/> 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, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

Document ID: ADL-INTERFACES-232-20161117

Table of Contents

Interfaces	vii
1 Introduction	1
Operational Environments	5
Other Documentation You May Need	5
Documentation Related to non-SAG Products	6
2 Batch Installation and Operation	7
CALLDLI Interface	8
Consistency Interface	12
User Exit	16
z/OS JCL Requirements	17
z/VSE JCS Requirements	18
3 CICS Installation and Operation	21
Overview CICS Installation	22
Prerequisites for z/OS CICS Installation	22
Prerequisites for z/VSE CICS Installation	24
Important Note for CICS Users	27
Generating the Runtime Control Tables	27
Tuning and Maintaining the Runtime Control Tables	37
z/OS Requirements	38
z/VSE Requirements	39
Activating and Controlling the ADL Interfaces	40
CALLDLI Interface	42
Consistency Interface	44
User Exit	46
4 IMS/TP Installation and Operation	47
Overview	48
Generating the Runtime Control Tables	48
CALLDLI Interface	49
ADL Pre-load Program	51
User Exit	51
JCL Requirements	51
5 ADL Online Services	53
Introduction	54
Main Menu	56
Maintaining the ADL Interfaces under CICS	58
ADL Directory Management Facility	63
Consistency DBD Maintenance	74
Maintenance of the Rolled-out PSBs	75
Maintenance of Checkpoints	80
Messages and Codes Retrieval	84
6 Precompiler for EXEC DLI Commands	87
Introduction	88
ADL Precompiler Input	91

ADL Precompiler Output	93
COBOL Generated Code	94
PL/I Generated Code	94
CICS Command Language Translator	94
Linkage-Editor Requirements for Application Programs	94
z/OS JCL Requirements	95
z/VSE JCS Requirements	96
7 Using ADL Files with Natural/Adabas	99
Introduction	100
Consistency Interface	100
Restrictions when using Natural/Adabas	101
Improve the Natural Access to Migrated Files	103
Error Situations and Consistency Response Codes	104
Availability of the Consistency Interface	105
Example Programs	105
8 Converting Natural for DL/I Programs	109
Introduction	110
Conversion of the Data Definitions and of the Data	110
Modification of the Application	111
9 SQL Access to the Migrated Files	115
Introduction	116
How to Access the Migrated Data	116
Improving the Access to Migrated Data	117
Restrictions for SQL Applications	117
10 Debugging Aids - ADL Trace Facility	119
Data Base Call Trace	120
Internal Routine Trace	124
z/OS JCL Requirements	125
z/VSE JCS Requirements	126
11 CALLDLI Test Program - DAZZLER	129
Introduction	130
DL/I Statements	131
Sets	131
DAZZLER Control Statements	135
z/OS JCL Requirements for DAZZLER	137
z/VSE JCS Requirements for DAZZLER	137
12 Managing ADL Files	139
Overview	140
Reorganization with ADL Utilities (Pseudo Mixed Mode)	140
Reorganization with ADL Utilities (Normal Mode)	143
Reorganization with Adabas Utilities	145
Reorganization with User-written Programs	146
Change of DL/I Segment Definitions	147
Change of DL/I Field Definitions	148
Change of the Data Layout	149

Re-establishment of the Hierarchical Sequence	153
Change of DL/I PSB Definitions	153
Change of the ADL Structure	153
Change of Adabas DBIDs and File Numbers	154
Change of the Adabas File Layout	156
13 Performance Considerations	159
Introduction	160
Data Processing and Pointers	161
Initial Load and Reload the Database	163
Using the Adabas Multifetch Feature	164
Insert and Delete	166
Splitting a DBD	167
Field Definitions	167
Enqueue Logic	168
CICS/Batch Communication	168
Application Program and DB Design	168
14 Recovery and Restart Procedures	173
Introduction	174
Extensions and Restrictions	174
Adabas User Types	175
Automatic ET Calls Issued by ADL	179
Restart/Recovery Logic under ADL	180
ADL Actions for Basic and Symbolic Checkpoints	185
How to Restart a Batch Program	186
15 Appendix A - DAZZLER Test Stream	189
16 Appendix B - z/OS Dataset Usage	195
17 Appendix C - z/VSE Dataset Usage	197
Index	199

Interfaces

This documentation describes the installation and operation of the Adabas Bridge for DL/I (ADL) Interfaces for DL/I and Adabas applications.

The following topics are covered:

Introduction

Batch Installation and Operation

CICS Installation and Operation

IMS/TP Installation and Operation

ADL Online Services

Precompiler for "EXEC DLI" Commands

Using ADL Files with Natural/Adabas

Converting "Natural for DL/I" Programs

SQL Access to the Migrated Files

Debugging Aids - ADL Trace Facility

CALLDLI Test Program - DAZZLER

Managing ADL Files

Performance Considerations

Recovery and Restart Procedures

Appendix A - DAZZLER Test Stream

Appendix B - z/OS Dataset Usage

Appendix C - z/VSE Dataset Usage

1 Introduction

- Operational Environments 5
- Other Documentation You May Need 5
- Documentation Related to non-SAG Products 6

This documentation describes the installation and operation of the Adabas Bridge for DL/I (ADL) Interfaces for DL/I and Adabas applications. It will be needed in the phase following the conversion of DL/I databases into ADL files.

The general installation of the ADL load and source libraries, the ADL Directory file and ADL Natural utilities are described in the *ADL Installation* documentation. This documentation also explains how to customize ADL to your site by means of specifying parameters for the ADL parameter module.

The Adabas Bridge for DL/I consists of essentially six major functional units, namely

- ADL Data Base Conversion Utility
- ADL Directory file
- ADL Online Services
- CALLDLI Interface
- Consistency Interface
- ADL Installation Verification Package

The ADL Data Base Conversion Utility allows you to convert your DL/I DBD and PSB descriptions into Adabas file descriptions and to convert data stored in DL/I data bases into data stored in Adabas files automatically. Whenever we talk about Adabas files which are based on the conversion of DL/I data bases we use the term “ADL file” to point out the special properties of these files. The information about the original DL/I structures and how they are translated into Adabas definitions is kept in the ADL Directory file. The contents of this file can be displayed with the ADL Online Services, which additionally allow you to maintain the ADL interfaces under CICS.

The other two functional units of ADL are the CALLDLI Interface and the Consistency Interface. The CALLDLI Interface allows DL/I applications to access ADL files in the same way as original DL/I data bases. The Consistency Interface provides access to ADL files from Natural programs or with Adabas direct calls. This interface preserves the hierarchical structure of the data, which is of importance for ongoing DL/I applications.

The ADL Installation Verification Package provides a DL/I application environment including an example database. The ADL Installation is verified and the conversion of the example database is practiced. Several DAZZLER test streams, Cobol and Assembler program are included as well as Natural applications for testing the ADL Consistency. Terms and concepts of DL/I and Adabas are introduced and it is shown how ADL maps one to the other. The ADL Installation Verification Package is described in details in the section *ADL Installation Verification Package* in the *ADL Installation* documentation.

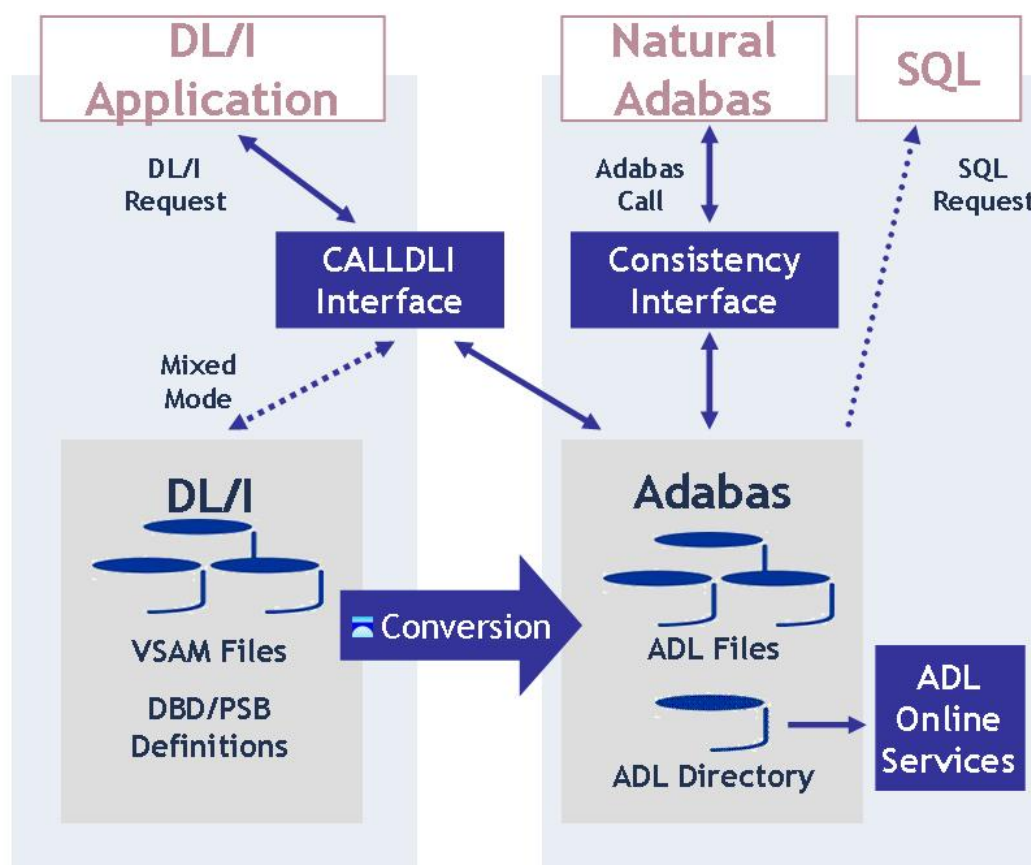


Figure 1: Functional units of ADL and their interrelation

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

This documentation provides you with the information necessary to operate both the CALLDLI Interface and the Consistency Interface once ADL has been installed and the DL/I databases have been converted.

This documentation is intended primarily for the system programmer responsible for the operation of Natural/Adabas and/or DL/I applications in batch and online systems. The section [Using ADL Files with Natural/Adabas](#) is of particular interest for application programmers. Similarly, the sections [Managing ADL Files](#), [ADL Online Services](#) and [Recovery and Restart Procedures](#) will be of interest to DBAs.

The first sections of this documentation cover the installation and operation of the ADL Interfaces for batch, CICS and IMS/TP.

The section [ADL Online Services](#) describes how to examine the contents of the ADL Directory file and maintain the ADL Interfaces for CICS.

The section *Precompiler for "EXEC DLI" Commands* describes how to use the ADL-supplied precompiler for "EXEC DLI" commands. Although this precompiler is not a direct part of the ADL Interfaces, it might be required in case DL/I and thus the High Level Programming Interface (HLPI) is not available.

The section *Using ADL Files with Natural/Adabas* describes the way in which ADL files can be accessed by Natural/Adabas applications.

The section *Converting "Natural for DL/I" Programs* describes what must be considered when converting an NDL program to a "normal" Natural/Adabas program.

The section *SQL Access to the Migrated Files* describes how SQL applications can access the migrated data with the Adabas SQL Gateway.

The section *Debugging Aids - ADL Trace Facility* covers the ADL Interfaces trace facility, which was designed to debug your applications as well as to track down problems related to the ADL Interfaces themselves.

A program to test DL/I calls in batch, DAZZLER, is described in the section *CALLDLI Test Program - DAZZLER*.

The management of converted DL/I databases is described in the section *Managing ADL Files*. As seen from an Adabas DBA point of view, there are a few things to be considered when managing ADL files with Adabas utilities. Also, this section explains how and in which way the DL/I structure or the Adabas file layout may be altered.

The tuning of ADL is described in the section *Performance Considerations*.

Obviously the conversion from DL/I to Adabas affects the restart and recovery scenarios. The section *Recovery and Restart Procedures* explains these changes.

Most of the information contained in this documentation refers to both operating systems z/OS and z/VSE. Information which applies only to a single operating system is clearly marked as such.

In order to install and operate the ADL Interfaces, a certain degree of knowledge of both data base systems, Adabas and DL/I, is required as well as a familiarity with the operating systems and TP monitors. This manual makes frequent usage of terms, synonyms, abbreviations and facts related to these systems. For clarity a glossary is provided with the *ADL Messages and Codes* documentation.

This chapter covers the following topics:

Operational Environments

The Adabas Bridge for DL/I operates in both online and batch environments. The online environments currently supported are CICS Versions 3.2 and above. Programming languages which use the standard DL/I call interface, e.g. COBOL, Assembler, and PL/I, are supported in both online and batch.

Operating system environments currently supported are: z/VSE and z/OS. The JCL/JCS examples given in this documentation are tailored for z/VSE and z/OS.

Batch Operation

Once all DL/I data bases accessed by a particular application program have been converted, the only other thing you need to do to operate the Adabas Bridge for DL/I in batch environments is to remove the JCL statements for the DL/I data bases and add input cards with ADARUN parameters. These are needed since the application has become an Adabas application program. The way in which the PSB name and application program names are specified, does not change. See the section [Batch Installation and Operation](#) for further details.

Online Operation (CICS)

Once all DL/I data bases have been converted, you may remove all JCL statements and FCT and ACT entries relating to them, as these are no longer required. Instead, you must generate a table of all the PSBs needed by the CICS application programs. Under z/OS CICS the ADL requires the installation of an SVC to operate properly. See the section [CICS Installation and Operation](#) for further details.

IMS/TP

Once all DL/I data bases have been converted, you may remove any JCL statements relating to them, as these are no longer required. See the section [IMS/TP Installation and Operation](#) for further details.

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 DBA Reference* documentation

For a complete list of Software AG documentation, prices, and ordering information refer to the [Software AG Documentation Overview](#) or contact your Software AG support representative.

Documentation Related to non-SAG Products

The documentation mentioned below might be of interest and helpful for the installation and operation of the ADL interfaces.

For z/OS users:

- IMS/VS Application Programming
- IMS/VS Application Programming for CICS/VS Users
- IMS/VS Utilities Reference documentation
- CICS TS Installation Guide
- CICS TS Operations and Utilities Guide
- CICS TS Resource Definition Guide

For z/VSE users:

- DL/I DOS/VS Guide for New Users
- DL/I DOS/VS Application Programming: CALL and RQDLI Interface
- DL/I DOS/VS Application Programming: High Level Programming Interface
- DL/I DOS/VS Utilities and Guide for the System Programmer
- DL/I DOS/VS Resource Definition and Utilities
- CICS TS Installation Guide
- CICS TS Operations and Utilities Guide
- CICS Resource Definition Guide

2 Batch Installation and Operation

- CALLDLI Interface 8
- Consistency Interface 12
- User Exit 16
- z/OS JCL Requirements 17
- z/VSE JCS Requirements 18

This chapter covers the following topics:

CALLDLI Interface

The ADL CALLDLI Interface processes all data base requests originating from DL/I calls or command level programs. For the application, it provides an interface identical to original DL/I. The CALLDLI Interface decides dynamically, whether a request is to be routed to DL/I or to be processed by ADL. In the first case, it simply acts as a front-end to DL/I.

ADL may be executed in batch in two different modes:

■ **Normal Mode**

This is used for application programs using PSBs which only reference DBDs describing previously converted DL/I data bases.

■ **Mixed Mode**

This is used for application programs using PSBs which reference at least one DBD describing an unconverted DL/I data base.

In normal mode, all requests are processed by ADL. In mixed mode, requests are routed to DL/I if they refer to non-converted DL/I data bases. GU and ISRT calls on the IOPCB and CHKP calls will be routed to ADL as well as to DL/I.

Positional Parameters for DAZIFP

All DL/I parameters will be ignored, with the exception of the positional parameters given below. These will be accepted and interpreted by DAZIFP in the same way as DL/I.

The syntax for the input statement is as follows:

```
runmode , pgmname , psbname [ , keyword ]
```

These parameters are explained on the next page.

Parameter	Description
runmode	A three-character string indicating the run mode. The possible values are described in the following table below.
pgmname	The name of the application program to be loaded and started.
psbname	The name of the PSB used by the application program (if omitted, it is assumed that the PSB name is the same as the application program name).

Run Mode	Description
BMP/MPS	Indicates a multi-user batch run. Programs using this mode of operation can run concurrently with other programs (batch or online). An IO-PCB is added in front of the PCBs. Automatic ET calls may be issued by the ADL. Checkpoints may be issued and will result in Adabas ET calls. See also the section <i>Recovery and Restart Procedures</i> . Compatible with DL/I BMP/MPS.
ELO	Indicates an Establish Logical Relationship utility run.
IMS/DLI	Indicates a normal run. Batch update programs using this mode of operation cannot run concurrently with any other program (batch or online) using the same data base(s). Read-only programs can. Checkpoints may be issued and will result in Adabas unsynchronized checkpoints being taken (C1 calls). See also the section <i>Recovery and Restart Procedures</i> . Compatible to DL/I IMS/DLI.
MIX	Indicates that the batch program is to be run in mixed mode. The same rules apply as for IMS/DLI mode.
MPX/SDX	Indicates an MPS/SDB batch run in mixed mode.
MSG	Indicates an IMS/TP message region processing run.
PRE	Indicates an ADL precompiler run for translating command level programs.
PRT	Indicates a Print Trace utility run.
REF	Indicates a Reformat utility run.
SDB	Indicates a shared-data base run. Programs running in this mode may access data bases concurrently with other programs (batch and online). This mode of operation is provided for compatibility with the DL/I "shared data bases" feature. The ET checkpoint logic is like BMP/MPS, but no IO-PCB is added to the PCBs.
SHI	Indicates a DAZSHINE utility run.
STA	Indicates a stand-alone batch run. Programs using this mode of operation can not run concurrently with online programs using the same data base(s), nor with any other batch programs. The ADL directory file is used exclusively. Checkpoints may be issued and will result in Adabas unsynchronized checkpoints being taken (C1 calls). See also the section <i>Recovery and Restart Procedures</i> .
UNL	Indicates an Unload utility run.
UTC	Indicates a CBC utility run.

Keywords for DAZIFP

In addition to the positional parameters mentioned above, you may specify one or more keywords to control operation of ADL.

Any keywords specified must immediately follow the positional parameters. Any keywords not on the list (e.g. the original DL/I parameters) encountered after the program and PSB names will cause scanning to be terminated. In this case, all other parameters will be ignored.

If the parameters are read in from a file (i.e., DAZIN1, DAZIN2 or SYSIPT), more than one line can be used for the parameter specification. A comma followed by a blank after the last keyword of the line indicates that the specification of the parameters is continued in the next line. Any character found after the blank in the initial line is treated as a comment.

The table below gives a short explanation of the keywords which may be specified. For further information (including defaults) on all parameters, see the section *ADL Parameter Module* in the *ADL Installation* documentation.

Keyword	Description
BUFSF	Suffix for the buffer table module name.
CHKPMSG	Checkpoint message destination.
CPID	The checkpoint ID from which the application is to be restarted.
DBD	The size (in kilobytes) of the ADL DBD ICB buffer.
DBDSF	Suffix for the DBD table module name.
DBID	Adabas data base for the ADL directory file.
DUO	Support of CA-DUO under z/OS.
EBUF	The size (in kilobytes) of the ADL ECB buffer.
ET	The number of times a different root segment occurrence may be accessed before an Adabas ET command is issued (BMP/MPS/SDB programs only).
FNR	Adabas file number for the ADL directory file.
FX	z/VSE only. Specifies the input data set for the Print Utility when printing the routine trace or for the ADL precompiler.
IMSY	IMS/TP syncpoint/Adabas ET synchronization.
LANG	Language of the application program. Overwrites the PSB language parameter.
LCS	The size (in kilobytes) of the last call save area.
LOAD	Indicates whether an ISRT against a PCB with PROCOPT=L is to be written to Adabas or to a sequential file.
MFT	Defines the ADL Multifetch Table.
OPENRQ	Specifies if an Adabas OPEN is required or not.
PLI	z/OS only. Passes PL/I parameters to PL/I. See the explanation on the next page.
PLILE	z/VSE only. Specifies whether PL/I uses the LE/VSE.
PR	z/VSE only. Specifies the number of logical printers.
PSB	The size (in kilobytes) of the ADL PSB ICB buffer.
PSBSF	Suffix for the PSB table module name.
RBE	Specifies the record buffer extension list.
RETRY	The number of times that ADL tries to put a record, which is currently held by another user, into hold status.
SQ	z/VSE only. Specifies the input/output data set for the Unload utility.
STACK	The size (in kilobytes) of the ADL internal subroutine stack.
TRACE	Activates the Trace facility and specifies what is to be traced. This keyword is described in the section Debugging Aids – the ADL Trace Facility .
UTI	Specifies the CBC utility work area.

PLI Parameter (z/OS Only)

This keyword is used to pass PL/I parameters to PL/I. The feature may only be specified if the PL/I application program has been link edited with “PLICALLB” as its entry point. Specifying “PLI” or “PLI=(...)” will cause the PL/I program to be called in conformity with the standards for calling the entry point “PLICALLB”.

Syntax

```
PLI =(options)
```

where *options* are PL/I execution options.

The following values may be entered:

REPORT, NOREPORT, SPIE, NOSPIE, STAE, NOSTAE.

For further details, see the IBM documentation *PL/I Optimizing Compiler: Programmer's Guide*.

Normal Mode Batch Execution

To invoke normal mode batch execution under ADL, execute the DAZIFP Bridge initialization program. This program requires input parameters similar to those for the DL/I initialization program. Both positional and keyword parameters exist (see above).

Neither the JCL describing the original DL/I data bases nor the load libraries containing the DL/I load modules are required in normal batch execution.

The ADL load library containing the executable ADL batch module and the Adabas load library must be included in the JCL. ADARUN control statements must also be provided, as is the case with any Adabas application program. For a detailed description of JCL requirements, see the end of this section.

Unlike DL/I, ADL opens all files which are referenced in the PSB during initialization. This must be considered, particularly if the file is opened in EXU mode and the application does not access the file at all.

Mixed Mode Batch Execution

To invoke mixed mode batch execution under ADL, execute the normal DL/I initialization program.

You still require all the JCL describing the original DL/I data bases and the load libraries containing the DL/I load modules.

Before running the program, make the following changes in the JCL/JCS:

1. Change the name of the application program in the DL/I parameters from the user program to DAZIFP. No other parameters have to be modified.

2. Insert an extra input statement for DAZIFP:
 - **z/OS Environments**
Include a DD statement with the DD name DAZIN2 in the JCL.
 - **z/VSE Environments**
The mixed mode control statement is read from SYSIPT.
3. Add a JCL/JCS statement for the ADARUN file "DDCARD".

For a more detailed description of the JCL/JCS requirements for mixed mode, see the end of this section.

The parameters and syntax for the input statement are the same as for normal mode batch execution (see the first topic of this section).



Note: DL/I and Adabas are not fully synchronized. Therefore, you should carefully consider restart and recovery procedures for programs running in mixed mode, especially when they are running concurrently to other programs (MPX or SDX run mode).

Pseudo Mixed Mode Batch Execution

In the "pseudo" mixed mode batch execution, the ADL interface program DAZIFP calls a copy of itself. This can be useful when changing structures after the conversion. Pseudo mixed mode is described in details in the section [Managing ADL Files](#).

Link-editing of Application Programs

In general, application programs do not have to be re-linked.

The ADL load library provides the language interface module DAZLIBAT as a substitute for the IBM language interface DFSLI000 (z/OS) and DLZLI000 (z/VSE). These modules provide the entry points ASMTDLI, PLITDLI, CBLTDLI and FORTDLI. You might need the ADL provided language interface in case DL/I is not available at your site, and you have to re-link application programs.

Consistency Interface

This paragraph describes how the ADL Consistency Interface intercepts Adabas calls and how you activate the Consistency in a batch environment.

Activating the Consistency Interface

The ADL Consistency Interface intercepts Adabas calls on the Adabas link module level. In batch, this is a module named ADALNK.

A substitute for the Adabas link module in batch, ADALNK, is delivered with the ADL installation tape. You will have to assemble and link-edit this module, as described in the section below.

Operation of the Consistency Interface requires the original Adabas link module to be present. This, however, has to be renamed. The default new name for the original Adabas link module is ADAOLK. When customizing the ADL substitute, you may choose any new unique name for the original Adabas link module.

Consequently, you have to rename the original Adabas link module for batch, ADALNK, to ADAOLK (or to whatever is the new name you choose.)

It is recommended that the ADL supplied substitute and the renamed copy of the original Adabas link module are both stored on the ADL load library.

For the Consistency Interface to become active, the order of the concatenation of the ADL and Adabas load libraries is of importance. For z/OS batch jobs, this means that the ADL load library must be concatenated before the Adabas load library in the JOBLIB or STEPLIB DD statements.

Example:

```
//STEPLIB DD DISP=SHR,DSN=ADL.LOAD
//          DD DISP=SHR,DSN=ADABAS.LOAD
//          DD DISP=SHR,DSN=.....
```

In z/VSE, the ADL load library must be defined in the search sequence of a LIBDEF statement before the Adabas load library.

Example:

```
// LIBDEF *,SEARCH=(ADL,ADABAS, ... .)
```

Customizing the Consistency Interface

The ADL source library contains the source of the substitute for Adabas link module in batch, ADALNK. The name of this source member is DAZLNK.

All parameters for this module are given as assembler variables and their meaning is explained in comment statements in the source.

All parameters correspond to customizable parameters in the original Adabas link module for batch. Note, however, that you do not have to specify an SVC nor a default DBID when you cus-

tomize the ADL link module substitute. The values for these will dynamically be taken from the original Adabas link module.

The assembly of the ADL link module substitute requires the ADL source library and the Adabas supplied macro library as well as the system macro library to be available.

See the sections *z/OS Installation* and *z/VSE Installation* in the *ADL Installation* documentation for more details on how to assemble and link-edit the ADL link module.

Parameters for the Consistency Interface

This paragraph describes which parameters are of importance for the Consistency Interface and how you may initialize them. The meaning of the individual parameters, their possible values and their default values are described in the section *ADL Parameter Module* in the *ADL Installation* documentation.

Most of the parameters for the *Consistency Interface* will be initialized by the assembly and link-edit of the ADL parameter module. Parameters of particular importance for the *Consistency Interface* are:

FDT	The size of the "file description table"
FSTAC	The size of the internal format buffer stack
PARM	Do (not) read dynamic parameters
RBSIZ	The size of the internal record buffer
SDT	The size of the "segment description table"

When you have specified "PARM=YES" in the assembly of the ADL parameter module, the *Consistency Interface* will read a card containing dynamic parameters from the file DAZIN1 (z/OS) or the logical unit SYSIPT (z/VSE). The layout of this card is:

```
Keyword=value,...
```

Keywords must be separated by commas, no intervening or leading blanks are allowed. Any undefined keyword causes the scanning to be terminated and the following statements to be ignored.

As described in the section [Keywords for DAZIPF](#) earlier in this section, more than one line can be used for the parameter specification. You may specify the following dynamic parameters:

DBD	The size (in kilobytes) of the internal control block area for DBDs
DBID	The DBID of the ADL directory file
EBUF	The size (in kilobytes) of the external control block area
FNR	The FNR of the ADL directory file
OPENRQ	Adabas "OP" (not) required
PSB	The size (in kilobytes) of the internal control block area for PSBs
STACK	The size (in kilobytes) of the internal subroutine stack
TRACE	Specify trace parameters. This keyword is described in the section Debugging Aids – the ADL Trace Facility in this documentation.

For a detailed explanation of these parameters, see the *ADL Installation* documentation, section *ADL Parameter Module*.

Table of Converted Adabas Files (DAZTCF)

In order to minimize the effort for batch programs accessing only native Adabas files (i.e. files which do not result from a conversion of a DL/I data base), you may assemble a table containing a list of the converted files. Link-edit it together with the ADL substitute for the Adabas batch link module, ADALNK.

You may use the JCL(JCS) in the member ADLTFC (ADLTFC.J) as an example of how to assemble the table under z/OS (z/VSE).

To create a table of converted files, DAZTCF, you assemble a source containing one call to the ADL supplied macro DAZTCF for each DBID to be included in the table.

The two keyword parameters of the DAZTCF macro are:

DBID=	the DBID to be included in the table
FILES=	list of file numbers to be included in the table

For the FILES= parameter, there are five forms of specifying the argument:

FILES=x	a single file to be included in the table
FILES=(x, ...)	a single file as part of a list
FILES=((x), ...)	a single file as part of a list
FILES=((x,y), ...)	a range of file numbers from x to y inclusive
FILES=ALL	all files for a given DBID

Any number of files may be specified with the FILES= parameter.

DBIDs and file numbers must be specified in ascending sequence. The same DBID can be specified more than once in the list.

Which files are to be included in the table of converted files, DAZTCF, should be evaluated very carefully. Any call referencing a DBID/FNR combination not included in the DAZTCF table will be routed directly to Adabas. Thus, if an ADL file is not included in this table, data integrity will *not* be maintained by the Consistency Interface. On the other hand, the ADL directory file and all Adabas, Natural etc. system files should not be included in the DAZTCF table.

The DAZTCF table can also be used under CICS. Refer to the section CICS Installation and Operation for more details.



Note: DAZTCF replaces the ADL 2.2 table of non-converted files DAZNCF which is no longer supported.

Example:

```
DAZTCF DBID=001,FILES=ALL
DAZTCF DBID=002,FILES=((14,20),25,(100,200))
DAZTCF DBID=005,FILES=(1060)
END
```

Summary

In short, to activate the ADL Consistency Interface for Adabas or Natural batch jobs you have to perform the following steps:

- create the ADL substitute for the Adabas link module,
- rename the original Adabas link module for batch, ADALNK,
- concatenate the ADL load library before the Adabas load library,
- add a DD card for DAZIN1 (z/OS) or an extra input statement on SYSIPT (z/VSE) when you intend to read dynamic parameters.

Besides that, Adabas or Natural batch jobs run unchanged as described in the corresponding Adabas and Natural documentations.

User Exit

ADL Interfaces optionally pass control to a user written routine (user exit) before each call to Adabas. This user exit may be used for monitoring purposes as well as to apply modifications to the call parameters before Adabas receives control. The conventions for this user exit and how it is activated is described in the section *Miscellaneous* in the *ADL Installation* documentation.

z/OS JCL Requirements

The JCL requirements for batch operation are discussed in the following pages. Both normal mode and mixed mode batch operation are described.

Normal Mode Batch Operation

The following table lists the data sets used by the ADL batch monitor when application programs are run in normal mode.

DDname	Medium	Description
DAZOUT1	Printer	Messages and codes.

Example

```
//G          EXEC PGM=DAZIFP,PARM='DLI,pgmname,psbname'
//STEPLIB   DD DISP=SHR,DSN=ADL.LOAD
//          DD DISP=SHR,DSN=ADABAS.LOAD
//DDCARD    DD *
ADARUN PROGRAM=USER,...
//DAZOUT1   DD SYSOUT=X
//*
//*  application program datasets
//*
```

Mixed Mode Batch Operation

The following table lists the data sets used by the ADL batch monitor when an application program is run in mixed mode.

DDname	Medium	Description
DAZIN2	Reader	Control input for the ADL batch monitor, DAZIFP.
DAZOUT1	Printer	Report, messages and codes.

Example

```
//          EXEC DLIBATCH,MBR=DAZIFP,PSB=psbname
//G.STEPLIB DD
//          DD
//          DD DISP=SHR,DSN=ADL.LOAD
//          DD DISP=SHR,DSN=ADABAS.LOAD
//*
//*  datasets describing DL/I data bases
//*
```

```
//G.file DD .....  
//*  
//G.DDCARD DD *  
ADARUN PROGRAM=USER,...  
//G.DAZOUT1 DD SYSOUT=X  
//G.DAZIN2 DD *  
MIX,pgmname,psbname  
//*  
//* application program datasets  
//*
```

DLIBATCH is the standard batch procedure provided by IBM as part of the IMS/DB installation.

z/VSE JCS Requirements

The JCS requirements for batch operation are discussed in the following pages. Both normal mode and mixed mode batch operation are described.

Normal Mode Batch Operation

The following table lists the files used by the ADL batch monitor when application programs are run in normal mode.

DTF	Logical Unit	Medium	Description
DAZIN1	SYSIPT	Reader	Control input for the ADL batch monitor, DAZIFP.
DAZOUT1	SYSLST	Printer	Messages and codes.

The control input for the batch monitor (DAZIFP) and for ADARUN is read from SYSIPT. The control statements must be specified in the following order:

```
DLI,pgmname,psbname,... input for DAZIFP  
/*  
ADARUN DB=dbid,MO=MULTI,PROGRAM=USER,... input for ADARUN  
/*  
user input  
.  
.  
/*
```

Example

```
// EXEC PROC=ADLLIBS
// EXEC DAZIFP
DLI,pgmname,psbname
/*
ADARUN PROGRAM=USER,...
/*
/ &
```

Mixed Mode Batch Operation

The following table lists the files used by the ADL batch monitor when an application program is run in mixed mode.

DTF	Logical Unit	Medium	Description
DAZIN1	SYSIPT	Reader	Control input for the ADL batch monitor, DAZIFP.
DAZOUT1	SYSLST	Printer	Report, messages and codes.

The control input for the batch monitor (DAZIFP), for ADARUN and for the DL/I initialization program, DLZRR00, is read from SYSIPT. The control statements must be specified in the following order:

```
DLI,DAZIFP,psbname,...           input for DLZRR00
MIX,pgmname,psbname,...         input for DAZIFP
ADARUN DB=dbid,MO=MULTI,PROGRAM=USER,...  input for ADARUN
/*
user input
.
.
/* ↵
```

Example

```
// EXEC PROC=ADLLIBS
// EXEC DLZRR00
DLI,DAZIFP,psbname
MIX,pgmname,psbname
ADARUN PROGRAM=USER,...
/*
/ &
```


3 CICS Installation and Operation

- Overview CICS Installation 22
- Prerequisites for z/OS CICS Installation 22
- Prerequisites for z/VSE CICS Installation 24
- Important Note for CICS Users 27
- Generating the Runtime Control Tables 27
- Tuning and Maintaining the Runtime Control Tables 37
- z/OS Requirements 38
- z/VSE Requirements 39
- Activating and Controlling the ADL Interfaces 40
- CALLDLI Interface 42
- Consistency Interface 44
- User Exit 46

This chapter covers the following topics:

Overview CICS Installation

To install the ADL interfaces in CICS under z/OS or z/VSE, perform the steps listed below.

Step	Description
Step 1	Run ADLCSD against the DFHCSDUP utility.
Step 2	Add entry to the CICS PLT (optional).
Step 3	Add JCS/JCL to the CICS start-up job.
Step 4	Edit, assemble and link edit the CICS runtime control tables.
Step 5	Install the ADL SVC for CICS (z/OS only). Add entries to the CICS DCT (z/VSE only).
Step 6	Generate the ACT (z/VSE only).

The individual steps in the CICS procedures are now explained in more detail.

Prerequisites for z/OS CICS Installation

Step 1

Use the member ADLCSD in the ADL source library as input for the CICS DFHCSDUP utility. This member contains all program, transaction and transient data queue entries required for ADL. It can be used as delivered or modified to satisfy the local requirements. In particular the following entries are mandatory:

```
PROGRAM(DAZCICS)
PROGRAM(DAZNUCC)
PROGRAM(DAZSYNC)
PROGRAM(DAZPSB)
PROGRAM(DAZDBD)
PROGRAM(DAZBUF)
TDQUEUE (DAZP)
```

The following entries are optional:

```

PROGRAM(DAZCINIT)
PROGRAM(DAZCEND)
PROGRAM(DAZCINF)
PROGRAM(DAZCTON)
PROGRAM(DAZCTOFF)
PROGRAM(DAZCDUMP)
TRANSACTION(DAZI)
TRANSACTION(DAZE)
TRANSACTION(DAZS)
TRANSACTION(DAZT)
TRANSACTION(DAZO)
TRANSACTION(DAZD)
TDQUEUE (DAZD)
TDQUEUE (DAZR)

```

If your DBDs are defined with user-supplied index maintenance exit routines, you have to add PROGRAM-entries for each of these routines. See the section *User-supplied Index Maintenance Exit Routines* in the *ADL Installation* documentation for more information.

Make sure that all programs in the ADLCSD are accessible by CICS. This can be achieved by putting the members into the ADL load library and by adding this library to the CICS load libraries.



Note: When you run under CICS TS 2.3 or below, the ADL.LC23 library must be concatenated in front of the ADL load library.

Step 2 (optional)

If you want the ADL to be initialized automatically when CICS is started up, you must add the following entry to the CICS PLT:

```
DFHPLT TYPE=ENTRY ,PROGRAM=DAZCINIT
```

For automatic shutdown of the ADL when CICS is shut down, you must add the following entry to the CICS PLT:

```
DFHPLT TYPE=ENTRY ,PROGRAM=DAZCEND
```

Step 3

Add the following JCL statements to the CICS start-up job(s):

```
//DAZOUT2 DD DSN=dsname, . . . . .  
//          DCB=(BLKSIZE=nnnn, LRECL=132, DSORG=PS, RECFM=FB)
```

If you want to run the trace facility under CICS, you must also add the following JCL statements to the CICS start-up job:

```
//DAZOUT1 DD DSN=dsname, . . . . .  
//          DCB=(BLKSIZE=nnnn, LRECL=132, DSORG=PS, RECFM=FB)  
//DAZOUT5 DD DSN=dsname, . . . . .  
//          DCB=(BLKSIZE=nnnn, LRECL=8192, DSORG=PS, RECFM=VB)
```



Note: “dsname” can be any valid data set name. Block Sizes: any legal block size may be used.

Step 4

When using ADL under CICS, you need to create the three additional runtime control tables DAZPSB, DAZDBD and DAZBUF as described in the section [Generating the Runtime Control Tables](#).

Step 5

Copy the ADL SVC program DAZCSVC to an LPA list library. Add the following entry for the ADL SVC to the SVC table:

```
SVC Parm nnn, REPLACE, TYPE(3), EPNAME(DAZCSVC), APF(NO) /* ADL SVC */
```

where *nnn* is a free SVC number in the range of 200-255.

Prerequisites for z/VSE CICS Installation

Step 1

Use the member ADLCSD in the ADL source library as input for the CICS DFHCSDUP utility. This member contains all program and transaction entries required for ADL. Remove the transient data queue entries if the TDQUEUE keyword is not supported by the CICS level in use. The ADLCSD member can be used as delivered or modified to satisfy the local requirements. In particular the following entries are mandatory:


```

PROGRAM(DAZCICS)
PROGRAM(DAZNUCC)
PROGRAM(DAZSYNC)
PROGRAM(DAZPSB)
PROGRAM(DAZDBD)
PROGRAM(DAZBUF)
TDQUEUE (DAZP) - if allowed

```

The following entries are optional:

```

PROGRAM(DAZCINIT)
PROGRAM(DAZCEND)
PROGRAM(DAZCINF)
PROGRAM(DAZCTON)
PROGRAM(DAZCTOFF)
PROGRAM(DAZCDUMP)
TRANSACTION(DAZI)
TRANSACTION(DAZE)
TRANSACTION(DAZS)
TRANSACTION(DAZT)
TRANSACTION(DAZO)
TRANSACTION(DAZD)
TDQUEUE (DAZD) - if allowed
TDQUEUE (DAZR) - if allowed

```

If your DBDs are defined with user-supplied index maintenance exit routines, you have to add PROGRAM-entries for each of these routines. See the section *User-supplied Index Maintenance Exit Routines* in the appendix of the *ADL Installation* documentation for more information.

Make sure that all programs in the ADLCSD are accessible by CICS. This can be achieved by putting the members into the ADL load library and by adding this library to the CICS load libraries.

Step 2 (optional)

If you want the ADL to be initialized automatically when CICS is started up, you must add the following entry to the CICS PLT:

```
DFHPLT TYPE=ENTRY ,PROGRAM=DAZCINIT
```

For automatic shutdown of the ADL when CICS is shut down, you must add the following entry to the CICS PLT:

```
DFHPLT TYPE=ENTRY,PROGRAM=DAZCEND
```

Step 3

Add the following JCS statements to the CICS start-up job(s):

```
// DLBL DAZOUT2,'filename',....  
// EXTENT SYSnnn,....
```

If you want to run the Trace facility under CICS, you must also add the following JCS statements to the CICS start-up job:

```
// DLBL DAZOUT1,'filename',....  
// EXTENT SYSnnn,....  
// DLBL DAZOT5D,'filename',....  
// EXTENT SYSnnn,....
```

Step 4

When using ADL under CICS, you need to create the three additional runtime control tables DAZPSB, DAZDBD and DAZBUF as described in the section [Generating the Runtime Control Tables](#).

Step 5

If the TDQUEUE keyword of the DFHCSDUP utility is not available in the CICS in use, add the following entries to the CICS DCT:

```
DFHDCT TYPE=SDSCI,DSCNAME=DAZOUT2,BLKSIZE=nnnn,RECSIZE=132,  
        RECFORM=FIXBLK,TYPEFLE=OUTPUT,DEVICE=DISK  
  
DFHDCT TYPE=EXTRA,DESTID=DAZP,DSCNAME=DAZOUT2,OPEN=DEFERRED
```

If you want to run the Trace facility under CICS, you must also add the following entries to the CICS DCT:

```
DFHDCT TYPE=SDSCI,DSCNAME=DAZOT5D,BLKSIZE=8196,RECFORM=VARBLK,  
        TYPEFLE=OUTPUT,DEVICE=DISK  
  
DFHDCT TYPE=SDSCI,DSCNAME=DAZOUT1,BLKSIZE=nnnn,RECSIZE=132,  
        RECFORM=FIXBLK,TYPEFLE=OUTPUT,DEVICE=DISK  
  
DFHDCT TYPE=EXTRA,DESTID=DAZD,DSCNAME=DAZOT5D,OPEN=DEFERRED  
  
DFHDCT TYPE=EXTRA,DESTID=DAZR,DSCNAME=DAZOUT1,OPEN=DEFERRED
```



Note: Block Sizes: Any legal block size may be used.

Step 6

If you run application programs under CICS which do not explicitly specify the psbname in the scheduling call but use the default psbname as defined in the CICS application control table, you have to assemble and link-edit the ACT using the ADL-supplied macros.

Use the sample JCS in the ADL Source Library member ADLACT.J as an example (do not forget to specify `OPTION SYSPARM='DAZACT'`). The default name of the load module produced by this procedure is DAZACT. A two character suffix may be appended to this name if it is necessary to keep more than one version in the load library. In this case, the ACTSF parameter has to be set to make the two-character suffix known to ADL. See the section *ADL Parameter Module* in the *ADL Installation* documentation for details. The full name of the ACT (for example, DAZACT) followed by the suffix must be added to the ADLCSD member used as input for the DFHCSDUP utility.

Important Note for CICS Users

Any CICS transaction that uses ADL must have a TWASIZE of at least 24 bytes. ADL saves the 24 bytes of the TWA, uses them itself, and then restores their former contents. After this, it returns to the user. ADL does not use the TWA when PLINTWA=NO is specified while creating the parameter module. See the section *ADL Parameter Module* in the *ADL Installation* documentation for details.

Generating the Runtime Control Tables

As explained in the section *ADL Conversion Utilities for DBDs and PSBs* in the *ADL Conversion* documentation, the information corresponding to the DBDs and PSBs is stored as external control blocks on the ADL Directory file. When an application program running under CICS schedules a particular PSB, all external control blocks related to and the DBDs referenced by that PSB must be converted to internal control blocks. These internal control blocks are kept in memory, allowing faster access to the necessary information.

For performance reasons, the conversion from external to internal control blocks should not be done repeatedly each time a program issues a scheduling call. It would be most appropriate to convert the control blocks only once and keep them in memory. On the other hand, the storage available might not be sufficient to hold all internal control blocks at the same time.

The solution chosen by ADL allows optimal performance in smaller systems where all internal control blocks will be kept in memory, but has the flexibility to handle larger systems with nearly the same performance.

When ADL is activated under CICS, all DBD and PSB external control blocks that will be accessed in the online environment are converted to internal control blocks. All DBD-related internal control blocks are kept in the DAZDBD table. The PSB-related internal control blocks are written to the ADL Directory file. Whenever a PSB is scheduled for the first time, the related internal control blocks are rolled into the table DAZBUF and remain there until ADL is switched off.

If the space in the DAZBUF buffer table is not sufficient to store all PSB internal control blocks, PSBs that are currently not in use by any application program are purged and, in turn, the requested PSB is rolled into DAZBUF. The DAZPSB table is used to control the DAZBUF buffer table.

The DAZBUF buffer table is arranged in groups, each containing slots. All slots in a group are of the same size. Thus, if a group of slots are occupied and some PSBs are rolled out, the other groups are not affected.

In addition to the DBD-related internal control blocks, the DAZDBD table contains the task ID table, the local user blocks (LUB), the segment description table (SDT), the file description table (FDT) and the exit routine table (EXR). The task ID table is used to assign a specific task ID to the corresponding internal areas. The local user blocks are used as reentrant storage to reduce CICS getmain requests. With the help of the SDT and FDT tables, the ADL Consistency Interface translates Adabas database requests to internal DL/I calls. The exit routine table maintains the user exists.

Step 1: Generating the Table of PSBs (DAZPSB)

Assemble and link-edit the table of PSBs, DAZPSB, using as input either

- an existing DL/I Directory List, PSB-PDIR under z/OS, or
- an existing Application Control Table, DFHACT under z/VSE

Ensure that you use the ADL-supplied macro substitutes that are in the ADL source library on the installation tape.



Note: For z/VSE users: You must specify `OPTION SYSPARM='DAZPSB'` for the assembly of the ACT.

Alternatively, the DAZPSB module may be created using the ADL-supplied macros MGPSTIN, MGPSTEN and MGPSTFI, as shown in the following example:

```
MGPSTIN DATE=mm/dd/yy
Initialization (assembly date)
MGPSTEN NAME=psbname
          .
          .           as many PSB entries as needed
          .
MGPSTEN NAME=psbname
MGPSTFI           final macro to trigger table generation
END
```

An entry for an internally used Consistency PSB will be automatically generated in the DAZPSB table. The name of this PSB is ADL\$PSB.

You may use the JCL (JCS) in member ADLCTG1 (ADLCTG1.J) as an example of how to assemble and link-edit the DAZPSB table under z/OS (z/VSE).

The default name for the PSB table module is DAZPSB; however, a two-character suffix may be appended, if needed to keep more than one version in the load library. See the section *ADL Parameter Module* in the *ADL Installation* documentation for details on the PSBSF parameter. The full name (DAZPSB followed by the two-character suffix) must be included in the ADLCSD member used as input for the DFHCSDUP utility.

Step 2: Determining Requirements for the Runtime Control Tables

As soon as all PSBs that will be accessed in the online environment are known, the DBDs referenced by these PSBs and the amount of storage needed for the internal control blocks can be determined using the external control blocks stored on the ADL directory file.

The ADL utility DAZSHINE can be used to determine requirements for the DAZDBD and DAZBUF Runtime Control Tables. Figure 2a below shows the four-step table generation process performed by DAZSHINE. The inputs to DAZSHINE are the DAZPSB table, the ADL Directory file containing the DBD and PSB external control blocks, the ADL files and a control card in the following format:

```
MODE=xxxx,RANGE=(start,step)
```

where *xxxx* is

SHORT	a report on the status and size of all PSBs will be produced.
FULL	as above, but in addition, a detailed summary on the status of all PCBs contained in each PSB will be produced.
NONE	none of the above will be printed.

and *start* and *step* determine the starting point and size (in bytes) of the steps used to group the PSBs and produce the PSB size statistics. If you specify RANGE=(128,256) for example, possible slot sizes will be 128, 384, 640 etc. The defaults are: MODE=NONE, RANGE=(128,128).

The PSBs are only grouped into the given slot sizes, if the total size of all PSBs exceeds 512 Kb. Otherwise every PSB will have its own slot.

The output of DAZSHINE is a run report, statistics of the size and status of the PSBs contained in DAZPSB, and the source needed for the generation of the DAZBUF buffer table. The DAZSHINE run report describes each PCB contained in each PSB, all DBDs referenced by each PCB, and the status of each referenced DBD (that is, converted or not converted). Thus, you can determine which PCBs will access converted data bases and which ones will still access non-converted data bases.

CICS Table Generation Procedure

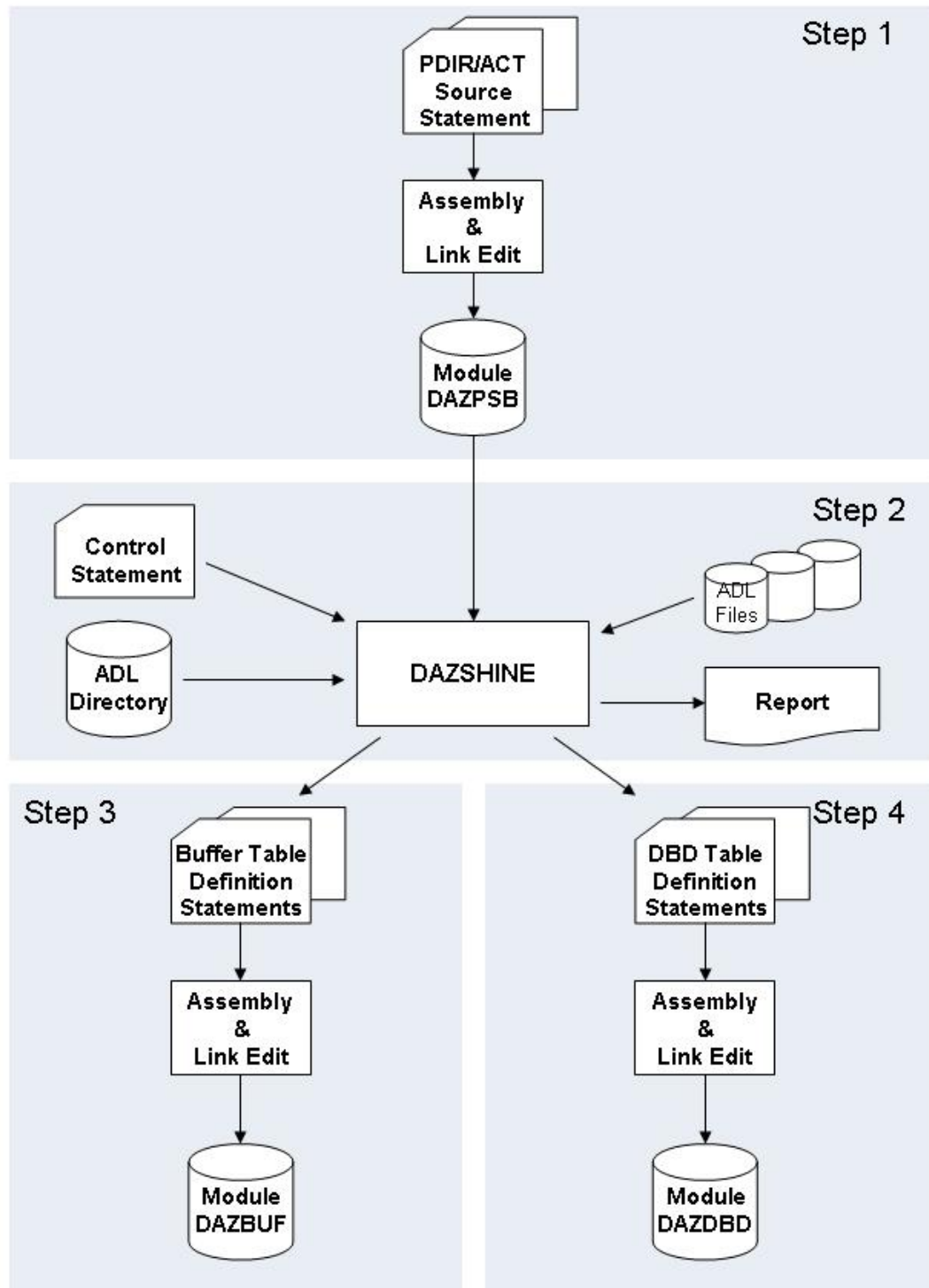


Figure 2a: CICS Table Generation

Note, however, that if a PCB references a logical DBD which is not converted, DAZSHINE cannot find the physical DBDs referenced by this logical DBD. It is therefore recommended that you convert the control blocks of all logical DBDs, even if the corresponding physical DBDs are not yet converted.

In order to determine the size requirement for the ADL FDT area in the DAZDBD table, DAZSHINE reads the FDTs of the ADL files. It uses the FDTs of all the files referenced by a physical DBD that was converted with `CONSI=YES` in the `GENDBD` statement (see the section *ADL Data Conversion Utilities* in the *ADL Conversion* documentation). A list of all these DBDs can be displayed and modified by the ADL Online Services (see the corresponding section in this documentation).

The output of DAZSHINE provides you with the DBIDs and FNRs of the selected ADL files and, if the corresponding FDT has been found, with the number of the Adabas fields in each file. Further the DBD, SDT and FDT area sizes will be displayed. These areas will be allocated with the given sizes in step 4, if the `DBDMAC` macro is assembled with the parameter values as calculated by DAZSHINE.

You may use the JCL (JCS) in the member `ADLCTG2` (`ADLCTG2.J`) as an example on how to execute the DAZSHINE utility under z/OS (z/VSE).

The default names for the PSB, DBD and buffer table modules are `DAZPSB`, `DAZDBD` and `DAZBUF`, respectively. A two-character suffix may be appended to each of these names if there is more than one version to be kept in the source or load libraries. The suffix is set by the `PSBSF`, `DBDSF` and `BUFSF` parameters. See the section, *ADL Parameter Module* in the *ADL Installation* documentation for details.

Step 3: Generating the Internal Control Block Table (DAZBUF)

The buffer table `DAZBUF` is arranged in groups of slots of the same size. Figure 2b below depicts the `DAZBUF` layout.

The slot size and number of slots in one group are defined with the ADL-supplied macro `BUFMAC`. The following keyword arguments are required when calling the `BUFMAC` macro:

TYPE=	ENTRY	to specify the entry for one group of slots.
	FINAL	triggers the generation of the table. A <code>BUFMAC</code> macro with <code>TYPE=FINAL</code> must be present as the last statement before the <code>END</code> statement.
SIZE=	<i>nnn</i> or <i>nnn K</i>	the size of the slots in bytes or kilobytes (K).
NUM=	<i>nnn</i>	the number of slots in the group.

You cannot specify two groups of the same size. The total size of `DAZBUF` is limited to 512K; the `BUFMAC` macro will issue an error message if this size is exceeded.

Usually, you do not have to modify the source to create DAZBUF; you assemble and link-edit the output produced by the DAZSHINE utility. However, in certain cases it might be appropriate to “tune” the DAZBUF table layout to gain performance when rolling in or rolling out the PSBs. Refer to the corresponding section below.

You may use the JCL (JCS) in the member ADLCTG3 (ADLCTG3.J) as an example of how to assemble and link-edit the DAZBUF table under z/OS (z/VSE).

The default name for the buffer table module is DAZBUF, however, a two character suffix may be appended if needed to keep more than one version in the load library. See the section *ADL Parameter Module* in the *ADL Installation* documentation for details on the `BUFSF` parameter. The full name (for example, DAZBUF) followed by the two-character suffix must be included in the `ADLCSD` member used as input for the `DFHCSDUP` utility.

DAZBUF Buffer Table

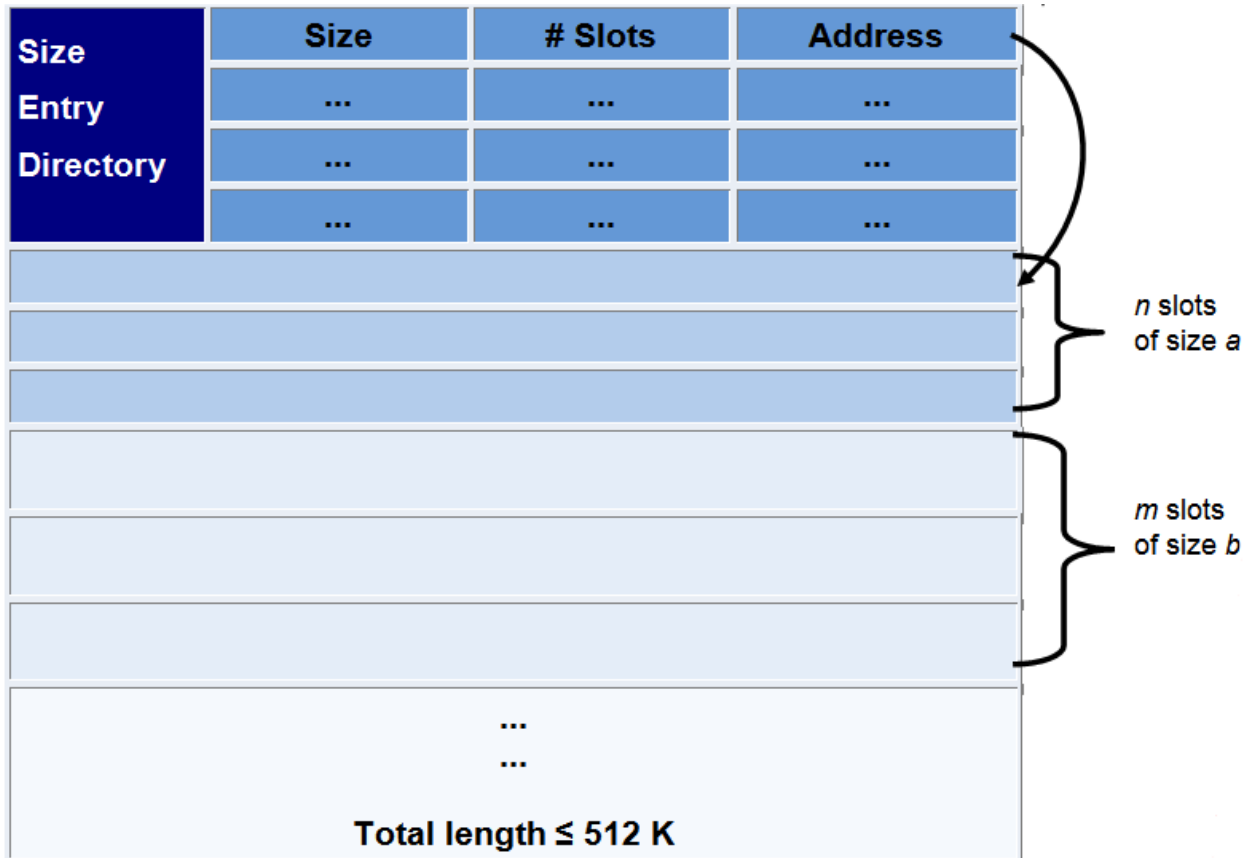


Figure 2b: DAZBUF Buffer Table

Step 4: Generating the DBD Table (DAZDBD)

The DAZDBD table is generated by the assembly and link-edit of a single call to the ADL-supplied macro DBDMAC, with the following keyword arguments:

NUMDBD=	the number of physical and secondary index DBDs referenced by all PSBs in DAZPSB. Default: 10
NUMSEG=	the total number of segments in the DBDs as included in NUMDBD, above. Default: 50
NUMFLD=	the total number of fields in the DBDs as included in NUMDBD, above. Default: 100
NUMSEC=	the total number of secondary indices specified by all physical DBDs included in NUMDBD, above. Default: 20
NUMEXR=	the total number of secondary indices for which an index maintenance exit routine is supplied. Default: 0
NUMAFI=	the total number of ADL files accessible by the ADL Consistency Interface. Default: NUMDBD
NUMAFD=	the total number of field definitions in all ADL files above. Default: 9 x NUMAFI + 2 x NUMSEG + NUMSEC + 2 x NUMFLD

The summary report of the DAZSHINE utility (Step 2 above) will provide you with these values, in the case where you do not use the source generated by DAZSHINE.

Note that you do not have to specify precise parameter values, but they must always be large enough to cover the total number of entries. You should, therefore, include those DBDs that are not yet converted but will be in the future. This may prevent having to re-create DAZDBD every time another database is converted.

In addition to tracking DBD internal control blocks, the DAZDBD module maintains a table of task entries currently active under ADL. The DBDMAC macro parameter `TSKENT` specifies the maximum number of tasks that can be active under ADL at the same time. `TSKENT` defaults to 255, which should normally be sufficient.

Also, there is an area which contains the local user blocks (LUBs). The LUBs are used at the start and end of a task which is running against ADL. The `LUBENT` parameter of the DBDMAC macro specifies the number of LUB entries. The default value of `LUBENT` is 10, which should normally be sufficient.

The other tables contained in the DAZDBD module are the Exit Routine Table, the Segment Description Table (SDT) and the File Description Table (FDT). The Exit Routine Table is used for the administration of the User-supplied Index Maintenance Exit Routines. See the appendix in the *ADL Installation* documentation for more information on exit routines. With the help of the other two tables, the ADL Consistency Interface assigns an Adabas field to a SENSEG of the internal PSB in order to generate an internal DL/I call.

You may use the JCL (JCS) in the member ADLCTG4 (ADLCTG4.J) as an example on how to assemble and link-edit the DAZDBD table under z/OS (z/VSE).

The default name for the DBD table module is DAZDBD; however, a two-character suffix may be appended if needed to keep more than one version in the load library. See the section *ADL Parameter Module* in the *ADL Installation* documentation for details of the `DBDSF` parameter. The full name followed by the two-character suffix must be included in the ADLCSD member used as input for the DFHCSDUP utility.

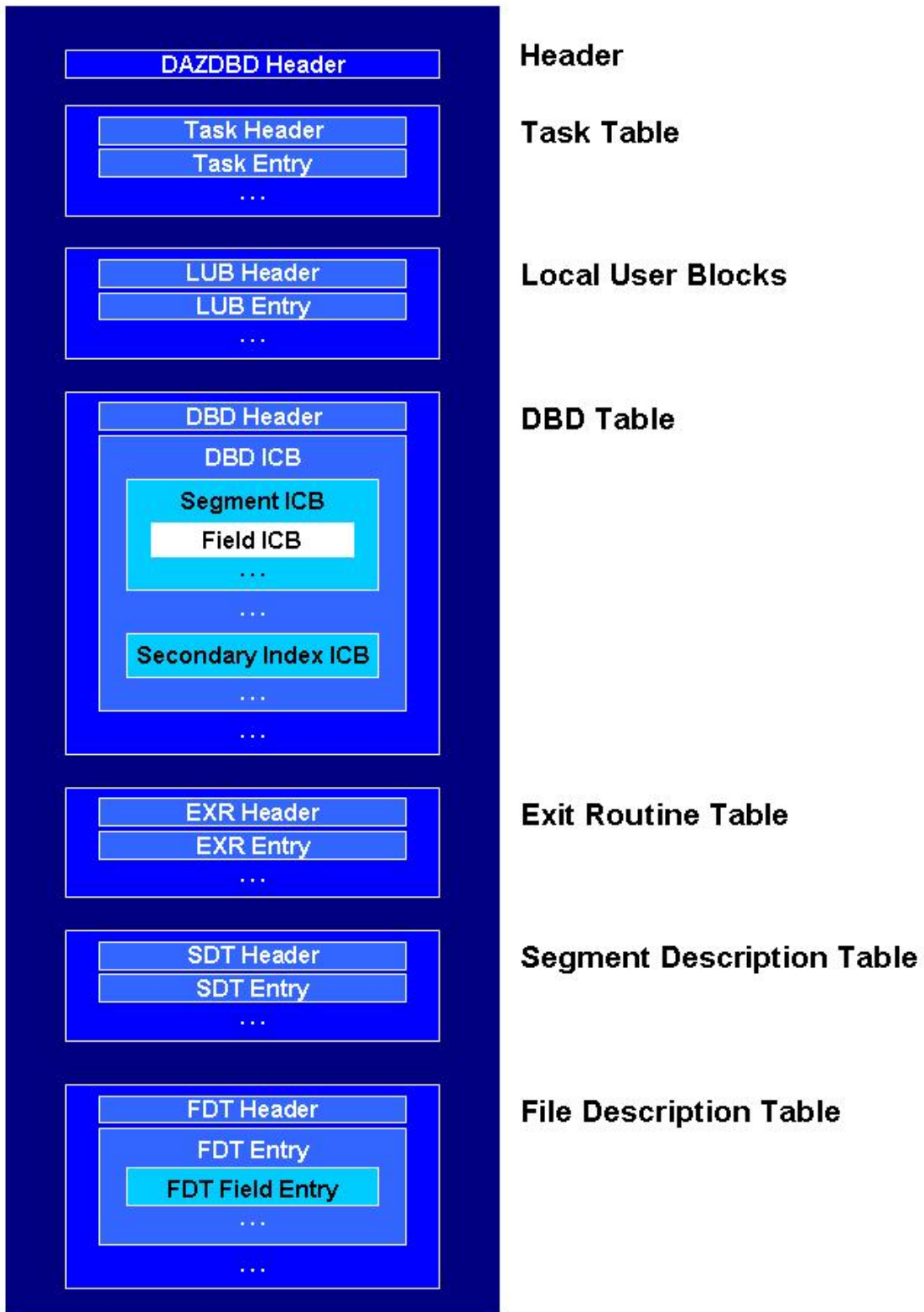


Figure 2c: DAZDBD Table

Tuning and Maintaining the Runtime Control Tables

When ADL is active under CICS, it continuously monitors the performance of the buffer table DAZBUF and the PSB usage. When switched off, ADL prints a summary of the buffer performance.

This CICS Monitor Performance Summary report, which is written to the extra partition dataset destination DAZP, can be printed using either the DAZPRINT utility or any standard printing utility. The DAZPRINT utility is activated by executing the ADL initialization program DAZIFP with the parameters shown below:

```
PRT,DAZPRINT  
MODE=ROUTINE
```

The CICS Monitor Performance Summary tells you, how often a group of slots was used to roll in a PSB from the directory file ("PSB Rolled In" count) and how frequently a task had to put in wait status because a slot could not immediately be freed ("Task Wait" count). If the Task Wait count is rather high compared to other groups of slots and to the Roll In count, the number of slots in this particular slot group should be increased.

The Performance Summary also lists all PSBs by slot groups. This list shows the actual size of the PSB internal control blocks, how frequently they have been used (Use Count) and how frequently they have been rolled out (Roll Out Count). A Use Count of zero indicates that the particular PSB was never scheduled during the CICS session, so you should check if such a PSB can be omitted from the list of PSBs in the DAZPSB Table.

The performance of the buffer management can be improved by changing the number of slots for the particular groups, by changing the group's slot sizes, or by adding or deleting slot groups.

The requirements for the buffer management will also change during the conversion process, when more and more data bases will be converted. In particular, when additional DBDs or PSBs are converted, you must check that the DAZDBD table is still large enough to handle all DBD internal control blocks and all corresponding FDTs. It is recommended, however, to repeat the steps 2, 3 and 4 of the CICS control table generation completely.

z/OS Requirements

The following table lists the data sets used by the utility DAZSHINE.

DDname	Medium	Description
DAZIN1	Reader	Control input for the ADL batch monitor, DAZIFP, and for the DAZSHINE utility.
DAZOUT1	Printer	Messages and codes.
DAZOUT2	Report	Report.
DAZOUT4	Disk	Source for the DAZBUF and DAZDBD table generation.

Examples:

The following is an example of a job to run the DAZSHINE utility:

```
//          EXEC PGM=DAZIFP,PARM='SHI,DAZSHINE'
//STEPLIB  DD DISP=SHR,DSN=ADL.LOAD
//          DD DISP=SHR,DSN=ADABAS.LOAD
//DDCARD   DD *
ADARUN PROGRAM=USER,...
//DAZIN1   DD *
MODE=mode,RANGE=(start,step)
//DAZOUT1  DD SYSOUT=X
//DAZOUT2  DD SYSOUT=X
//DAZOUT4  DD DSN=&&DECK,DISP=(,PASS),UNIT=SYSDA,
//          SPACE=(80,(100,100),RLSE),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//*
//          EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD SYSOUT=X
//SYSUT2   DD DSN=ADL.SOURCE,DISP=SHR
//SYSIN    DD DSN=&&DECK,DISP=(OLD,DELETE)
```

The following is an example of a job to print the CICS Monitor Summary:

```
//PRT      EXEC PGM=DAZIFP,PARM='PRT,DAZPRINT'
//STEPLIB  DD DISP=SHR,DSN=ADL.LOAD
//DAZIN5   DD DISP=SHR,DSN=cics-file-dazout2
//DAZOUT1  DD SYSOUT=X
//DAZIN1   DD *
MODE=ROUTINE
/*
```

z/VSE Requirements

The following table lists the data sets used by the utility DAZSHINE.

DTF	Logical Unit	Medium	Description
DAZIN1	SYSIPT	Reader	Control input for the ADL batch monitor, DAZIFP, and the DAZSHINE utility.
DAZOUT1	SYSLST	Printer	Report, messages and codes.
DAZOUT2	SYS011	Printer	Report. *
DAZOT3D	SYS013	Disk	Report. **
DAZIN3D	SYS013	Disk	Report. **
DAZOUT4	SYSxxx	Disk	Source for the generation of the DAZBUF table.

* Only required when more than one logical printer is available. In this case, SYS011 may be used to assign a second printer to which the report will be routed directly.

** Only required when only one logical printer is available. In this case, the report which is normally directed to DAZOUT2 as the second print file will be directed to disk. At the end of the job it will be read from disk and routed to DAZOUT1.

The control input for the batch monitor (DAZIFP), for ADARUN, and for DAZSHINE itself are all read from SYSIPT. The control statements for this must be specified in the following order:

```
SHI,DAZSHINE                               Input for DAZIFP
/*
ADARUN DB=dbid,MODE=MULTI,PROGRAM=USER, ... Input for ADARUN
/*
MODE=mode,RANGE=(start,step)              Input for DAZSHINE
/*
.
. ↵
```

Examples:

The following is an example of a job to execute the DAZSHINE utility:

```
// ASSGN SYS010,DISK,VOL=volser,SHR
// DLBL DAZOUT4,'punchfile',0,SD
// EXTENT SYS010,volser, ...
// ASSGN SYS013,DISK,VOL=volser,SHR
// DLBL DAZOT3D,'printfile',0,SD
// EXTENT SYS013,volser, ...
// DLBL DAZIN3D,'printfile',0,SD
// EXTENT SYS013,volser, ...
// EXEC DAZIFP
```

```
SHI,DAZSHINE                               Input for DAZIFP
/*
ADARUN PROGRAM=USER, ...                    Input for ADARUN
/*
MODE=mode,RANGE=(start,step)               Input for DAZSHINE
/*
// DLBL IJSYSIN,'punchfile'
// EXTENT SYSIPT,volser ASSGN SYSIPT,DISK,VOL=volser,SHR
// EXEC LIBR,PARM='ACCESS S=data_set_name'
/ &
CLOSE SYSIPT,READER
```

The following is an example of a job to print the CICS Monitor Summary:

```
// JOB
// ASSGN SYS014,DISK,VOL=xxxxxx,SHR
// DLBL DAZIN5D,'data_set_name',0,SD
// EXTENT SYS014,xxxxxx
// EXEC PROC=ADLLIBS
// EXEC DAZIFP
PRT,DAZPRINT
/*
MODE=ROUTINE
/*
/ &
```

Activating and Controlling the ADL Interfaces

Under CICS, the ADL Interfaces may be activated in the following three ways:

- Automatic activation
- ADL Online Services
- ADL transactions

Automatic activation and deactivation of ADL may be achieved by adding entries to the CICS PLT for the ADL Interface modules DAZCINIT (for start-up) and DAZCEND (for shut-down). This is described earlier in this section.

The most convenient way of managing the ADL interfaces is provided by the ADL Online Services, which allow you to activate and deactivate the ADL interfaces as well as start and stop the ADL trace facility for some or all terminals. You can list the CICS PSB table and write the ADL tables on the CICS dump file. These features are described in more detail in the section *ADL Online Services*.

ADL provides transactions to activate, deactivate and control the ADL Interfaces under CICS. In contrast to the ADL Online Services (which is a Natural application), these transactions are written in Assembler, so they can be used in environments where Natural is not available. Each transaction

corresponds to one function as described below. After the requested function is performed, the current status of ADL is displayed on the screen and control is returned to CICS.

The following functions are available:

Transaction	Program	Function
DAZI	DAZCINIT	Switch on the ADL Interfaces
DAZE	DAZCEND	Switch off the ADL Interfaces
DAZT	DAZCTON	Start the ADL trace facility for all terminals
DAZO	DAZCTOFF	Stop the ADL trace facility
DAZD	DAZCDUMP	Write the ADL tables to the CICS dump files
DAZS	DAZCINF	Display the current status of the ADL Interfaces

In addition to the programs outlined above, the program DAZCFCT is delivered as source code. It can be used to switch on the ADL trace facility for a specific terminal. The parameters of DAZCFCT are given as assembler variables and their purpose is explained in comments in the source code. A sample JCL/JCS to assemble and link-edit the program is provided in the source library member ADLCFCT (ADLCFCT.J).

When the ADL Interfaces are activated, the following sequence of informational messages is sent to the operator console:

```
ADL0931 - Adabas Bridge for DL/I - Nucleus loaded
ADL0936 - ADL table DAZPSB loaded
ADL0937 - ADL table DAZDBD loaded
ADL0935 - ADL table DAZBUF loaded
ADL0933 - ADL Consistence Interface is active
ADL0938 - ADL task related user exit is active
ADL0939 - Adabas Bridge for DL/I initialised
```

When the ADL Interfaces are activated manually, the message ADL0939 is also displayed at the terminal.

If the initialization of ADL fails, or the ADL Interfaces are in INDOUBT status, you should carefully evaluate the cause of this problem. After correcting the error, you can activate the ADL Interfaces with the DAZI transaction. It is recommended, however, to restart your CICS system after a failure of the ADL Interfaces initialization in order to ensure a defined state of the system.

At the start of each transaction the INDOUBT is indicated. This will be overwritten at the end of the transaction by the final status.

```

*** Adabas BRIDGE for DL/I 2.3.1 ***

CALLDLI Interface ..... On
Consistence Interface .. On
Routine Trace ..... Off
Call Trace ..... Off
Trace Term ID .....
DL/I in System ..... Yes
CICS Level ..... 0640

ADL0939 - Adabas Bridge for DL/I initialised

```

Figure 2d: The ADL transaction DAZI

CALLDLI Interface

Normal Mode and Mixed Mode

As is the case with batch processing, the ADL CALLDLI Interface may be used with CICS in both normal mode and mixed mode. In contrast to batch processing, however, there is no difference between execution of CICS transactions in the two modes. The decision whether to pass a DL/I call to ADL or to DL/I is made completely automatically, and is based on the status of the PSB and DBD being used. The status of the PSB is displayed with the CICS PSB table in the ADL Online Services. The various possibilities are as follows:

PSB Status	Description
ADL	The PSB and all related DBDs are converted; and the PSB is included in DAZPSB. Each call will be passed on to ADL (Adabas).
Mixed	The PSB is converted and included in DAZPSB. At least one referenced DBD is converted and one is not converted. The ADL CALLDLI Interface will check the DBD accessed. If the DBD has been converted to Adabas, the call will be passed on to the ADL CALLDLI Interface: if not, it will be passed on to DL/I.
DL/I	The PSB is not converted or not included in DAZPSB or all related DBDs are not converted. Each DL/I call will be passed on to DL/I.

Note that all DBDs related to a PSB which has the status DL/I or Mixed must be properly defined for DL/I and that the corresponding databases must be opened, regardless of the status of the DBD (converted or not). ADL will pass the scheduling call for each of the PSBs to DL/I, which will return a scheduling error code (one of "TA", "TE", "TJ", "TH", "TK" or another) to the application.

Also, ADL will pass all calls referring to PSBs which are not contained in the ADL table of PSBs, DAZPSB to DL/I.

Link-Editing of Application Programs

In general, application programs do not have to be re-linked. ADL provides a language interface with the ADL load libraries:

DAZLICI3	for CICS under z/OS.
DAZLICID	for z/VSE CICS.

These modules provide the entry points CEETDLI, ASMTDLI, PLITDLI, CBLTDLI, RPGDLI and FORTDLI. You may need these interfaces to replace the IBM language interfaces DFSLI000 (z/OS) and DLZLI000 (z/VSE) in case of DL/I not being available in your system.

For detailed information on how to link-edit a CICS application, see the *CICS Installation and Operation Guide* for z/OS or z/VSE.



Note: Application programs which have been linked with DAZLICI2 of ADL 2.2 or earlier cannot run with ADL 2.3. Therefore these applications must be re-linked with the new DAZLICI3. DAZLICI3 is CICS release independent so that a further relinking will be no more required.

De-synchronization of CICS Applications

Programs which issue DL/I and native Adabas calls (e.g. Natural for DL/I programs), will encounter “synchronization” problems after the DL/I data has been converted to Adabas and is accessed through the ADL. This is because from the Adabas point of view, all calls are issued by one user. The open call issued by the program is overwritten by the open call of ADL, similar problems arise for ET, BT, RC and CL commands.

ADL offers a parameter “ADAUSR”. If you set this parameter to “YES”, the ADL CALLDLI Interface generates its own Adabas user ID which avoids the mismatching with the Adabas direct commands issued by the program. The Adabas user ID is only generated when the ADL Consistency interface is active. Adabas calls against the ADL Consistency run under the same user ID as the native Adabas calls.

If you do not have programs issuing DL/I and Adabas calls simultaneously, you can set “ADAUSR=NO” which is also the default.

Consistency Interface

This section describes how the ADL Consistency Interface intercepts Adabas calls in a CICS environment and the actions necessary to activate the Consistency Interface.


The internal PSB with the name ADL\$PSB is automatically generated by the ADL Consistency Interface. It is built up dynamically based on the contents of the ADL Directory file whenever ADL is switched on in a CICS system. With the CONSI parameter of the GENDBD function in the ADL Control Block Conversion Utility, you can define which DBDs should be used to build up the internal PSB (refer to the *ADL Conversion* documentation for details). The same functionality is provided by the **DBDs for Consistency** function in the ADL Online Services (see the corresponding section in this documentation).

The purpose of the consistency PSB, ADL\$PSB, is to allow the Consistency Interface to generate internal DL/I calls to serve data base requests from Natural or Adabas applications.

Installing the Consistency Interface

The ADL Consistency Interface for CICS intercepts Adabas calls in an Adabas user exit named ADLEXITB.

To operate with the Consistency Interface, add a statement for ADLEXITB to the linkage editor input for the Adabas link module under CICS, as described in the *ADL Installation* documentation.

 **Important:** When the ADL Consistency is used with Adabas version 8.1, an Adabas correction must be applied. This is CI812002 for Adabas 8.1.2 or CI813001 for Adabas 8.1.3.

Customizing the Consistency Interface

When ADL is “switched off” in a CICS system, each Adabas call will be rejected by the ADLEXITB user exit with a response 216. However, if you link a table of converted Adabas files (DAZTCF) to the Adabas link module under CICS, a call will be rejected only if it refers to a file which is in this table. The method for generating a table of converted Adabas files is described in *Consistency Interface* in the section *Batch Installation and Operation*.

The contents of the DAZTCF table should be evaluated very carefully. Do not include the ADL Directory file, Adabas and Natural system files etc., but include ADL files. Once ADL is switched on, the DAZTCF table is no longer used. Instead, a similar table is generated dynamically based on the information in the ADL Directory file.

Activating the Consistency Interface

Finally, for the Consistency Interface to become active under CICS, ADL has to be “switched on” in that particular CICS system. See the section [Activating and Controlling the ADL Interfaces](#).

Once ADL is switched on, the status of the internal Consistency PSB 'ADL\$PSB' indicates the status of the Consistency Interface.

PSB	Status Description
ADL	The Consistency Interface is active and the Consistency PSB is successfully initialized.
Mixed	The Consistency Interface is active but the Consistency PSB is found to be empty.
DL/I	The Consistency Interface is not active.

Parameters for the Consistency Interface

This section describes which parameters are of importance for the Consistency Interface and how you may initialize them. The meaning of the individual parameters, their possible values and their default values are described in the *ADL Installation* documentation in the section *ADL Parameter Module*.

All parameters are initialized by the assembly and link-edit of the ADL parameter module. Parameters of particular importance for the Consistency Interface in a CICS environment are:

FSTAC	the size (in bytes) of the internal format buffer stack
RBSIZ	the size (in bytes) of the internal record buffer

Summary

To activate the ADL Consistency Interface under CICS, you have to perform the following steps:

- assemble the table DAZTCF of converted Adabas files (optional);
- include ADLEXITB in the linkage editor input statements of the Adabas link or globals module;
- include DAZTCF in the linkage editor input statements of the Adabas link or globals module (optional);
- ensure that ADL is “switched on” in the CICS system.

User Exit

ADL Interfaces optionally pass control to a user written routine (user exit) before each call to Adabas. This user exit may be used for monitoring purposes as well as to apply modifications to the call parameters before Adabas receives control. The conventions for this user exit and how it is activated is described in section *Adabas Call User Exit* in the *ADL Installation* documentation.

4 IMS/TP Installation and Operation

- Overview 48
- Generating the Runtime Control Tables 48
- CALLDLI Interface 49
- ADL Pre-load Program 51
- User Exit 51
- JCL Requirements 51

This chapter covers the following topics:

Overview

For application programs accessing ADL to be able to run under IMS/TP, the batch-specific steps of the installation procedure must have been performed.

Each application program affected must be re-linked with an ADL front-end program, DAZENTRY. This front-end program then passes control to ADL for PSB initialization every time the application program is entered. After this, control is given to the application program.

Each DL/I call issued by the application program is passed on to the ADL CALLDLI Interface, which then determines whether it must be passed on to IMS, handled by ADL itself, or both. Calls using a DB PCB are handled as if the application program were running in mixed mode. All calls against converted data bases are handled by ADL; all other calls are passed on to IMS. Calls using an I/O PCB are generally passed on to IMS, except for the GU and CHKP calls using the first I/O PCB. These are treated first of all by ADL as a checkpoint call (i.e. an Adabas ET call is issued), and are then passed on to IMS.

If all DB PCBs of a PSB refer to converted databases, the DB PCBs can be deleted from the IMS PSB (but not from the ADL PSB). This avoids warning messages from IMS/TP at start-up of the message region, if the corresponding VSAM file is no longer available.

Generating the Runtime Control Tables

To operate the ADL Interfaces under IMS/TP you need to generate three runtime control tables:

DAZPSB	a table containing an entry for each PSB to be used in the IMS/TP system. A PSB, which does not contain DB PCBs (i.e., it contains TP-PCBs only) should not be included in the table.
DAZDBD	a table which is used to store DBD internal control blocks.
DAZBUF	a table used as a buffer for PSB related internal control blocks which are subject to roll in/roll out mechanisms.

These tables are exactly the same as those needed to operate the ADL Interfaces under CICS. For brevity, the generation of these tables is described only once, in the section [CICS Installation and Operation](#).

As is the case under CICS, the ADL CALLDLI Interface will only route those data base requests to ADL which reference a PSB contained in the DAZPSB table.

A summary report on the performance of these tables in a particular message region will be written to the file DAZOUT2 when the message region is terminated.

CALLDLI Interface

IMS/TP Message Region Execution

To invoke an IMS/TP Message Region under ADL, execute the DAZIFP initialization program. This program requires input (positional and keyword) parameters similar to those for the IMS Message Region Controller. These are explained below.

The JCL describing the original IMS data bases which have been converted is not required.

The ADL load library containing the executable ADL batch module and the Adabas load library must be included in the JCL. ADARUN control statements must also be provided, as is the case with any Adabas application program. For a detailed description of JCL requirements, see the end of this section.

Pre-requirements for IMS/TP Application Programs

Each application program running under a Message Region Controller started with ADL must be re-linked with DAZENTRY using the following link edit directives:

```
ENTRY DAZENTRY
CHANGE pgment(DLITASM)
INCLUDE APPL(pgmname)
INCLUDE ADLLOAD(DAZENTRY)
NAME pgmname(R)
```

where

pgmname	is the name of the application program and
pgment	is the name of the entry part of the application program.

Only main programs, i.e., programs which are initially called by a transaction, should be linked with DAZENTRY, but not subprograms, etc..

Parameters for the ADL Message Region Control Program

All IMS parameters (i.e. those parameters specified in the EXEC statement) will be passed on unchanged to IMS. The first positional parameter is also interpreted by DAZIFP and should be specified as follows:

MSG

This indicates an IMS/TP Message Region run.

In addition to the positional parameter mentioned above, you may specify one or more keyword parameters in order to control operation of ADL. Such keyword parameters must be specified in a separate control statement with the following format:

keyword ,

The following table provides a brief explanation of the various keywords which can be specified. For further information on all the parameters (including default settings), see the section *ADL Parameter Module* in the *ADL Installation* documentation.

Keyword	Description
BUFSF	A two-character suffix for the name of the buffer table module.
DBD	The size (in kilobytes) of the ADL DBD ICB buffer.
DBDSF	A two-character suffix for the name of the DBD table module.
DBID	The Adabas data base ID for the ADL directory file.
EBUF	The size (in kilobytes) of the ADL ECB buffer.
FNR	The Adabas file number of the ADL directory file.
IMSY	IMS/TP syncpoint/Adabas ET synchronization.
PLI	Passes parameters on to PL/I. See the section <i>Batch Installation and Operation</i> in this documentation for details.
PSB	The size (in kilobytes) of the ADL PSB ICB buffer.
PSBSF	A two-character suffix for the name of the PSB table module used under CICS.
STACK	The size (in kilobytes) of the ADL internal subroutine stack.
TRACE	Activates the TRACE facility and specifies what is to be traced.

ADL Pre-load Program

Add an entry for DAZMPL to the IMS/TP pre-load list DFSMPLxx. The module DAZMPL is delivered in the ADL load library. It is linked as re-usable and not re-entrant. This ensures that IMS/TP will reload it after a failure.

User Exit

ADL Interfaces optionally pass control to a user written routine (user exit) before each call to Adabas. This user exit may be used for monitoring purposes as well as to apply modifications to the call parameters before Adabas receives control. The conventions for this user exit and how it is activated is described in the section *Adabas Call User Exit* in the *ADL Installation* documentation.

JCL Requirements

The following table lists the data sets used by the ADL batch monitor when an IMS/TP Message Processing Region is run.

DDname	Medium	Description
DAZIN2	Reader	Control input for the ADL batch monitor, DAZIFP, and the keyword parameters.
DAZOUT1	Printer	Messages and codes.
DAZOUT2	Printer	DAZIFP initialization and termination report.

Example

```
//REGION EXEC PGM=DAZIFP,PARM=(MSG,.....)
//STEPLIB DD DSN=IMSVS.PGMLIB,DISP=SHR
// DD DSN=IMSVS.RESLIB,DISP=SHR
// DD DSN=ADL.LOAD,DISP=SHR
// DD DSN=ADABAS.LOAD,DISP=SHR
//DDCARD DD *
ADARUN PROGRAM=USER,...
//DAZIN2 DD *
DBID=9,FNR=30
//DAZOUT1 DD SYSOUT=X
//DAZOUT2 DD SYSOUT=X
```


5 ADL Online Services

■ Introduction	54
■ Main Menu	56
■ Maintaining the ADL Interfaces under CICS	58
■ ADL Directory Management Facility	63
■ Consistency DBD Maintenance	74
■ Maintenance of the Rolled-out PSBs	75
■ Maintenance of Checkpoints	80
■ Messages and Codes Retrieval	84

This chapter covers the following topics:

Introduction

The ADL Online Services provides services which are used:

- to maintain the ADL Interfaces under CICS,
- to report the contents of the ADL directory file, in particular the DBD and PSB control blocks,
- to assign DBDs for the ADL Consistency Interface,
- to handle the rolled-out PSBs for CICS or IMS/TP,
- to maintain the checkpoints stored in the directory,
- to retrieve ADL messages and codes.

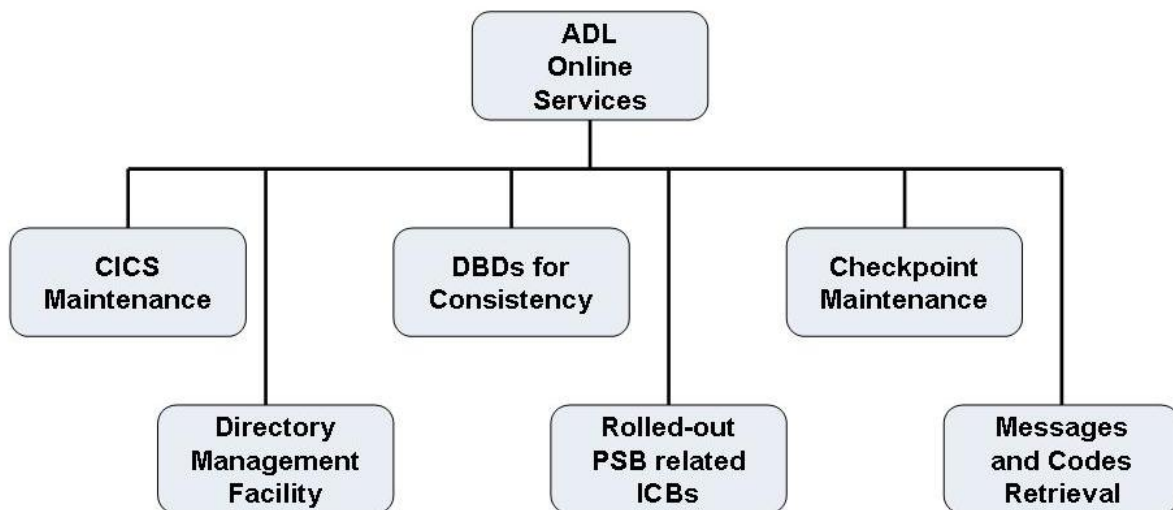


Figure 1: ADL Online Service Main Functions

Each of these services is selectable from the Main Menu of the ADL Online Services, and the following sections describe them in detail. Note that the service “CICS Maintenance” is only of interest if you are running ADL in a CICS environment, and the service “Rolled-out PSB related ICBs” is only of interest if you are running in a CICS environment or under IMS/TP. The service “DBDs for Consistency” is only of interest if Natural or Adabas direct call programs are modifying the migrated data.

Prerequisites

The ADL Online Services is an online application written in Natural. Therefore it can only be accessed if Natural is available on your site. Furthermore, the service “Maintenance of the ADL Interfaces under CICS” can only be used if the ADL Online Services are installed in the same CICS as the ADL Interfaces. The main functions under CICS, however, can also be performed from Assembler applications. Refer to the section *CICS Installation and Operation* in this documentation for more information on the ADL functions under CICS.

In order to access the ADL Directory file, the ADL Online Services must be aware of the DBID and the file number of the ADL Directory file. You have to specify the following Natural parameter:

```
LFILE=(207,dbid,fnr)
```

where *dbid* and *fnr* are the database ID and file number of the ADL Directory file. The `LFILE` parameter is described in detail in the *Natural Parameter Reference* documentation. If the `LFILE` parameter is not defined, The ADL Online Services will display the following message at startup:

```
Warning: No Natural LFILE parameter for ADL directory!
```

In this case, you must specify the DBID and file number of the ADL Directory file in the Main Menu, as described later.

Starting the ADL Online Services

The programs of the ADL Online Services are stored on the SYSADL library. To start the application, enter the following statements:

```
LOGON SYSADL MENU
```

Terminating the ADL Online Services

To terminate the session, press the PF3, PF12 or CLEAR key in the Main Menu, or select the code “.” in the Main Menu.

Online Help

For each menu of the ADL Online Services there is a help facility available. Press the PF1 key to access the help information corresponding to the current screen.

General Key Assignments

If not otherwise stated, the following general key assignments are valid for all of the screens in the ADL Online Services:

Key	Value	Meaning
PF1	Help	Display online help for the current screen.
PF3	Exit	Save modifications, if any, and return to previous screen.
PF7	Prev	Scroll to previous page.
PF8	Next	Scroll to next page.
PF10	Top	Scroll to first page.
PF11	Bot	Scroll to last page.
PF12	Can	As PF3, but modifications are not saved.
PF15	Menu	Save modifications, if any, and return to the Main Menu.
Clear		As PF15, but modifications are not saved.

General Map Elements

Every screen of the ADL Online Services contains the following elements:

- Current date and time.
- ADL Online Services version.
- DBID and file number of the ADL directory.
- Name of the active function.
- Name of current map.

Note that in the CICS Maintenance the ADL Online Services version and DBID/FNR displayed are those of the ADL nucleus rather than those of the Natural environment currently used.

Main Menu

When you start the ADL Online Services, the following menu is displayed:


```

10:55:01          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802          - Main Menu -                          MENU---M

```

```

Code  Function          ADL Directory
-----
A     DBDs for Consistency (ADABAS)  DBID: 1424
C     CICS Maintenance             FNR : 802
D     Directory Management Facility  -----
M     Messages and Codes Retrieval
R     Rolled-out PSB-related ICBs
V     ADL Version and Correction Status
X     Checkpoint Maintenance
?     Help
.     Terminate Session
-----

```

Code:

```

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help           Exit                               Can

```

In the Main Menu you can select which of the ADL online service you want to use. The functions provided are described below in details. Additionally you can specify the DBID and file number of the ADL directory. The DBID/FNR specified will be used in all services; solely the CICS Maintenance uses the DBID and file number of the ADL CICS nucleus.

The Main Menu provides the following functions:

Code	Function
A	DBDs for the ADL Consistency Interface. You can list and modify the status of all converted physical databases.
C	CICS Maintenance. The current status of the ADL Interfaces and of the ADL trace facility is outlined together with some system information. You can start and stop the ADL Interfaces, switch the ADL trace facility on and off, list the ADL PSB table for CICS, list the ADL zap status and write the ADL tables to the CICS dump file.
D	Directory Management Facility. You can list the DBDs and PSBs of the ADL Directory file. The segment and field definitions are outlined together with their corresponding Adabas definitions. There is also a function which supplies you with a detailed list of information of each external control block.
M	Messages and Codes Retrieval. You can display the full ADL error message text corresponding to ADL error numbers. Also, the ADL abend codes and the DL/I status codes can be retrieved.
R	Rolled-out PSB-related ICBs. You can list the PSBs and corresponding environments, and delete unused PSBs.

Code	Function
X	Checkpoint Maintenance. You can list and delete checkpoints of programs which terminated abnormally.
?	Online Help.
.	Terminate the ADL Online Services.

To terminate the session, press either PF3 or PF12 or the Clear key.

Maintaining the ADL Interfaces under CICS

This service is reached by selecting code C in the Main Menu, and allows you to maintain the ADL Interfaces under CICS.

```

10:57:37          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
1424 / 802          - CICS Maintenance -                      ADLCICSM

CALLDLI Interface ..... Off                               Terminal ID ... TCG7
Consistency Interface .. Off                               Routine Trace .. Off
DL/I in System .....                               Call Trace .... Off
CICS Level ..... 0640                                     Trace Term ID ..

                Functions :

                _ Switch ADL Interfaces on
                _ Switch Trace Facility on
                _ List CICS PSB Table
                _ List ADL Zap Status
                _ Dump ADL Tables

Mark the requested function(s).

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit ADL  Trace PSBs                      Zaps          Can
    
```

The screen for this service displays the DBID and file number of the ADL Directory file as defined in the parameter module of the ADL CICS nucleus. The version of the ADL CICS nucleus is also displayed. These values may therefore differ from those displayed in other screens of the ADL Online Services.

The screen informs you about the status of the ADL Interfaces. The ADL CALLDLI Interface can be in any one of the following states:

Status	Description
Off	The ADL CALLDLI Interface is not active. All DL/I requests are routed to DL/I (if available).
On	The ADL CALLDLI Interface is active. All DL/I requests are routed to ADL, which decides whether they can be handled by Adabas or whether they have to be forwarded to DL/I (mixed mode only).
Indoubt	The ADL Interfaces are in an undefined state. The initialization could not be completed successfully. Do not attempt to run your system when it is in this state. Deactivate the ADL interfaces and determine the cause of the problem. When the problem has been solved, you can start the ADL Interfaces again. You are recommended, however, to restart your CICS system after a failure of the ADL initialization.

The ADL Consistency Interface can be in one of the following states:

Status	Description
Off	The ADL Consistency Interface is either not installed or not activated. If it is not installed, all Adabas requests are routed to Adabas. Otherwise, depending on the entries in the DAZTCF table, the call is either routed to Adabas or is refused with response 216.
On	The ADL Consistency Interface is installed and active. All Adabas requests are routed to ADL, which then decides if the requests can be forwarded directly to Adabas, or if they have to be processed by ADL first.

Furthermore, the CICS Maintenance menu outlines the terminal ID, the CICS level, and whether DL/I is in the system. The latter information can only be retrieved if the ADL CALLDLI Interface is active. Finally the status of the ADL trace facility is displayed. It shows if the ADL internal routines and/or the DL/I and Adabas calls will be traced and for which terminal(s).

The functions provided by the CICS Maintenance menu are described now in detail. If you mark more than one function, they are processed one after the other.

Switch ADL Interfaces on / off

When you mark this function in the CICS Maintenance menu, the ADL CALLDLI and Consistency Interfaces are switched on or off, depending on their current status. When switching ADL off, you will be asked to confirm the requested function in a pop-up window.

The ADL Consistency Interface will only be activated if it is installed. This function will also reset all internal tables used by ADL to handle the PSB scheduling. In addition a summary will be printed on the performance of the ADL buffer management. See the section [Generating the Runtime Control Tables](#) in the for more details on how to use this report.

When the ADL CALLDLI Interface is successfully activated the following message is displayed:

ADL0939 - Adabas Bridge for DL/I initialized

When the ADL Interfaces are deactivated, the following message is displayed:

ADL0940 - Adabas Bridge for DL/I switched off

Switch Trace Facility on / off

When you select this function in the CICS Maintenance menu the following pop-up window is displayed

```

11:10:15          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
1424 / 802          - CICS Maintenance -                      ADLCICSM

CALLDLI Interface ..... On          Terminal ID .... TCG7
Consistency Interface .. On          Routine Trace .. Off
DL/I in System ..... No           Call Trace ..... Off
CICS Level ..... 0640              Trace Term ID ..

          Functions :

          _ Switch A+-----+
          x Switch T| Switch on ADL Routine Trace . Y      |
          _ List CIC| Switch on ADL Call Trace .... Y      |
          _ List ADL| Terminal ID(s)..... TCG7             |
          _ Dump ADL|                                     |
                   | Enter to perform, PF3 to exit        |
                   +-----+

Mark the requested function(s).

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit ADL  Trace PSBs              Zaps              Can
    
```

This Trace facility provides the same trace possibilities as the batch Trace facility described in the section [Debugging Aids - ADL Trace Facility](#) later in this documentation.

You can specify which kind of trace should be activated and for which terminal(s). The default terminal ID is the current one. A blank character is treated as a wildcard. For example to switch on the ADL Trace facility for all terminals where the ID starts with an 'A', specify 'A ' as terminal ID. If the terminal ID is completely blank, the ADL Trace facility is started for all terminals. Note, that you can only activate the Trace facility successfully, if the trace datasets have been installed.

List CICS PSB Table

If you select this function in the CICS Maintenance menu, the following screen is displayed.

```

11:19:15          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
1424 / 802          - CICS PSB Table -          ADLCPSBM

Nr.      Name      Status  Lan #PCB #IO-PCB DBID Schedule Total use  Rolled out
-----
 29  PSB026  DL/I   -   -   -   -   -   -   -
 30  PSB027  DL/I   -   -   -   -   -   -   -
 31  PSB028  DL/I   -   -   -   -   -   -   -
 32  PSB029  DL/I   -   -   -   -   -   -   -
 33  PSB030  DL/I   -   -   -   -   -   -   -
 34  SCHOOL  ADL   CBL  10   0  1424   0   0   0
 35  SCHOOLL ADL   CBL   1   0  1424   0   0   0
 36  TSTPSB  ADL   CBL   6   0  1424   0   0   0
 37  UNKNOWN DL/I   -   -   -   -   -   -   -

Start:                                                    Total : 37

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help           Exit                               Prev Next           Top Bot  Can

```

It lists all the PSBs contained in the DAZPSB table. The status indicates how the PSB was initialized. Possible status values are: 'ADL', 'DL/I' and 'Mixed'. Their meaning is described under [CALLDLI Interface](#) in the section *CICS Installation and Operation*. The following information can only be retrieved if the PSB status is 'ADL' or 'Mixed':

Column	Description
Lan	The language of the corresponding application program(s) as defined in the PSB source.
#PCB	The total number of PCBs in the PSB.
#IO-PCB	The number of IO-PCBs in the PSB.
DBID	The Adabas DBID of the ADL files belonging to the PSB.
Schedule	The number of tasks currently scheduling the PSB.
Total use	The number of tasks which have used the PSB up to now.
Rolled out	The number of times the PSB-related ICBs have been rolled out to the ADL directory file.

You can use the PF-keys for scrolling or you can specify a start value to restart the list from that value. Note that pressing ENTER refreshes the screen.

List ADL Zap Status

If you select this function in the CICS Maintenance menu, the following screen is displayed:

```

11:23:47          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
1424 / 802        - List ADL Zap Status -                    ADLCZAPM

      Range                Applied Zaps
-----
001 - 010    001  002  ...  ...  ...  006  ...  ...  ...  ...
011 - 020    ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
021 - 030    ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
031 - 040    ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
041 - 050    ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
051 - 060    ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
061 - 070    ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
071 - 080    ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
081 - 090    ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
091 - 100    ...  ...  ...  ...  ...  ...  ...  ...  ...  ...

Start: 1

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit                Prev  Next                Top  Bot  Can
    
```

It lists all zaps applied to the ADL CICS nucleus DAZNUCC. A zap which is applied, is indicated by its number. Possible zap numbers are 1 through 500. You can use the PF-keys for scrolling or you can specify a start value to restart the list from that value.

Dump ADL Tables

When you select this function in the CICS Maintenance Menu, the following pop-up window is displayed:

```

11:28:11          *** ADL ONLINE SERVICES 2.3.1 ***          19.06.2007
1424 / 802          - CICS Maintenance -                    ADLCICSM

CALLDLI Interface ..... On          Terminal ID .... TCG7
Consistency Interface .. On          Routine Trace .. Off
DL/I in System ..... No            Call Trace .... Off
CICS Level ..... 0640              Trace Term ID ..

          Functions :

          _ Switch ADL Interfaces off
          _ Switch Trace Facility on
          _ List CICS PSB Table
          _ L +-----+
x D | Writing ADL tables to the dump file |
   | Do you wish to continue? Y/N ..... Y |
   | ENTER to perform, PF3 to exit       |
   +-----+

Mark the requested function(s).

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit ADL  Trace PSBs              Zaps              Can

```

When you enter 'Y' (yes), the ADL CICS tables DAZPSB, DAZDBD and DAZBUF are written to the CICS dump file, together with the ADL internal area DAZSYSDS and the ADL zap directory. This information is helpful when debugging abnormal terminations of application programs under CICS.

ADL Directory Management Facility

This service is reached by selecting code D in the Main Menu.

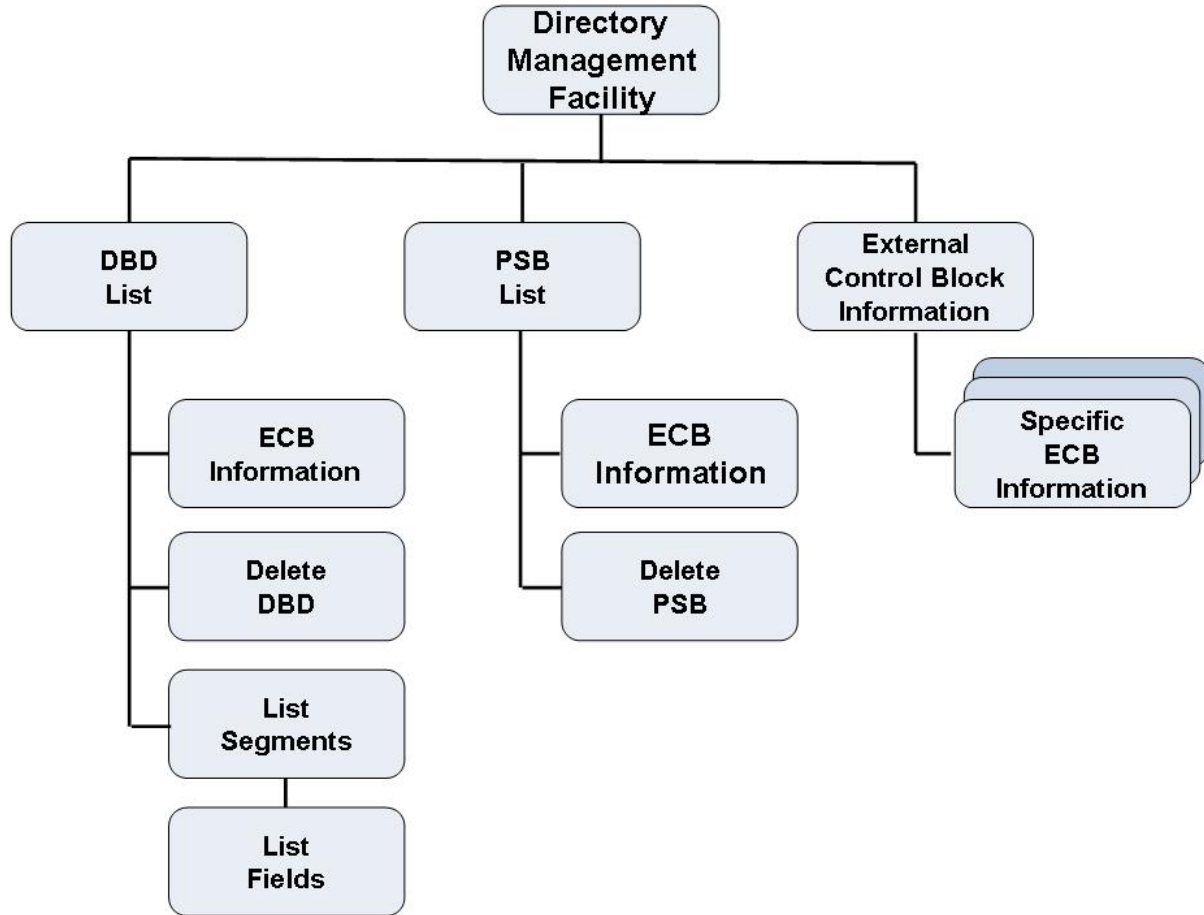


Figure 2: Directory Management Facility

The ADL Directory Management Facility (DMF) consists of following parts:

- The “Directory Management Facility” menu is the entry to DMF and all its services.
- The “DBD List” menu provides a list of all DBDs which are stored on the ADL directory file.
- The “List Segments” menu shows you the segments of a specific DBD.
- The “List Fields” menu exposes the definitions of the fields under a given segment.
- The “PSB List” menu provides a list of all PSBs which are stored on the ADL directory file.
- You can “delete DBDs” or “PSBs” from the ADL directory file.
- In the “External Control Block Information” menu you can retrieve an external control block (ECB) in hexadecimal and character format.
- The “specific ECB Information” provides you with all information of a specific ECB.

Directory Management Facility Menu

```

11:31:32          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802    - Directory Management Facility -          ADBDMF-M

          Code  Function                                     Name
          ----  -
          D    DBD List                                     0
          L    List Segments of Data Base                 R
          P    PSB List                                     0
          I    ECB Information
          ?    Help
          .    Back to the Main Menu
          ----  -

Enter Code :      Name:                                     R -> required
                                                         0 -> optional

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit                                         Can

```

This service is reached by selecting code D in the Main Menu. It is the entry to the DMF. The codes in the menu provide the following functions:

Code	Function
D	DBD List. Display all data bases, which are stored in the ADL directory file. The list starts at the specified DBD name, or if none is given, at the top.
L	List Segments. Explode the segments of the given DBD. A name of a DBD is required.
P	PSB List. Display all PSBs, which are stored in the ADL directory file. The list starts at the specified PSB name, or if none is given, at the top.
I	ECB Information. Retrieve a detailed list of information of any external control block.
?	Online Help.
.	Back to the Main Menu.

DBD List Menu

```

11:33:57          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802          - DBD List -          ADBDBD-M

      Cmd DBD      Typ      Cmd DBD      Typ      Cmd DBD      Typ
      -----      ---      -----      ---      -----      ---
      _  COURSEDB  P          _          _          _
      _  COURSEL   L          _          _          _
      _  INSTDB    P          _          _          _
      _  INSTL     L          _          _          _
      _  STUDIDX   X          _          _          _
      _
      _
      _
      _
      _

Start: _____

Available Commands: Delete  Inform  List  Quit

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help          Exit          Next          Top          Can
    
```

This service is reached by selecting code D in the Directory Management Facility menu. It lists all DBDs of the ADL directory file, together with their type. Possible types are:

Type	Meaning
P	Physical DBD
L	Logical DBD
X	Secondary Index DBD

The DBDs can be marked with the following line commands:

Line Command	Meaning
D	A pop-up window for interactive confirmation is displayed. The marked DBD will be deleted from the ADL directory file, if you confirm with 'Y' (yes). Note that the Adabas file(s) containing the data of the DBD, will not be deleted.
I	Display the ECB Information of the marked DBD.
L	List the segments of the marked DBD.
Q	Exit the DBD List menu.

If you want to delete a physical DBD, you must also delete all secondary index DBDs, which are related to this DBD. Otherwise the CBC utility will indicate an error situation in case you reconvert the DBD. A DBD can also be deleted with the DELDBD function of the ADL CBC utility which automatically deletes the related secondary index DBDs.

If you enter a new start name in the corresponding field, the DBD list will start at the given value.

List Segments Menu

```

15:24:52          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802          - List Segments -          ADBSEG-M

          Data Base: COURSEDB          Type: Physical
                                          Converted
                                          Completed
There are  4 Segments in  3 Levels with  1 Secondary Index.

  Cmd  Seg No  Segment Name  Level  ADA Grp  ADA PCK  ADA VCK  Seg No.  ADA  ADA  Log  L.Parent  L.Parent
          ADA  ADA  ADA  Len  Fld  Dbid  Fnr  Id  DB-Name  Seg-Name
-----
  _    1  COURSE      1   SA  PC      30   1  1424  833  71
  _    2  CLASS       2   SB  PB      20   1  1424  833  71
  _    3  STUDENT    3   SA              40   2  1424  835  73
  _    4  INSTRP     3   SB              25   2  1424  834  72  INSTDB  INSTRUCT
  _
  _
  _
  _

Available Commands:  List  Quit

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit              Next      Top      Can

```

This service is reached by selecting code L in the Directory Management Facility menu or by marking a DBD with code L in the DBD List menu. It informs you about a DBD and all its segments. The name and type of the DBD are displayed. Possible types are 'Physical', 'Logical', and 'Index'. It is also indicated, whether the data base is converted to Adabas and whether the conversion is completed. 'Completed' means, that all DBDs with logical relationships to the given one, are converted, too. Furthermore the number of segments, of hierarchical levels and of secondary indices is given.

The list of segments contains the following information:

Column	Meaning
Cmd	Here the user can insert a line command. The available commands are explained later.
Seg No.	The DL/I hierarchical segment number (position).
Segment Name	The name of the segment as defined in the original DL/I DBD definition.
Level	The DL/I hierarchical level. This defines together with the segment number the hierarchical structure of the DBD.
ADA Grp	The name of the Adabas group corresponding to the DL/I segment.
ADA PCK	The Adabas name of the partial concatenated key (PCK) field. For every segment with a sequence field a PCK field is created. It contains the data of the sequence field, if a dependent segment is accessed.
ADA VCK	The Adabas name of the virtual concatenated key (VCK) field. It follows the same rules as the PCK field, but is only created, if a DBD is involved in logical relationships by itself. This means, that one segment of the DBD is a logical parent of a segment of the same DBD. The VCK field contains the data of the sequence field, if a logically dependent segment is accessed.
Segm. Length	The length of the segment as defined in the original DL/I DBD definition. This is also the total length of all Adabas fields, which belong to the corresponding Adabas group.
No. Flds	The number of DL/I fields belonging to this segment. Be aware that this is often not the number of Adabas fields of the corresponding Adabas group. In fact ADL creates Adabas fields not only for every DL/I field, but also for every part of a DL/I segment for which no DL/I field is defined. This are the so-called "filler" fields. On the other hand for a redefined DL/I field no Adabas field may be created.
ADA Dbid/Fnr	The Adabas DBID and file number of the ADL file with the data of this segment. If the DBID is 0, the DBID of the ADL directory is used at runtime.
Log Id	The logical ID used for the unique identification of the hierarchical structured data.
L. Parent DB/Seg-Name	If the segment is a logical child of another segment, the DL/I data base and segment name of this "logical parent" segment are exposed here.

Note that for segments in a logical DBD only the segment number, name and level are exposed.

Possible line commands are:

Line Command	Meaning
L	List the field definitions of the marked segment.
Q	Exit the List Segments menu.

List Fields Menu

```

11:38:57          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802          - List Fields -          ADBFIE-M

          Data Base : COURSEDB  Type ..: Physical
          Segment ..: CLASS    Length : 20

          DL/I
Name      Type  Format Start Length   ADABAS      Comments
          Type  Format
-----
CLASSNO   SEQ,U  C      1      5      AC          A      UNIQUE FIELD

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
          Help      Exit          Next          Top          Can

```

This service is reached by marking a segment with code L in the List Segments menu. It informs you about the fields of a segment. The name and type of the DBD is shown together with the name and length of the segment.

The list of fields is divided in three parts:

Part	Description
DL/I	The definitions of the DL/I field are displayed here: Name, format, start and length. For more information about these values see the <i>IMS/VS Utilities Reference</i> documentation.
Adabas	The definitions of the Adabas field are shown here: Name, type and format. The Adabas field length is the same as the corresponding DL/I field length. For more information about these values see the <i>Adabas Utilities</i> documentation .
Comments	A comment can be displayed here.

PSB List Menu

```

11:40:32          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802          - PSB List -          ADBPSB-M

      Cmd PSB          Cmd PSB          Cmd PSB
      ----          - - - - -          - - - - -
      _ SCHOOL          -              -
      -              -              -
      -              -              -
      -              -              -
      -              -              -
      -              -              -
      -              -              -
      -              -              -
      -              -              -
      -              -              -
      -              -              -
      -              -              -
      -              -              -
      -              -              -

Start: _____

Available Commands:  Delete  Inform  Quit

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help          Exit              Next          Top          Can

```

This service is reached by selecting code P in the Directory Management Facility menu. It lists all PSBs of the ADL directory file.

The PSBs can be marked with the following line commands:

Line Command	Meaning
D	A pop-up window for interactive confirmation is displayed. The marked PSB will be deleted from the ADL directory file, if you confirm with 'Y' (yes).
I	Display the ECB Information of the marked PSB.
Q	Exit the PSB List menu.

If you enter a new start name in the corresponding field, the PSB list will start at the given value. A PSB can also be deleted with the DELPSB function of the ADL CBC utility.

External Control Block Information Menu

```

11:42:48          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802    - External Control Block Information -      TESTECBM

Code .....: I          (Ecb,Info)
Type .....: DBD        (Dbd,Seg,FlD,seX,Lch,Psb,pCb,SEs,SeF,xRf)
PSB/DBD name ....: COURSEDB
PCB/DBD offset ..: 00000000      PSB/XRF offset ...: 00000000

ECB type ..... DBD          Lines ..... 2

L Offset      External Control Block (hex)          ECB (char)          DBD,PSB
-----      -
1  0000      00000000 00000460 C3D6E4D9 E2C5C4C2 *      ?-COURSEDB *      00000000
   0010      00000000 00000000 00000000 00000000 *      * *      00000010
2  0020      00000040 00000460 04010329 000003A0 *      ?-???? ?? *      00000020
   0030      00000000 00000000 00000000 00000000 *      * *      00000030
3  0040      40404040 40404040 40404040 40404040 *      * *      00000040
   0050      40404040 40404040 40404040 40404040 *      * *      00000050
4  0060      40404040 40404040 40404040 40404040 *      * *      00000060
   0070      40404040 40404040 40404040 40404040 *      * *      00000070

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help          Exit          Next          Can

```

This service is reached by selecting code I in the Directory Management Facility menu. To retrieve information of an external control block (ECB) you have to specify a code, the type of the ECB, the PSB or DBD name, and the offset to the ECB.

The “Code” defines how the information is to be returned.

Code	Meaning
E	The ECB is listed in hexadecimal and character format on the bottom of the current screen.
I	A detailed list of information of the specified ECB is displayed. When returning to the current map, the ECB is listed as with code E.

The “Type” specifies the type of the requested ECB. Possible types are : DBD, SEG (segment), FLD (field), SEX (secondary index), LCH (logical child), PSB, PCB, SES (sensitive segment), SEF (sensitive field), XRF (external reference). The shortest possible abbreviation for a type is denoted by uppercase characters, like 'D' for DBD. In general you may use a 'D' for all DBD-related ECBs (DBD, SEG, FLD, SEX, LCH, XRF) and a 'P' for all PSB-related ECBs, (PSB, PCB, SES, SEF). But if you use the 'Next' (PF8) function, you must specify the correct type.

The “PSB/DBD name” is the name of the PSB or DBD, to which the ECB belongs.

The offsets have to be specified in hexadecimal format. Which offsets you have to specify depends on the ECB type you want to retrieve.

ECB Type	PCB/DBD Offset	PSB/XRF Offset
DBD	0	0
SEG,FLD,SEX,LCH	offset to ECB	0
PSB	0	0
PCB	0	offset to PCB
SES,SEF	offset to ECB	offset to PCB
1st XRF	offset to 1st XRF	0
more XRF	offset to 1st XRF	offset to current XRF

If you want to retrieve a specific ECB and you don't know the offset to it, you have to start with the DBD or PSB ECB. For these the offsets are all zero. In the ECB Information you find the offset to the first sub-structure like SEG or PCB. Then you can continue with the sub-structures until you reach the requested ECB.

Example

You want to retrieve the ECB information of the first field 'CLASSNO' in the second segment 'CLASS' of the DBD 'COURSEDB'. Specify Code 'I', PSB/DBD name 'COURSEDB', and PSB/XRF offset '0'. Then specify the following types and PCB/DBD offsets to retrieve the information which is required for the next step.

Step	Type	DBD Offset	Retrieved ECB	Page	Line	Value
1	DBD	00000000	DBD COURSEDB	1	'First segment'	00000040
2	SEG	00000040	Segment COURSE	1	'Next segment'	00000160
3	SEG	00000160	Segment CLASS	2	'First field'	000001E0
4	FLD	000001E0	Field CLASSNO			

For each retrieved ECB, the type (which can differ from the specified type) and the number of lines used by this ECB are displayed. The list contains a line counter ('L'), the offset relative to the start of the ECB, the ECB in hexadecimal and character format, and the offset relative to the start of the corresponding DBD or PSB ECB. This last offset is the sum of the two specified offsets.

With the PF8 ('Next') key, you retrieve the next ECB of the same type. This function is not available for DBD and PSB ECBs. In the example above you can press the PF8 key in step 3 instead of specifying the DBD offset '160', to retrieve the second segment (CLASS).

Specific ECB Information

```

11:53:47          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802          - ECB Information -          ADLPCB1M

SCHOOL   + 00000520          PCB   7          Page 1 of 3

Keyword  Name          Description          Value
-----  -
          TYPEC$TY    Type of ADL External Control Block ... 18          => PCB
          TYPECNXT  Next PCB ECB (PSB offset) ..... 000005E0
DBDNAME  PCBECNAM    Name of corresponding DBD ..... COURSEDB
PROCSEQ  PCBECIND    Index DBD name .....
          PCBECNUM  PCB progressive number ..... 7
          PCBECLV#  Number of levels ..... 3
          PCBEC$SN#  Total number of sensitive segments ... 4
KEYLEN   PCBECKEY    Key feedback area length ..... 35
          PCBECSEN  First sensitive segment (PCB offset) . 00000040 => COURSE
          PCBEC$TY    Type of PCB ..... 02
TYPE     PCBEC#GM    GSAM PCB .....
POS      PCBEC#MP    Multiple positioning support ..... X

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help          Exit          Prev Next          Top Bot Can

```

This service is reached by selecting code I in the External Control Block Information menu or by marking a line with I in the DBD List or PSB List menu. It supplies you with all information of a specific external control block (ECB). In the header is the type and the name of the ECB displayed, together with the DBD/PSB name and the offset to the current ECB. The page number can be overwritten, if a specific page should be listed. Beside of the standard PF keys, you can also specify a page '0' to exit the menu. The columns in the list provide the following information:

Column Meaning

Column	Meaning
Keyword	The DL/I keyword of the DBD/PSB definition, which specifies the value. For more information about the keywords see the <i>IMS/VS Utilities Reference</i> documentation.
Name	The ADL internal used name of the value.
Description	A short description of the value.
Value	Counters and numbers are printed in decimal format, offsets and type or flag bytes in hexadecimal format. A flag is set, if the value is 'X'. Some values are coded, for example the 'Type'. In this case the corresponding read-able value is denoted by '=>'. For some offsets the corresponding ECB name is denoted in the same way.

Consistency DBD Maintenance

This service is reached by selecting code A in the Main Menu.

```

11:56:47          *** ADL ONLINE SERVICES 2.3.1 ***          19.06.2007
01424 / 00802    - DBDs for the ADL Consistency Interface -    ADBCFD-M

      Cmd DBD   Status      Cmd DBD   Status      Cmd DBD   Status
      --- ---   -
      _  COURSEDB N          _
      _  INSTDB   N          _
      _
      _
      _
      _
      _
      _
      _
      _
      _

Start: _____

Available Commands: Yes  No

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit  Yes  No  Undo      Next  Save  Top      Can

```

This menu lists all converted physical DBDs. It exposes if the corresponding Adabas files can be accessed by the ADL Consistency Interface or not by the status “Y” (yes) or “N” (no), respectively. You can change the status by entering the appropriate line commands. Save your changes with PF9.

The status is defined with the `CONSI` parameter of the `GENDBD` function at the DBD conversion. See the section *ADL Conversion Utilities for DBDs and PSBs* in the *ADL Conversion* documentation for more details. Any changes of the status should be evaluated carefully. An Adabas/ Natural call (batch or online) referencing a DBID / FNR which belongs to a DBD with status “N”, is routed directly to Adabas. Thus, if a DBD should be accessed by both DL/I and Adabas / Natural applications, it's status must be “Y”.

You may set the status of a DBD to “N”, if there is no Adabas / Natural application which updates the corresponding files.

You may also set the status to “N”, if only Adabas / Natural applications access this DBD, and you have decided not to run any other DL/I applications against it. In this case there is no need to support the ADL internal fields anymore.

Unused DBDs should have the status “N” or should be deleted from the directory file. This should be done, in order to avoid problems during the building up of the internal Consistency PSB, which picks up all converted physical DBDs with status “Y”.

If there is more than one DBD referencing the same data (i.e., the same Adabas file), only one of them should have the status “Y”. All others should be set to “N”.

Note, that the CICS or IMS/TP tables have to be re-generated (see the section [Generating the Control Tables](#) in this documentation), if you change the status of a DBD from “N” to “Y”.

Possible line commands are:

Line Command	Meaning
Y	Set the status to “Y” (yes). Calls against this DBD will be handled by the ADL Consistency Interface.
N	Set the status to “N” (no). Calls against this DBD will be routed directly to Adabas.

If you enter a new start name in the corresponding field, the list will start at the given value.

The assignment of the function keys is as follows:

Key	Function
PF4	Set the status of all DBDs on the current page to “Y”.
PF5	Set the status of all DBDs on the current page to “N”.
PF6	Undo all modifications on the current page.
PF9	Save all modifications.

If you leave the menu and there are any modifications which have not yet been saved, you will be asked whether you want to save them or not.

Maintenance of the Rolled-out PSBs

This service is reached by selecting code R in the Main Menu.

At start-up of ADL under CICS or IMS/TP all PSB related external control blocks (ECBs) are converted to internal control blocks (ICBs) and rolled out to the ADL directory file. They are stored with the name of the environment (CICS or IMS/TP start-up job name) and the PSB name. They can be deleted, if ADL is not active in the corresponding environment.

The Maintenance of the Rolled-out PSBs consists of following parts:

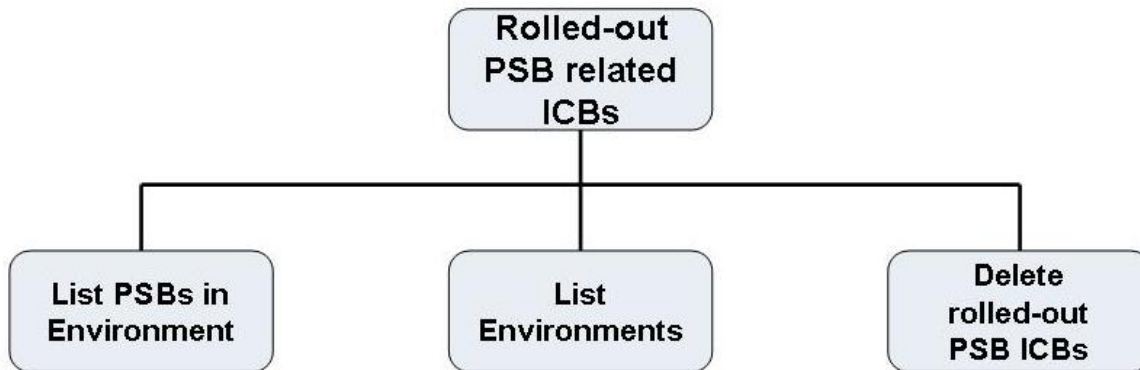


Figure 3: Rolled-out PSBs

Rolled-out PSB-related ICBs menu

This service is reached by selecting code R in the Main Menu.

```

11:58:31          *** ADL ONLINE SERVICES 2.3.1 ***          19.06.2007
01424 / 00802    - Rolled-out PSB-related ICBs -          ADLPIM-M

Code  Function
-----
P    List PSBs in Environments
E    List Environments
D    Delete Rolled-out PSB ICBs
?    Help
.    Back to the Main Menu
-----

Enter Code :

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit  PSBs  Env.                               Can
  
```

This menu is the entry to the Maintenance of the Rolled-out PSBs. The codes in the menu provide the following functions:

Code	PF-Key	Function
P	PF4	List all PSBs for all environments.
E	PF5	List all environments.
D		Delete all or specific PSBs.
?		Online Help.
.		Exit the menu.

List of the PSBs in the Environments

```
Dir 1424 / 802 - ADL 2.3.1 Rolled-out PSB-related ICBs - Page 1
-----
```

No.	Env. (Hex)	PSB (Hex)	Env.	PSB	Blocks
1	C4C1C5C6 C3C9F0F2	C1C4D35B D7E2C240	DAEFCIO2	ADL\$PSB	3
2	C4C1C5C6 C3C9F0F2	C6C5D5C4 E3D7E2C2	DAEFCIO2	BIGPSB	1
3	C4C1C5C6 C3C9F0F2	E2C3C8D6 D6D34040	DAEFCIO2	SCHOOL	3
4	C4C1C5C6 C3C9F0F2	E2C3C8D6 D6D3D340	DAEFCIO2	SCHOOLL	1
5	C4C1C5C6 C3C9F0F2	E3E2E3D7 E2C24040	DAEFCIO2	TSTPSB	3

```
Total number of rolled-out PSB-related ICBs .. : 5
Total number of environments ..... : 1
```

MORE

This service is reached by selecting code P or pressing PF4 in the Rolled-out PSB- related ICBs menu. It lists the names of the environments and of the PSBs in hexadecimal and character format. Additionally it outlines how many blocks (Adabas records) are used to store the ICBs. At the end of the list the total number of the rolled-out PSB-related ICBs and of the environments is printed.

Note, that the list is printed in sequential sequence. You can only page forward (by pressing ENTER). When all PSBs have been listed, the Rolled-out PSB-related ICBs menu is displayed.

List of the Environments

```

Dir 1424 / 802 - ADL 2.3.1 Environments of PCB-related ICBs - Page 1
-----
No.          Env. (Hex)          Env.
-----
    1      C4C1C5C6 C3C9F0F2      DAEFC102
Total number of environments ..... :          1

MORE
    
```

This service is reached by selecting code E or pressing PF5 in the Rolled-out PSB-related ICBs menu. It lists the names of the environments in hexadecimal and character format. At the end of the list the total number of the environments is printed.

Note, that the list is printed in sequential sequence. You can only page forward (by pressing ENTER). When all environments have been listed, the Rolled-out PSB-related ICBs menu is displayed.

Delete Rolled-out PSBs

This service is reached by selecting code D in the Rolled-out PSB-related ICBs menu. First a pop-up window is displayed, where you can choose, how to continue.

Enter	Meaning
Y	For each environment you will be asked if you want to delete the PSBs.
N	No PSB is deleted. Back to the Rolled-out PSB-related ICBs menu.
X	All PSBs in all environments will be deleted. There is no additional confirmation.
T	Test mode. All environments will be processed, but no PSB will be deleted.

If you have entered 'Y', 'X', or 'T' in the pop-up window, the Delete PSB-related ICBs menu is displayed.

```

13:23:16          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802      - Delete PSB-related ICBs -                ADLPID-M

  No.   Environ.   PSBs   Status
  ----  -
    1   DAEFCI02

Environments :
PSBs .....:
Records .....:

Start time ..: 13:23:16.7
Elapsed time  : 00:00:00.0

Confirm deletion of PSBs in DAEFCI02
with Y                               N

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Next  Help           Exit                               Next                               Can

```

It lists the processed environments, the number of PSBs in the environment and the status. The status can be either 'deleted' or blanks (not deleted). Additionally it informs you about the total number of environments, PSBs, and records (blocks), which have been processed up to now. The start and elapsed time of the processing is also displayed.

If you have entered 'Y' in the pop-up window, the name of an environment is displayed and you have to confirm the deletion.

Enter	
Y	All PSBs of the displayed environment will be deleted.
N	No PSB of the displayed environment will be deleted.
X	All PSBs of the displayed and of all following environments will be deleted. There is no additionally confirmation.
T	Test mode. No PSB of the displayed environment will be deleted. The following environments will be processed without confirmation, and no PSB of them will be deleted.

Note that a PSB must not be deleted, if ADL is active in the corresponding environment.

Maintenance of Checkpoints

This service is reached by selecting code X in the Main Menu.

When a batch application issues a checkpoint, ADL writes a checkpoint entry to the ADL directory file. A checkpoint entry is identified by the program name and the checkpoint ID. If the program terminates normally, all written checkpoints will be removed from the directory and will, therefore, not appear on either list mentioned below. But if the program terminates abnormally, the checkpoints will be listed and can be used to restart the program. For more information on how to restart a program, see the section [Recovery and Restart Procedures](#) in this documentation.

Checkpoint Maintenance produces two types of lists:

- Checkpoint Information List: shows each individual checkpoint ID together with the name of the abnormally terminated program,
- Checkpoint Program List: shows the total amount of checkpoints for each abnormally terminated program.

Checkpoints can also be deleted, whereby all checkpoints related to one program, are always deleted together. It is, therefore, impossible to delete individual checkpoints.



Note: Do not delete any checkpoints belonging to programs which are to be restarted.

The checkpoint maintenance feature consists of the following parts:

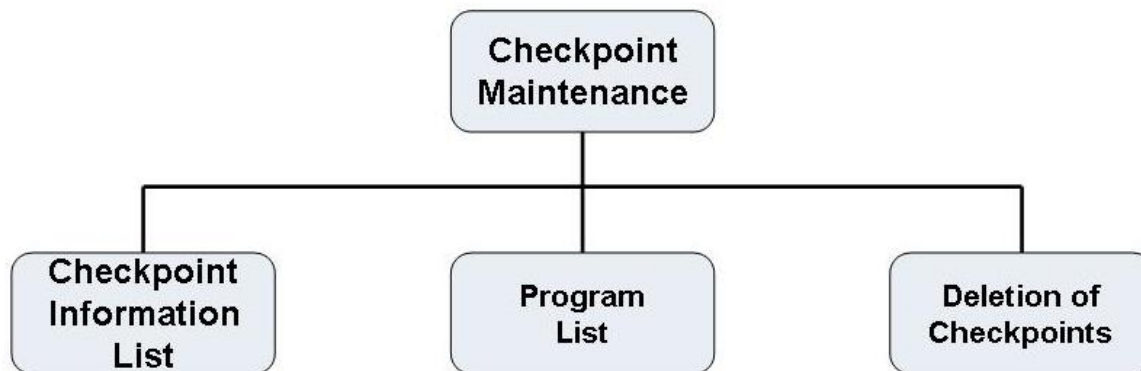


Figure 4: Checkpoint Maintenance

Checkpoint Maintenance Menu

This service is reached by selecting code X in the Main Menu.

```

13:25:12          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802      - Checkpoint Maintenance -                ADLXCM-M

                Code  Function
                -----
                L    List all Checkpoints
                P    Program List
                D    Delete Checkpoints
                ?    Help
                .    Back to the Main Menu
                -----

Enter Code :

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit  Chkp  Prog                                Can

```

This menu is the entry to the Checkpoint Maintenance. The codes in the menu provide the following functions:

Code	PF-Key	Function
L	PF4	List each checkpoint ID and program name.
P	PF5	List all abnormally terminated programs.
D		Delete checkpoints of all/specific programs.
?		Online Help.
.		Exit the menu.

Checkpoint Information List

This service is reached by selecting code L or pressing PF4 in the Checkpoint Maintenance menu. It lists the names of all abnormally terminated programs and the individual checkpoint IDs for each of those programs. Additionally, it outlines how many Adabas records are used for each checkpoint entry. At the end of the list the total number of checkpoints and of effected programs is printed.

Note, that the list is printed in sequential order. You can only page forward (by pressing ENTER). When all programs with the related checkpoint IDs have been listed, the Checkpoint Maintenance menu is displayed.

```
Dir. 1424 / 802 - ADL 2.3.1 List Checkpoint Information - Page 1
-----
No.      Program      Checkpoint      Records
-----
1        DAZZLER        GSM06X01        3

Total number of Checkpoints ..... :      1
Total number of effected programs ..... :      1

MORE
```

Checkpoint Program List

```
Dir. 1424 / 802 - ADL 2.3.1 Checkpoint Program List - Page 1
-----
No.      Program      Checkpoints
-----
1        DAZZLER        1

Total number of Checkpoints ..... :      1
Total number of effected programs ..... :      1

MORE
```

This service is reached by selecting code P or pressing PF5 in the Checkpoint Maintenance menu. It lists the names of abnormally terminated programs and the total amount of checkpoints for each program.

Note, that the list is printed in sequential order. You can only page forward (by pressing ENTER). When all applicable programs have been listed, the Checkpoint Maintenance menu is displayed.

Deletion of Checkpoints

This service is reached by selecting code D in the Checkpoint Maintenance menu. First, a pop-up window is displayed, where you can choose from the following options:

Enter	Meaning
Y	For each program you will be asked if you want to delete the checkpoints.
N	No deletion. Back to the Checkpoint Maintenance menu.
X	All checkpoints of all programs will be deleted. There is no additional confirmation.
T	Test mode. All programs will be processed, but no checkpoint will be deleted.

If you have entered 'Y', 'X', or 'T' in the pop-up window, the Delete Checkpoints menu is displayed.

```

14:08:28          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802          - Delete Checkpoints -          ADLXCD-M

  No.   Program   CPIDs   Status
  ----   -
    1   DAZZLER
                                     Programs ....:
                                     Checkpoints .:
                                     Records .....:

                                     Start time ..: 14:08:28.0
                                     Elapsed time : 00:00:00.0

                                     Confirm deletion of the checkpoints
                                     of program DAZZLER with Y N

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Next Help           Exit           Next           Can

```

It lists the program names, the number of checkpoints found in them and their status. The status can be either 'deleted' or blanks (not deleted). Additionally, it informs you about the total number

of programs, checkpoints and Adabas records, which have been processed up to now. The start and elapsed time of the processing is also displayed.

If you have entered 'Y' in the pop-up window, the name of an environment is displayed and you have to confirm the deletion.

Enter	Meaning
Y	All checkpoints of the displayed program will be deleted.
N	No checkpoint of the displayed environment will be deleted.
X	All checkpoints of the displayed and of all following programs will be deleted. There is no additionally confirmation.
T	Test mode. No checkpoint of the displayed program will be deleted. The following programs will be processed without confirmation, and checkpoints will not be deleted.

Checkpoints must not be deleted, if it is planned to restart the program.

Messages and Codes Retrieval

This service is reached by selecting code M in the Main Menu. It starts with the ADL messages retrieval.

```

14:10:30          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802    - Messages and Codes Retrieval -          ADLEMR-M

ADL message no.: 0701

Description:
Illegal or missing parameters

Cause   : The mandatory parameters for DAZIFP were not specified
          correctly. There should be three positional parameters:
          xxx,pgmname,psbname,
          These may be followed by keyword parameters.

Action  : Check the input parameters and correct the error. See the
          chapter Batch Installation and Operation, ADL Interfaces Manual
          for details.

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Read  Help Code Exit          Prev Next          Can
    
```

Enter a message number (range 1 to 1400) to retrieve the description of the ADL error message. Additionally the probably cause for the failure and the recommended action is displayed.

With the PF2 key you can switch to the ADL ABEND and DL/I status codes retrieval.

```
14:12:13          *** ADL ONLINE SERVICES  2.3.1 ***          19.06.2007
01424 / 00802      - Messages and Codes Retrieval -          ADLEMC-M
```

```
ABEND/Status code : ix
```

```
Description:
```

```
Violated insert rule
```

```
Cause .....: An insert rule has been violated.
```

```
Action .....: Check the insert rules and correct the program.
```

```
Command completed : No
```

```
Error situation ..: Yes (CMD or Conversion)
```

```
Possible commands : ISRT.
```

```
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Read Help Msg  Exit                               Prev Next                               Can
```

Enter the last two characters, to retrieve the description of the corresponding ADL ABEND code, i.e. 'Z1' for ABEND code 'DAZ1'. The ABEND codes are displayed together with the probably cause and the recommended action.

Enter a two character DL/I status code to retrieve the description of that status code. Further the following information is displayed: The cause, the recommended action, whether the command has been completed, whether it is an error situation, and for which commands this status code can be retrieved. It is also indicated, if a status code can only be issued by DL/I (i.e. not by ADL), or if it can only be issued by ADL.

Press the PF2 key to switch back to the ADL messages retrieval.

The message number 1500 (status code VR) has a special meaning. It indicates the current version of ADL and gives some information about the ADL directory file.

6 Precompiler for EXEC DLI Commands

■ Introduction	88
■ ADL Precompiler Input	91
■ ADL Precompiler Output	93
■ COBOL Generated Code	94
■ PL/I Generated Code	94
■ CICS Command Language Translator	94
■ Linkage-Editor Requirements for Application Programs	94
■ z/OS JCL Requirements	95
■ z/VSE JCS Requirements	96

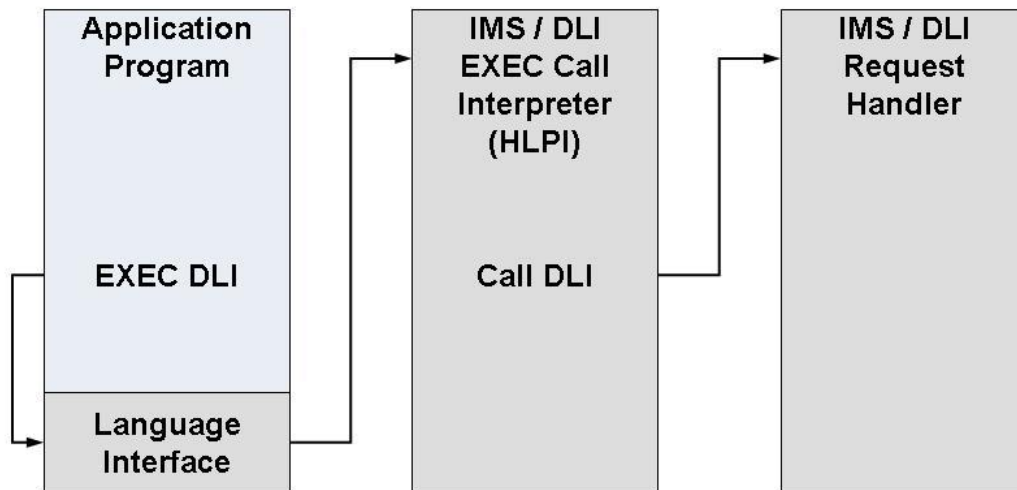
This chapter covers the following topics:

Introduction

This section is only of interest for installations in which application programs using the Higher Level Programming Interface (HLPI) need to be converted. It describes the different possibilities and procedures available for converting such application programs written in COBOL or PL/I .

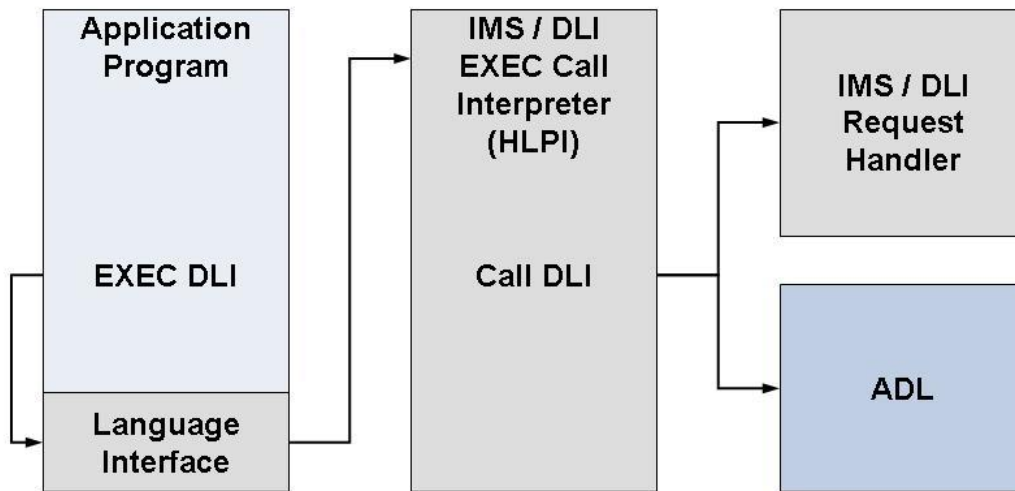
When an application program using the HLPI issues a DL/I request in an unconverted environment (i.e. where ADL is not involved), an extra interface layer is used to interpret the call from the application program (which originates from an EXEC command) and transform them into normal calls. The figure below illustrates this process.

HLPI Interface



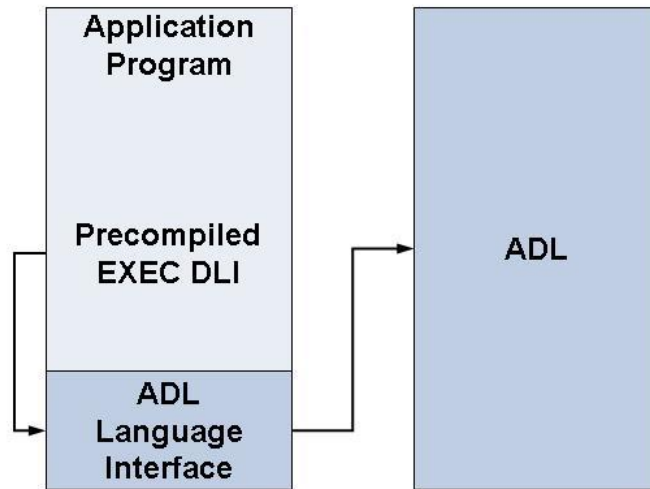
Application programs using the HLPI can be run against ADL in one of two different ways: with or without the ADL precompiler.

Running HLPI Programs without the ADL Precompiler



The application programs do not issue standard DL/I calls directly: instead, these are issued by the HLPI. This means that the DL/I environment must be present. Command level programs can be run against ADL in mixed mode.

Running HLPI Programs with the ADL Precompiler



The ADL precompiler translates each EXEC DLI command in the application program into one (and only one) call to ADL. This call is passed directly to ADL in the same manner as for normal DL/I calls. This means that no extra interface layer is required. Thus, in a completely converted environment, DL/I need not still be available. When the ADL precompiler has been used, however, application programs cannot be run in mixed mode.

ADL Precompiler Input

The ADL precompiler accepts as input a source program written in PL/I or COBOL in which EXEC DLI commands have been coded. It produces as output an equivalent source program in which the commands have been translated into statements in the language of the source program. The statements generated result in a call to the ADL nucleus (batch or CICS) to fulfill the function requested.

For more details on the syntax of the EXEC DLI commands, see the appropriate IBM documentations.

The ADL precompiler is activated separately before the compile and link edit steps.

The program source input data set must contain fixed length records, which must be 80 bytes long. The first statement of the source program must be a CBL statement for COBOL or a *PROCESS statement for PL/I. This control statement may be passed on unchanged as part of the program source output.

The control statement contains the parameters for the ADL precompiler itself, and has the following syntax:

LAN=XXX,keyword=value

Keyword	Explanation						
INPUT	(z/VSE only). Indicates whether the source program input should be read in from the reader or from disk/tape. Possible values:						
	<table border="1"> <tr> <td>DISK</td> <td>The program source input is read in from disk or tape. You must specify DAZIN5D or DAZIN5T, respectively, as DTF. The corresponding logical unit is SYS014. This default logical unit can be overwritten by the ADL 'FX' parameter as described in the section <i>ADL Parameter Module</i> in the <i>ADL Installation</i> documentation. Since the ADL precompiler expects a record size of 80 bytes, the 'FX' setting should be overwritten during the generation of the precompiler nucleus DAZNUCP. For example: FX=(14,80,80)</td> </tr> <tr> <td>READER</td> <td>The program source input is read in from the reader, i.e. from SYSIPT.</td> </tr> <tr> <td>Default:</td> <td>READER</td> </tr> </table>	DISK	The program source input is read in from disk or tape. You must specify DAZIN5D or DAZIN5T, respectively, as DTF. The corresponding logical unit is SYS014. This default logical unit can be overwritten by the ADL 'FX' parameter as described in the section <i>ADL Parameter Module</i> in the <i>ADL Installation</i> documentation. Since the ADL precompiler expects a record size of 80 bytes, the 'FX' setting should be overwritten during the generation of the precompiler nucleus DAZNUCP. For example: FX=(14,80,80)	READER	The program source input is read in from the reader, i.e. from SYSIPT.	Default:	READER
DISK	The program source input is read in from disk or tape. You must specify DAZIN5D or DAZIN5T, respectively, as DTF. The corresponding logical unit is SYS014. This default logical unit can be overwritten by the ADL 'FX' parameter as described in the section <i>ADL Parameter Module</i> in the <i>ADL Installation</i> documentation. Since the ADL precompiler expects a record size of 80 bytes, the 'FX' setting should be overwritten during the generation of the precompiler nucleus DAZNUCP. For example: FX=(14,80,80)						
READER	The program source input is read in from the reader, i.e. from SYSIPT.						
Default:	READER						
LAN	The source program language, a three-character abbreviation. Possible values: CBL for COBOL PLI for PL/I Default: none						
LIST	Indicates whether or not the program listing, including the EXEC commands with sequence numbers, is to be produced. Possible values: Y : (the program listing is to be produced) N : (the program listing is not to be produced) Default: Y						
MARGIN	(PL/I only) Specifies the left and right margin of the input statements of the source program. The ADL precompiler will only scan that part of any input source statement between the two margins. The syntax of this parameter is as follows: MARGIN=(L,R) L : Left margin of source program Possible values: 1-71 Default: 2 R : Right margin of source program Possible values: 2-72 Default: 72						
PREF	The three-character prefix to be used for variables (COBOL) and ENTRY names (PL/I). Possible values: AXX where: A : is any character which may be used as the first character of a COBOL variable name or PL/I ENTRY name. X : any character allowed to be the second or any subsequent character of a COBOL variable name or PL/I ENTRY name. Default: DAZ						
QNUM	(COBOL only). The maximum number of qualification statements to be expected in any one command. Possible values: 1-999 Default: 8						
SEGM	Indicates whether or not a program segment is to be precompiled. Possible values: Y : A program segment is to be precompiled for later inclusion in the main program. This means that only the EXEC commands will be translated. No variables and no user DIB will be generated. N : A main program is to be precompiled. Default: N If SEGM=Y is specified for a PL/I program segment, make sure that the parameter PREF is specified as well. The ADL precompiler generates a separate ENTRY statement with a unique name for each EXEC command in PL/I. Where a program segment is precompiled separately, the ENTRY names generated must differ from those in the main program. This can be achieved by specifying a different prefix (PREF parameter).						
SNUM	(COBOL only) The maximum number of SSAs to be expected in any one EXEC command. Possible values: 1-15 Default: 15						

Keyword	Explanation
UDIB	Indicates whether or not fields need to be generated for the user DL/I Interface Block (DIB). Possible values: Y : (fields for the user DIB will be generated). N : (fields for the user DIB will not be generated). Default: N
XOPTS	Specifies whether or not the first statement should be output. Possible values: Y : where the first statement should be output. N : where the first statement should not be output. Default: N

ADL Precompiler Output

The program source output is output as fixed length records with a length of 80 bytes.

The first output listing produced contains messages produced by the ADL precompiler. Where one or more errors were encountered during translation, an error message with the following layout will be produced:

```
Warning nnnn from ADL module DAZEXEC/DAZEXSER at address aaaaa
ADLnnnn message .....
Error occurred during interpretation of EXEC DLI statement with sequence no:
99999
```

where

nnnn	is the number of the message,
aaaaa	is the offset within the ADL module DAZEXEC in which the message was generated, and
99999	is the sequence number of the EXEC DLI command which was found to be in error. This sequence number corresponds to the number printed on the output data set containing the complete listing (see below).

The second output listing produced contains the original source program including the EXEC DLI commands. In addition, each EXEC DLI command has been given a sequence number which can be used to find particular commands found to be in error. Where an EXEC DLI command is found to be in error, the error message written to the message output listing is reproduced.

COBOL Generated Code

For COBOL, each EXEC command is replaced by a series of MOVE statements followed by a CALL statement. The MOVE statements take care of possible type conversions for numeric arguments and assign constants to data variables. For this purpose declarations for these temporary variables are automatically included in the working storage. Declarations for the user DIB are also automatically included in the working storage. It is possible to precompile program segments separately for later inclusion in a main program.

PL/I Generated Code

For PL/I each EXEC command is replaced by a DO statement, a declaration of a generated unique ENTRY name, a CALL statement and an END statement. The ENTRY declaration takes care of possible type conversions for numeric arguments. The ADL precompiler generates the declarations for the user DIB variables for each valid PL/I PROCEDURE statement. It is possible to precompile program segments separately for later inclusion in a main program.

CICS Command Language Translator

Where an application program also contains EXEC CICS commands, the CICS Command Language Translator has to process the application program as well before it can be compiled. The CICS Command Language Translator may be activated either before or after the ADL precompiler. In either case, make sure that the translator options for the CICS Command Language Translator do not specify DLI.

Linkage-Editor Requirements for Application Programs

After having passed the ADL precompiler, the CICS language translator and the compiler, the application program must be linked together with the appropriate language interfaces for ADL. Depending on the operational environment, the following modules are to be linked to the application program:

CICS online programs:

z/OS - DAZLIC13

z/VSE - DAZLIC1D

Batch programs:

z/OS - DAZLIBAT

z/VSE - DAZLIBAT

All language interface modules described above are on the installation tape in the ADL load library. The language interfaces mentioned above replace the IBM modules DFSLI000 (z/OS) and DLZLI000 (z/VSE). For details on how to link-edit command level application programs, see the *CICS Installation and Operations Guide*



Important: All application programs which have been linked with DAZLICI2 of ADL 2.2 or before, must be relinked with DAZLICI3. DAZLICI3 is CICS release independent.

z/OS JCL Requirements

The table below lists the data sets used by the ADL precompiler during processing of an application program.

DDname	Medium	Description
DAZIN2	Disk/Tape	Program source input
DAZIN1	Reader	ADL precompiler control statement
DAZOUT1	Printer	Program listing
DAZOUT2	Printer	Error messages
DAZOUT4	Disk/Tape	Precompiled program

Example

The following is an example of an ADL Precompiler Run for a COBOL application program:

```
//PRE EXEC PGM=DAZIFP,PARM='PRE,DAZEXPRE'
//STEPLIB DD DISP=SHR,DSN=ADL.LOAD
// DD DISP=SHR,DSN=ADABAS.LOAD
//DAZOUT1 DD SYSOUT=X
//DAZOUT2 DD SYSOUT=X
//DAZOUT4 DD DSN=&&TEMP,SPACE=(TRK,(20,20)),UNIT=VIO,DISP=(,PASS),
// DCB=(RECFM=FB,DSORG=PS,BLKSIZE=3120,LRECL=80)
//DDCARD DD *
ADARUN PROGRAM=USER,...
//*
//DAZIN1 DD *
LAN=CBL
//DAZIN2 DD *
...
... application program source code
...
//*
//APPLPROG EXEC DFHEITCL
//TRN.SYSIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
//LKED.ADL DD DISP=SHR,DSN=ADLxxx.LOAD
//LKED.SYSIN DD *
INCLUDE ADL(DAZLIC13)
NAME anyname(R)
/*
```

z/VSE JCS Requirements

The table below lists the data sets used by the ADL precompiler during application program processing.

DTF	Logical Unit	Medium	Description
DAZIN2	SYSIPT	Reader	Program source input *1
DAZIN5D	SYS014	Disk	Program source input *2/ *3
DAZIN5T	SYS014	Tape	Program source input *2/ *3
DAZIN1	SYSIPT	Reader	ADL precompiler control statement
DAZOUT1	SYSLST	Printer	Program listing
DAZOUT2	SYS011	Printer	Error messages
DAZOT3D	SYS013	Disk	Error messages *4
DAZOT3T	SYS013	Tape	Error messages *4
DAZOUT4	SYSxxx	Disk	Precompiled program

*1 Only required, if the default input mode INPUT=READER is active.

*2 Only required, if the input mode INPUT=DISK is specified.

*3 Either one (disk or tape) is required. The logical unit indicated, is the default logical unit. To change it, specify the `FX` parameter as described in the section *ADL Parameter Module* in the *ADL Installation* documentation. Note that the ADL precompiler expects a record length of 80 bytes.

*4 Only required when only one logical printer is available. In this case, the message which is normally directed to `DAZOUT2` as a second print file will be written to disk. At the end of the job, it will be read from disk and routed to `DAZOUT1`.

If the default input mode `INPUT=READER` is active, the control input for the batch monitor (`DAZIFP`), `ADARUN`, the ADL precompiler, and the program source input are all read in from `SYSIPT`. The control statements must be specified in the following order:

```
PRE,DAZEXPRE          input for DAZIFP
/*
ADARUN ...           input for ADARUN
/*
LAN= ...             input for the ADL precompiler
/*
.                   application program source
.
.
/*
```

Example

The following is an example of an ADL Precompiler Run for a COBOL application program:

```

// ASSGN SYS010,DISK,VOL=volume,SHR
// DLBL DAZOUT4,'punch-dataset',0,SD
// EXTENT SYS010,volume,,rtrk,ntrks
// ASSGN SYS013,DISK,VOL=volume,SHR
// DLBL DAZOT3D,'temp-dataset',0,SD
// EXTENT SYS013,volume,,rtrk,ntrks
// ASSGN SYS013,DISK,VOL=volume,SHR
// DLBL DAZIN3D,'temp-dataset',0,SD
// EXEC PROC=ADLLIBS
// EXEC DAZIFP,SIZE=512K
PRE,DAZEXPRES
/*
ADARUN PROGRAM=USER,...
/*
LAN=CBL
/*
  COBOL application program source
  ....
/*
// DLBL IJSYSPH,'cobol-translation',0
// EXTENT SYSPCH,,1,0,rtrk,ntrks
ASSGN SYSPCH,DISK,VOL=volume,SHR
// DLBL IJSYSIN,'punch-dataset',0
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=volume,SHR
// EXEC DFHECP1$
CLOSE SYSIPT,SYSRDR
CLOSE SYSPCH,uu
// DLBL IJSYSIN,'cobol-translation',0
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=volume,SHR
// OPTION CATAL
  PHASE pgmname,*
  INCLUDE DFHECI
// EXEC FCOBOL
  INCLUDE DAZLICID
/*
CLOSE SYSIPT,SYSRDR
// EXEC LNKEDT
/&
// JOB RESET
ASSGN SYSIPT,SYSRDR
ASSGN SYSPCH,uu
/&

```

7 Using ADL Files with Natural/Adabas

▪ Introduction	100
▪ Consistency Interface	100
▪ Restrictions when using Natural/Adabas	101
▪ Improve the Natural Access to Migrated Files	103
▪ Error Situations and Consistency Response Codes	104
▪ Availability of the Consistency Interface	105
▪ Example Programs	105

This chapter covers the following topics:

Introduction

After a DL/I data base was converted to one or more ADL files, these files can be accessed and manipulated with Natural applications or other programs using direct Adabas calls. This section explains:

- how you can do this,
- how ADL guarantees the integrity of the data as expected by original DL/I applications,
- which restrictions apply for updates from Natural/Adabas applications.
- how to improve the access to the migrated file for Natural applications.



Note: Set the Natural parameter `ADAMODE` to “0” when you run Natural programs against the ADL Consistency.

Consistency Interface

When hierarchical structures, like those being defined in DL/I data bases, are mirrored into a table like structure as those of ADL files, the original structure of the data has to be preserved by either the use of physical pointers or foreign keys. The latter may be considered as logical pointers. ADL maintains both kinds of pointers automatically.

The foreign keys maintained by ADL for a given segment type are built up in a similar way as a DL/I concatenated key. In other words, the foreign key stored together with a segment occurrence determines its position in the data base by identifying its parent segments (if there are any). ADL introduces the term “partial concatenated key” (PCK) for the foreign key of *one* particular segment type.

The physical pointers are needed for DL/I applications. Whenever a DL/I application issues a data base call, this call is intercepted by the ADL CALLDLI Interface. Implicitly, a DL/I data base call is serviced by referring to the physical pointers. We say implicitly, because these physical pointers are determined by the actual position of the segment occurrences in the data base. Based on this, the ADL CALLDLI Interface then maintains the foreign keys automatically, though they are never used directly by DL/I applications.

In case of Natural/Adabas applications, the situation is just the other way around. An Adabas call on ADL files supplies the segment data and all PCKs of its parent segments. In order for DL/I applications to be able to “see” these data as well, the corresponding physical pointers have to be built up. This task is performed by the ADL Consistency Interface.

The ADL Consistency Interface intercepts Adabas calls. On the first level, all kind of retrieval calls (like Adabas L3, S1 etc.) are routed directly to Adabas. On a second level, only calls referring to ADL files will be considered. If necessary, the Consistency automatically constructs the physical pointers for the record to be inserted, updated or deleted based on the PCKs provided by the user.

In a later stage of the data base conversion process, you might want to phase out the old DL/I applications completely. Therefore, the further operation of the ADL Consistency Interface may become obsolete. For this reason, the Natural applications or programs using direct Adabas calls should originally be designed as if the Consistency Interface is not existent.

Note that the installation and operation of the Consistency Interface is described earlier in this documentation.

Summary

- the CALLDLI Interface automatically sets and maintains the PCKs of all parent segments,
- the Consistency Interface sets and maintains the physical pointers needed to preserve the hierarchical structure of the data base,
- the Consistency Interface is necessary to update ADL files with Natural or programs using direct Adabas calls, when access to these files for DL/I applications must still be provided.

Restrictions when using Natural/Adabas

Natural applications or programs using Adabas direct calls follow exactly the same syntax in accessing either a normal Adabas file or an ADL file. The only difference is, that the applications may receive response codes from the ADL Consistency Interface, if it is active.

The rules having to be obeyed when manipulating ADL files with Natural/Adabas programs are derived directly from the fact that the hierarchical structure of the data, as seen from DL/I applications, has to be preserved. The ADL Consistency Interface is a product which guarantees the referential integrity of your data.

The following is an example of the hierarchical structure of a DL/I data base consisting of four segments types:

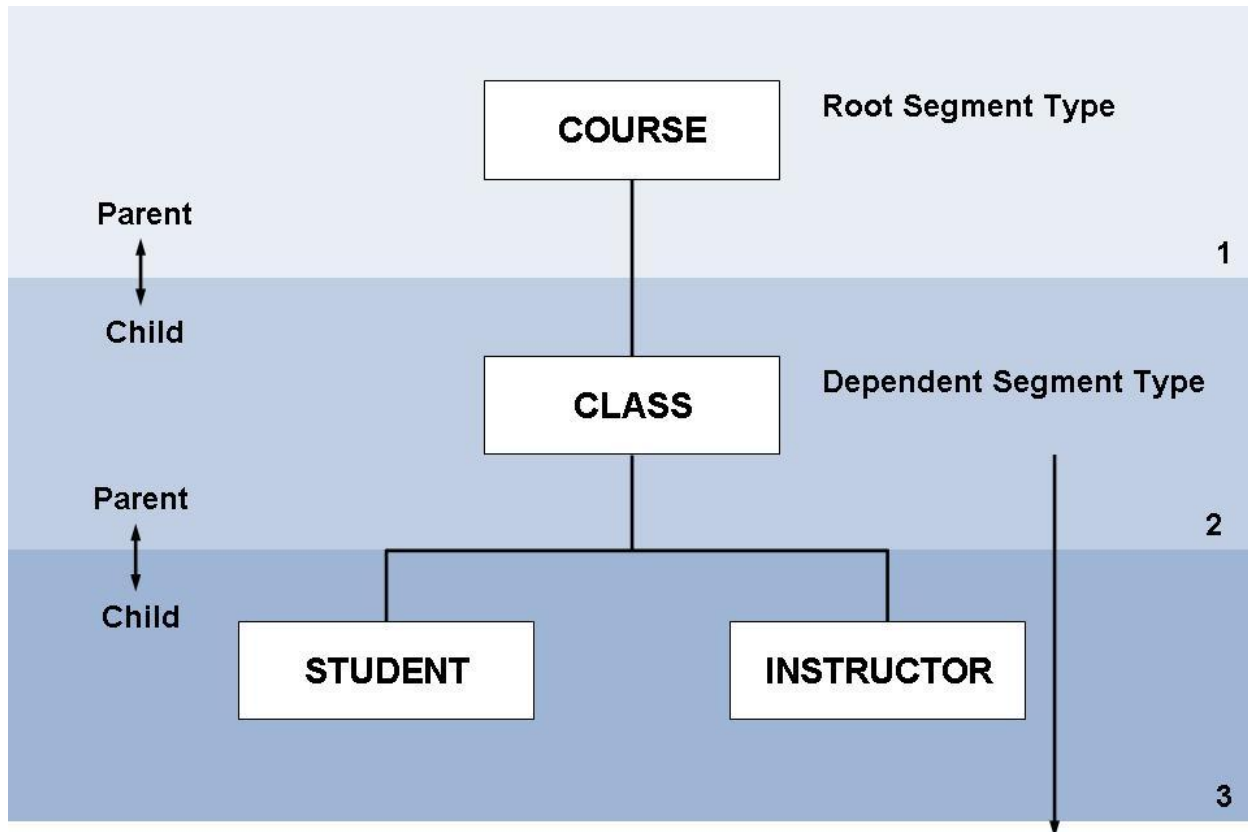


Figure: Sample data base used in our examples

In DL/I, the segments **STUDENT** and **INSTRUCTOR** are child segments to the parent segment **CLASS**, which in turn is a child segment to **COURSE**. The parent segments are identified by a course number or class number respectively.

Transformed to an ADL file the course number is denoted the PCK of the record **COURSE** and serves as a link for the records **COURSE** and **CLASS**. ADL stores the course number together with the **CLASS** data as shown in the following table:

DL/I Segment Type	ADL Record Type
COURSE	COURSE
CLASS	PCK-COURSE, CLASS
STUDENT	PCK-COURSE, PCK-CLASS, STUDENT
INSTRUCTOR	PCK-COURSE, PCK-CLASS, INSTRUCTOR

Referential integrity in this context means that each **STUDENT/INSTRUCTOR** record occurrence must refer to an existing **CLASS** record occurrence, which in turn must also refer to an existing **COURSE** record occurrence.

In order to allow the Consistency Interface to maintain the referential integrity, relation data have to be supplied with each update request to an ADL file. For insert and replace calls (Adabas N1 or A1) this simply means that you have to supply the foreign keys of all related record types. In DL/I calls this means supplying the concatenated key. Furthermore, when replacing a record, you must not alter the foreign keys of its parent segment.

Delete calls may not attempt to delete parent segment occurrences which have dependent child segment occurrences. For ADL files this means, that record occurrences without PCKs referred to by other record occurrences must be deleted first. Considering the above example, note that a record occurrence of CLASS can only be deleted after all record occurrences of STUDENT and INSTRUCTOR have been deleted.

Summary

- when inserting records in an ADL file, specify the corresponding foreign keys or PCKs for the related record type;
- when replacing records in an ADL file, specify the corresponding foreign keys or PCKs for the related record type but NEVER alter the PCKs;
- when deleting a record occurrence, be sure that it is not referred to by PCK fields in other record occurrences;
- the Consistency Interface will preserve the referential integrity of the data base.

Improve the Natural Access to Migrated Files

When ADL migrates the DL/I structure, the generated Adabas file layout is optimized to satisfy the needs of the DL/I applications. Therefore only the root sequence field, the physical pointer fields ("Z1-field") and the secondary index fields are defined as descriptors. From these fields only the root sequence field should be used by Natural applications because the other fields are ADL internal fields.

Natural applications read the hierarchy through the PCK fields. If the application should access the data of a dependent segment, it is recommended to create an Adabas superdescriptor build up by the PCK fields of the parent segments. If the segment itself has a sequence field, the corresponding Adabas field should be added to the superdescriptor as well.

If Natural applications should access data in a secondary index like manner, any other field (like a sequence field of a dependent segment) can be defined as descriptor. If you build up new superdescriptors take care that only those fields are included which belong to the same segment plus PCK fields of parents of the segment. If you use fields from different segments, one part will always be empty and the superdescriptor will not give you any value.

It is recommended that you define only those descriptors and superdescriptors which are really used by any application. Every descriptor cost some performance not only in the Natural applica-

tions but also in the DL/I applications and when utilities run. On the other hand if you have use for a descriptor or superdescriptor, you should define it. A non-descriptor search gives you the poorest performance.

Error Situations and Consistency Response Codes

Whenever you violate the rules for manipulating ADL files defined earlier in this section, the ADL Consistency Interface will return a non-zero response code in the Adabas control block. For Natural applications, this response code is available in the system variable *ERROR-NR. The Adabas response code returned is "216" and the corresponding Natural *ERROR-NR is "3216".

A more detailed error message can then be obtained from the ADL Consistency Interface by submitting an Adabas S1 call conforming to a defined standard. This is in detail explained in the documentation *ADL Messages and Codes*. Natural programmers may simply obtain this error message by a call to the ADL supplied Natural subprogram ADLERROR, as in the example below:

```
IF *ERROR-NR = 3216
  CALLNAT 'ADLERROR' #ERRMESS
END-IF
```

providing an alphanumeric variable "#ERRMESS" of at least 80 bytes length.

The error text returned has the following layout:

```
'ADLxxxx - error message .... '
```

where xxxx is a four character error code. For an explanation of the error code, please refer to the documentation *ADL Messages and Codes*.

As explained above, the ADL Consistency Interface will guarantee the referential integrity of your ADL files. However, we recommend a programming style, which avoids updates resulting in ADL Consistency Interface response codes. In other words, the logic to preserve the referential integrity of the data should as far as possible be included in the application program itself.

Example

Consider our example data base: Before placing a delete call for a record occurrence of CLASS, place a read call to make sure that no record occurrences of STUDENT/ INSTRUCTOR still exist.

The reason for our recommendation is, that in a later stage of the data base conversion process the ADL Consistency Interface could become obsolete. Then, the logic of the application programs might be affected by not receiving any Consistency response codes.

Availability of the Consistency Interface

Natural programs may check the availability of the ADL Consistency Interface by calling the ADL supplied Natural subprogram ADLACTIV. It returns a 2-bytes integer value. If the ADL Consistency Interface is active the value '0', otherwise the value '8' is returned. Under CICS, if the ADL user exit ADLEXITB is installed but the ADL Interfaces are not activated, a value '4' is returned.

In general, a program which updates migrated data, should only run when the response from ADLACTIV is '0'.

Example

```

DEFINE DATA LOCAL
1 #RSP(I2)
END-DEFINE
CALLNAT 'ADLACTIV' #RSP
DECIDE ON FIRST VALUE OF #RSP
VALUE 0
  WRITE 'ADL Consistency is active.'
VALUE 4
  WRITE 'ADL user exit installed, but ADL is not active.'
VALUE 8
  WRITE 'ADL Consistency is not active.'
NONE
  WRITE 'Unexpected response from ADL Consistency.'
END-DECIDE
END

```

Example Programs

The following Natural programs demonstrate how to code with respect to the referential integrity. The full sources can be found on the SYSADLIV Natural library.

Program INS-STUD: Insert a Student

Before a dependent segment is inserted, it should be verified that the parents, i.e. the path to the segment, exists.

```

* Check if the path to the student exists
*
FIND (1) COURSEDB-CLASS WITH CLASSNO = #CLASSNO
  WHERE PCK-COURSENO = #COURSENO
  IF NO RECORDS FOUND
    REINPUT 'COURSE / CLASS NOT FOUND !'
  END-NOREC
*
* Store the data in the data base
*
  MOVE #COURSENO TO COURSEDB-STUDENT-UPD.PCK-COURSENO
  MOVE #CLASSNO TO COURSEDB-STUDENT-UPD.PCK-CLASSNO
  MOVE #SURNAME TO COURSEDB-STUDENT-UPD.SURNAME
  MOVE #FILLER-AB TO COURSEDB-STUDENT-UPD.FILLER-AB
  STORE COURSEDB-STUDENT-UPD
  END OF TRANSACTION
END-FIND

```

Program DEL-COUR: Delete a Course

Before a parent is deleted, it should be verified that it has no dependents. The following program does not perform the delete if there are dependents

```

* Check if the data exists
*
COURSE. FIND (1) COURSEDB-COURSE WITH COURSENO = #COURSENO
  IF NO RECORDS FOUND
    REINPUT 'COURSE NOT FOUND !'
  END-NOREC
*
* Check if there are dependents
*
  RESET #NBR
CLASS. READ COURSEDB-CLASS WHERE PCK-COURSENO = #COURSENO
  MOVE *COUNTER TO #NBR
  WRITE #NBR '. CLASS FOUND UNDER THIS COURSE:' CLASSNO CLASSNAME
  END-READ
*
  IF #NBR = 0 /* NO DEPENDENTS
*
* Delete the data
*
  DELETE (COURSE.)
  END OF TRANSACTION
  WRITE 'COURSE DELETED!'
  END-IF
END-FIND

```

Program CDELCOUR: Cascaded delete of a Course

When DL/I deletes a segment, all dependents are deleted automatically. This is named “*cascaded delete*”. The program CDELCOUR (together with the programs called by it) is an example on how to code a cascaded deletion with Natural.

```
FIND (1) COURSEDB-COURSE WITH COURSENO = #COURSENO
  IF NO RECORDS FOUND
    MOVE 'Course not found!'
      TO #ADL-MESS
    ESCAPE BOTTOM
  END-NOREC
* Delete dependents recursively
  CALLNAT 'CASDELCL' #COURSENO #N-CL #N-ST #N-IP #RESPONSE
  IF #RESPONSE NE 0
    ESCAPE BOTTOM
  END-IF
* All dependents are deleted -> delete the COURSE
  DELETE
  MOVE 1 TO #N-SUM
END-FIND
```

8

Converting Natural for DL/I Programs

■ Introduction	110
■ Conversion of the Data Definitions and of the Data	110
■ Modification of the Application	111

This chapter covers the following topics:

Introduction

The Natural for DL/I (NDL) interface provides access to data stored in an IMS/DB database for Natural applications. When the DL/I databases access by NDL are converted into Adabas files with the ADL, the NDL applications can continue to run like any other DL/I application. NDL converts the Natural generated internal Adabas call into an DL/I call which is converted by ADL into an Adabas call and forwarded to Adabas. On the way back, the response of Adabas is converted by ADL into a DL/I response and returned to NDL which converts it into an Adabas response for the Natural application.

NDL applications have special restrictions and enhancements caused by the hierarchical structure of the underlying database. Thus after the migration, the NDL application cannot run “as they are” against Adabas. This chapter describes what you have to respect when you convert a NDL program to a “normal” Natural/Adabas program.

After the conversion, the Natural program must run through the ADL Consistency as long as there is any DL/I application (Cobol, PL/I) accessing the same data. The restrictions for Natural programs running against the ADL Consistency are described in the section [Using ADL Files with Natural/Adabas](#) in this documentation. Although the Consistency is an overhead too, the conversion from Natural for DL/I to Natural/Consistency has several advantages:

- Better performance because read calls are routed directly to Adabas.
- The hierarchical view to the data is replaced by a flat view with foreign keys.
- Normal Natural (i.e. Natural/Adabas) syntax can be used. No special NDL syntax restrictions, error messages etc. must be obeyed.
- When all NDL applications are converted, NDL is no longer required.
- Natural with the Consistency is the path towards standard Natural/Adabas. I.e. as soon as no DL/I application accesses the data anymore, the ADL Consistency is obsolete, too.

Conversion of the Data Definitions and of the Data

The DBD definitions accessed by the NDL application must be converted with the ADL Control Block Conversion (CBC) utility. You can use one file per segment (GENSEG statement of the ADL CBC utility), or collect more than one segment in one file. The ADL Conversion Utility is described in the *ADL Conversion* documentation.

In a first step, the FDT is loaded without data into Adabas. Then load the field descriptions generated by ADL into Predict with the Predict Incorporate function.

Build new Predict masterfiles and DDMs from the ADL definitions and from the NDL definition.

Use from ADL

Field	Description
Zx	ADL internal fields.
Xx	Descriptors corresponding to secondary indices.
Sx	Adabas groups corresponding to DL/I segments. Use the DL/I segment name as long name.

Use from NDL

Field	Description
xx	All Adabas field definitions corresponding to parts of the DL/I segment. Take care that the short names are unique. These fields must be part of the group corresponding to the DL/I segment.

Special considerations for reference fields:

- Define the sequence fields as descriptors. Note that the root sequence field is already defined by ADL as descriptor. The short name of the root sequence field must be the one defined by ADL.
- For the foreign primary key fields, i.e. the PCK and VCK fields, use the ADL short name (normally Px/Qx), and the NDL long name. Define these fields as descriptors.
- Build superdescriptors with the source segment to reflect secondary indices. Use NDL long-names for those superdescriptors.
- Do not allocate fields with foreign keys of secondary indices for dependent segments. Because the secondary index fields belong to another segment, DL/I applications would not fill these fields.

After this modifications have been performed, generate ADACMP cards with Predict. Then unload the data from DL/I and load it into Adabas with ADL utilities. For the compression, use the ADACMP cards from Predict. The data conversion is described in the section ADL Conversion Utility in the *ADL Conversion* documentation.

Modification of the Application

This section describes what have to be considered in the NDL applications concerning database related commands when converting to Natural/Adabas and the ADL Consistency.

READ/FIND commands

READ and FIND commands are only affected when the application reads a dependent segment and uses in the root level a secondary index. In this case the root level must first be read with the sec-

ondary index to get the primary key, and this must be used as foreign key when reading the dependent.

Example

We have a root segment “A” with primary and secondary index and a dependent segment “B” with a sequence field. We would like to access segment B and use on the root level the secondary index.

Natural Name	Description
ROOT-A-VIEW	view of root segment A
SEG-B-VIEW	view of dependent segment B
A-PRIMKEY	root primary key
ROOT-PRIM	root primary key used as foreign key
A-SEC	root secondary index
ROOT-SEC	root secondary index used as foreign key
SEG-B-SEQ	dependent segment sequence field

NDL

```
FIND  SEG-B-VIEW WITH ROOT-SEC=SEC-VAL1    /* sec index
      AND SEG-B-SEQ=B-VAL2                /* dependent
```

ADL


```
FIND  ROOT-A-VIEW WITH A-SEC=SEC-VAL1      /* sec index
MOVE  A-PRIMKEY to #PRIM-VAL1             /* prim key
FIND  SEG-B-VIEW WITH ROOT-PRIM=#PRIM-VAL1 /* prim key
      AND SEG-B-SEQ=B-VAL2                /* dependent
```

STORE

When performing a STORE command, the foreign keys must be filled. Before the command is executed, it must be verified that the referenced values exist. This is eventually already ensured by the application logic. In case of an inconsistency an error handling must be provided.

DELETE

DL/I and thus NDL too, offer a cascaded delete. When a segment occurrence is deleted, all dependent segment occurrences are deleted automatically. In Natural/Adabas there is no such function. If a record has dependents, i.e. records which refer the current sequence field as foreign keys, these records must be deleted explicitly.

 **Note:** As long as the ADL Consistency is active, the dependents must be deleted from bottom to top and finally the record itself can be deleted

UPDATE

UPDATE commands are not affected by the conversion.



Note: Sequence fields and foreign keys can neither be updated with NDL nor with the ADL Consistency. You should carefully evaluate, before updating such a field with Natural/Adabas.

END TRANSACTION/BACKOUT TRANSACTION

DL/I makes an implicit checkpoint at each terminal I/O. Because Adabas does not make any implicit ET call, the transaction logic of the application must be checked and if required, additionally 'END TRANSACTION' statements must be added.

ERROR HANDLING

There will be no more NDL specific error message in the application. The error handling must reflect only Natural/Adabas error situations. Additionally when the ADL Consistency is used, a Consistency response code can be obtained. This is described in details in the chapter *Using ADL Files with Natural/Adabas* in this documentation.

9 SQL Access to the Migrated Files

- Introduction 116
- How to Access the Migrated Data 116
- Improving the Access to Migrated Data 117
- Restrictions for SQL Applications 117

This chapter covers the following topics:

Introduction

The Adabas SQL Gateway provides business users real-time access to Adabas data, no matter where that data resides, using standard SQL-based desktop tools.

With Adabas SQL Gateway, mission-critical Adabas data can be accessed quickly and easily by any ODBC, JDBC, OLE DB or .NET standard SQL application. IT and business users also can access Adabas with SQL-based reporting tools, such as Business Objects, Crystal Reports and MS Office. With Adabas SQL Gateway, IT can deliver business users the information they need, straight to their desktops.

After the DL/I databases have been converted to Adabas files, these files can be accessed by SQL applications with the Adabas SQL Gateway. The advantages for the Adabas data offered by the Adabas SQL Gateway can be extended to DL/I data this way.

This section explains:

- How to access the migrated data from SQL applications.
- How to improve the access.
- What restrictions apply for the SQL applications.

The installation and operation of the Adabas SQL Gateway is described in the Adabas SQL Gateway manual.

How to Access the Migrated Data

Import the Adabas definitions to the Adabas SQL Gateway data dictionary with the CONNX Data Dictionary Manager. You can either use the FDT import or if you have already generated DDMs for Natural, you can use the SYSTRANS import function.

As described in the section [Using ADL Files with Natural/Adabas](#) in this documentation, ADL mirrors the hierarchical structure defined in the DL/I database by using physical pointers and foreign keys. An SQL table should reflect all the fields belonging to a DL/I segment plus the foreign key fields. SQL tables should not contain the physical pointer fields nor other ADL internal fields (Z0 – Z8, secondary index descriptor).

Once the data definitions are imported, you can read the data with SQL applications.

Improving the Access to Migrated Data

Depending on the data you want to read with the SQL application, you should consider defining new descriptors and superdescriptors in the Adabas file. You may define as descriptor:

- Sequence fields of dependent segments (the root sequence field is already a descriptor).
- Foreign key fields.

You may define as superdescriptors:

- The concatenated key fields, i.e. all foreign key fields of the parents of a segment. This super-descriptor may additionally include the sequence field of the segment itself if available.
- Fields building up a secondary index.

You may define other fields as descriptors or superdescriptors even if there is no counterpart in the DL/I structure. But ...

- Do not build superdescriptors from fields of different segments. It will not contain any data.
- Do not allocate descriptors/superdescriptors which you do not use. It is an overhead in performance.

Restrictions for SQL Applications

SQL applications must not update the migrated data. If an SQL application would insert dependent segment data, DL/I applications would not “see” this data. This is because there is no “Consistency” for SQL which updates the ADL internal physical pointer fields which are required for the DL/I applications.

Moreover if an SQL application updates the data, it can destroy the hierarchy. For example if it deletes a root segment data, the dependent data is still in the database but cannot longer be accessed by DL/I applications. Once again, no “Consistency” prevents the SQL application from destroying the referential integrity.

Note that these restrictions apply only as long as DL/I applications access the data. As soon as all DL/I applications have been replaced, you can modify the data with SQL applications without restrictions.

10 Debugging Aids - ADL Trace Facility

▪ Data Base Call Trace	120
▪ Internal Routine Trace	124
▪ z/OS JCL Requirements	125
▪ z/VSE JCS Requirements	126

The Adabas Bridge for DL/I includes a Trace facility with which you can gather information on the following:

- DL/I and Adabas calls (referred to as “database call trace”).
- Internal routine ADL calls (referred to as “routine trace”)

You may activate the Trace facility in different ways:

- By specifying the TRACE parameter in the ADL parameter module (see the section *ADL Parameter Module* in the *ADL Installation* documentation).
- By specifying the TRACE parameter as one of the parameters for the ADL initialization program, DAZIFP (see the section *Batch Installation and Operation* in this documentation for details), or
- Dynamically under CICS in the CICS Maintenance of the ADL Online Services or by running the DAZT transaction.

This chapter covers the following topics:

Data Base Call Trace

Tracing DL/I and Adabas Calls

The Trace facility for DL/I and Adabas calls consists of two parts, one that is responsible for data acquisition and the other for reporting.

Data Acquisition

The data acquisition part of the Trace facility collects data for every DL/I and Adabas call while the application program is running, and stores them in a sequential file known as the trace file.

The default setting causes data to be collected for all calls, from the beginning of the job to the end. However, you may also use the TRACE parameter operands listed below to trace a specific number of DL/I calls or to start a trace at a particular call.

When you activate the Trace facility under CICS with the ADL Online Services, the system prompts you for a terminal ID and the type of trace wanted. Specifying a terminal ID results in a trace of calls from the specified terminal only. Any blank character in the specified terminal ID is treated as a wildcard, i.e. it matches all characters. Thus, if a blank terminal ID is specified (4 blank characters) a trace of calls from all active terminals is conducted.

The syntax of the Trace parameter for DAZIFP or the ADL Parameter Module is as follows:


```
TRACE=( [t],[s],[a],[n],[m],[c],[u],[b])
```

All parameters are positional parameters. Trailing commas can be omitted, e.g.

```
TRACE=(RD)
```

The operands of the TRACE parameter have the following meaning:

Operand	Explanation
t	The type of trace requested. One or more of the following values may be specified: R : All internal routine calls will be traced. D : All DL/I and Adabas calls will be traced. U : Special trace for the CBC utility. The order in which these values are specified is irrelevant. Default: None
s	The start, i.e. the first DL/I call to be traced. Possible values: 1 - 9999999 Default: 1
a	The number of DL/I calls to be traced. Possible values: 1 - 9999999 Default: 9999999
n	The name of the routine at which the routine trace is to be started. The name given must start with "DAZ", and may have a maximum of 8 characters. Default: DAZCOMDC
m	The number of the call to the routine specified in "n" at which the routine trace is to be started. An entry of 0 causes "n" to be ignored and the complete routine trace to be printed. Default: 0
c	The name of the routine which is to be counted. The name given must start with "DAZ", and may have a maximum of 8 characters. Default: DAZCOMDC
u	The logical unit number (for z/VSE only). Possible values: 0-999 Default: 11
b	The block size (for z/VSE only). Possible values: 0-99999 Default: 8196

For the ADL CALLDLI Interface the data collected contain the following information:

- The "before" image of a DL/I call (i.e. its form before it is passed to the system): the function, SSAs and I/O area (for insert and replace calls) are traced.
- The "after" image of an Adabas call (i.e. its form when it is returned by the system): the Adabas control block and buffers (any of the format, record, search, value and ISN buffers, if used) are traced. The fields of the Adabas control block are listed individually in readable format (character

or numeric) and in hexadecimal format (denoted by a leading “x”). The buffers are listed in hexadecimal and in character format.

- The “after” image of a DL/I call: the user PCB, key feedback area and I/O area (for retrieval calls) are traced.

```

000001 GU
          SSA1 :C3D6E4D9 E2C54040 4DC3D6E4 D9E2C5D5 D67E40D4 C1E3C8C5 D4C1F2F0
F05D      *COURSE (COURSENO= MATHEMA200) *
-----
L6  ISN: 0000000021 x00000015  DBID: 01424 x0590  FNR: 00833 x0341  RSP:
00000 x0000  CID: 'Z???' xE9010101 TERM:
      TIME: 0000000005 x00000005  ISNL: 0000000000  ISNQ: 0000000000  OP1:
'R'  xD9  OP2: 'V'  xE5  TASK:
      ADD1: 'AA' C1C1BBCA40404040  ADD2: ' ?' 001E003F  ADD3: 4040404040404040  ADD4:
4040404040740590  ADD5: 0000000000000000
      FBL: 0024 FB: F4E76BF4 E76BE9F4 6BF8E76B F8E76BF4 E76BF4E7 6BE2C14B
          *4X,4X,Z4,8X,8X,4X,4X,SA.  *
      RBL: 0063 RB: 40404040 40404040 00404040 40404040 40404040 40404040 40404040
40404040  *
          40D4C1E3 C8C5D4C1 F2F0F0D4 C1E3C8C5 D4C1E3C9 C3E24040 40404040
404040  * MATHEMA200MATHEMATICS  *
      SBL: 0003 SB: C1C14B
          *AA.  *
      VBL: 0010 VB: D4C1E3C8 C5D4C1F2 F0F0
          *MATHEMA200  *
-----
      DBD: COURSEDB STA:  SEG: COURSE  LEV: 01 SEN: 00000004 OPT: AP  RES:
01030470
      KYL: 0010 KY: D4C1E3C8 C5D4C1F2 F0F0
          *MATHEMA200  *
      IOL: 0030 IO: D4C1E3C8 C5D4C1F2 F0F0D4C1 E3C8C5D4 C1E3C9C3 E2404040 40404040
4040  *MATHEMA200MATHEMATICS  *

```

Database call trace

For the ADL Consistency Interface, the following data are collected:

- A “before” image of the user Adabas call.
- A “before” image of internally emulated DL/I type calls. Only the I/O area but no SSAs will be traced.
- An “after” image of Adabas calls issued by ADL i.e. in its form when it is returned from Adabas.
- An “after” image of internally emulated DL/I type calls.
- An “after” image of the user Adabas call. This is after the Consistency has completed its processing, but before the call is either passed to Adabas or control is returned to the user. Note that a non-zero Adabas return code displayed is the actual ADL return code (in a range from 1201 to 1299) and not the return code which will be seen by the user (216).

Only those user Adabas calls will be traced which perform an update function (insert, delete or replace) and reference to an ADL file.

For the ADL Consistency Interface it is recommended to overwrite the default routine name of the 'n' and 'c' operands with the ADL routine name `DAZCFDEC`.

Data Reporting

The data reporting part of the Trace facility reads and interprets the data stored in the trace file and produces a trace report. As such, it is part of the ADL nucleus and is activated by executing the ADL initialization program `DAZIFP` with the parameters shown below as provided by the file `DAZIN1` (see the section [Batch Installation and Operation](#) for details).

```
PRT,DAZPRINT
```

You can control the Trace data reported using an additional control statement read in from either `DAZIN1` (z/OS) or `DAZIN2` (z/VSE). This statement has one of the following two formats and options:

```
MODE=SHORT[,TERM=t][,TASK=i]
```

or

```
MODE=FULL[,ADA=x][,DLI=y][,TERM=t][,TASK=i]
```

where

Operand	Explanation
t	is a four-character CICS terminal ID indicating that only DL/Iand Adabas calls originating from the specified terminal will be repeated.
i	is a task number up to five digits long indicating that only DL/Iand Adabas calls originating from the specified task number will be repeated.

x is any combination of the following values:

Value	Suppressed Buffer/Calls
R	Adabas record buffer
F	Adabas format buffer
S	Adabas search buffer
V	Adabas value buffer
I	Adabas ISN buffer
N	Adabas calls before the first DL/I call

and *y* is any combination of the following values:

Value	Suppressed Areas
I	DL/I I/O area
K	DL/I key feedback area
P	DL/I user PCB
S	DL/I SSAs

Specifying any of the x or y parameter values suppresses reporting of the indicated buffer or area.



Note: MODE=SHORT

is the same as

```
MODE=FULL,ADA=RFSVI,DLI=IKS
```

Example:

```
MODE=FULL,ADA=RFI,DLI=I
```

This parameter statement would cause the Adabas search and value buffers and the DL/I PCB, key feedback area and SSAs to be reported.

Internal Routine Trace

The Trace facility may be used to trace all internal routine calls for ADL. The default setting causes all routine calls from the beginning of the job to the end to be traced. However, you may use the TRACE parameter operands listed earlier in the section on Data Acquisition to specify the name of the routine at which tracing should be started and to ignore the first “n” times the routine was called.

In batch, the routine calls trace is routed direct to the printer. In CICS, an extra partition data set is used for this. Any standard printing utility, or the DAZPRINT utility, may then be used to print the trace. The DAZPRINT utility is activated by executing the ADL initialization program DAZIFP with the parameters shown below (see the section *Batch Installation and Operation* for details):

```
PRT,DAZPRINT
```

The following additional parameter statement must be specified:

```
MODE=ROUTINE[,TERM=t]
```

where

t is a four-character CICS terminal ID, indicating that only routine calls originating from the terminal specified should be printed.

z/OS JCL Requirements

The JCL requirements for the Trace facility are discussed below. Explanations of how to use the Trace facility during a batch run and how to print out the trace data set are given.

Using the Trace Facility

The table below lists the data sets used by the ADL batch monitor when the Trace facility has been activated. These data sets are used in addition to any others.

DDname	Medium	Description
DAZOUT5	Tape/Disk	Trace output file *
DAZOUT1	Printer	ADL routine calls trace **

* Only required when the trace to be performed is to include DL/I and Adabas calls.

** Required when the trace to be performed is to include internal ADL routine calls. This file is also used to print error messages, and should therefore already have been included.

Example

```
// EXEC PGM=DAZIFP,PARM='DLI,pgmname,psbname,TRACE=(RD)'  
//STEPLIB DD DSN=ADL.LOAD,DISP=SHR  
// DD DSN=ADABAS.LOAD,DISP=SHR  
//DDCARD DD *  
ADARUN PROGRAM=USER,...  
//DAZOUT1 DD SYSOUT=X  
//DAZOUT5 DD DISP=SHR,DSN=d-trace-dataset
```

Printing the Trace Data Set

The following table lists the data sets used by the Print Trace utility, DAZPRINT.

DDname	Medium	Description
DAZIN1	Reader	Control input for the Print utility, DAZPRINT.
DAZOUT1	Printer	Report, messages and codes.
DAZIN4	Tape/Disk	DL/I and Adabas calls trace file.
DAZIN5	Tape/Disk	Routine trace file.

Examples

The following is an example of a job to print the data base call trace:

```
// EXEC PGM=DAZIFP,PARM='PRT,DAZPRINT'  
//STEPLIB DD DSN=ADL.LOAD,DISP=SHR  
//DAZIN4 DD DSN=d-trace-dataset,DISP=SHR  
//DAZOUT1 DD SYSOUT=X  
//DAZIN1 DD *  
MODE=FULL  
//*
```

The following is an example of a job to print the internal routine trace from disk:

```
// EXEC PGM=DAZIFP,PARM='PRT,DAZPRINT'  
//STEPLIB DD DSN=ADL.LOAD,DISP=SHR  
//DAZIN5 DD DSN=r-trace-dataset,DISP=SHR  
//DAZOUT1 DD SYSOUT=X  
//DAZIN1 DD *  
MODE=ROUTINE  
//*
```

z/VSE JCS Requirements

The JCS requirements for the Trace facility described in this section are discussed on the following pages. Explanations of how to use the Trace facility during a batch run and how to print out the trace data set are given.

Using the Trace Facility

The table below lists the files used by the ADL batch monitor when the Trace facility has been activated. These files are used in addition to any others.

DTF	Logical Unit	Medium	Description
DAZOT5D	SYS012	Disk	Trace output file. *
DAZOT5T	SYS012	Tape	Trace output file. *
DAZOUT1	SYSLST	Printer	ADL routine calls trace. **

* Only required when the trace to be performed is to include DL/I and Adabas calls. Only one of the two files is needed.

The logical unit indicated is the default logical unit. To change it, specify the TRACE parameter either in the ADL parameter module or as a dynamic parameter:

```
TRACE=(,,,,,6)
```

** Required when the trace to be performed is to include internal ADL routine calls. This file is also used for the printing of error messages, and is therefore automatically included.

Example

```
// ASSGN SYS012,DISK,VOL=volume,SHR
// DLBL DAZOT5D,'d-trace-dataset'
// EXTENT SYS012,volume,.....
// EXEC PROC=ADLLIBS
// EXEC DAZIFP
DLI,pgmname,psbname,TRACE=(RD)
/*
ADARUN PROGRAM=USER,.....
/*
/&
```

Printing the Trace Data File

The following table lists the files used by the Print Trace utility, DAZPRINT.

DTF	Logical Unit	Medium	Description
DAZIN1	SYSIPT	Reader	Control input for the ADL batch monitor, DAZIFP.
DAZIN2	SYSIPT	Reader	Control input for the Print utility, DAZPRINT.
DAZIN4D	SYS012	Disk	DL/I and Adabas call trace file. *
DAZIN4T	SYS012	Tape	DL/I and Adabas call trace file.*
DAZOUT1	SYSLST	Printer	Report, messages and codes.
DAZIN5D	SYS014	Disk	Routine trace file. **

DTF	Logical Unit	Medium	Description
DAZIN5T	SYS014	Tape	Routine trace file. **

* Only one of the two files is required. The logical unit indicated is the default logical unit. To change it, specify the TRACE parameter either in the ADL parameter module or as a dynamic parameter:

```
TRACE=(,,,,,6)
```

** Only one of the two files is required. The logical unit indicated is the default logical unit. To change it, specify the FX parameter in the ADL parameter module:

```
FX=(15,)
```

The control input for both the batch monitor (DAZIFP) and DAZPRINT is read from SYSIPT. The control statements must be specified in the following order:

```
PRT,DAZPRINT          input for DAZIFP
/*
MODE=FULL             input for DAZPRINT
/*
```

Examples

The following is an example of a job to print the data base call trace:

```
// ASSGN SYS012,DISK,VOL=volume,SHR
// DLBL DAZIN4D,'d-trace-dataset'
// EXTENT SYS012,volume
// EXEC PROC=ADLLIBS
// EXEC DAZIFP
PRT,DAZPRINT
/*
MODE=FULL
/*
/&
```

The following is an example of a job to print the internal routine trace:

```
// ASSGN SYS014,DISK,VOL=volume,SHR
// DLBL DAZIN5D,'r-trace-dataset'
// EXTENT SYS014,volume
// EXEC PROC=ADLLIBS
// EXEC DAZIFP
PRT,DAZPRINT
/*
MODE=ROUTINE
/*
/&
```


11 CALLDLI Test Program - DAZZLER

■ Introduction	130
■ DL/I Statements	131
■ Sets	131
■ DAZZLER Control Statements	135
■ z/OS JCL Requirements for DAZZLER	137
■ z/VSE JCS Requirements for DAZZLER	137

This chapter covers the following topics:

Introduction

DAZZLER is a special test utility provided with the Adabas Bridge for DL/I. It is used to format DL/I calls and to monitor their results. This may be necessary, for example, to compare DL/I and Adabas performance.

DAZZLER handles up to the following maximum values:

Number of PCBs	10
Number of SSAs	15
I/O area length (bytes)	256
SSA length (bytes)	256

You are most likely to want to use DAZZLER for low volume input, and for this reason the eight possible statement types have been divided into two groups of four. The first group contains those statements necessary for creating DL/I calls, and are referred to as DL/I statements. The second group consists of statements provided for the user's benefit, particularly when handling medium-to-large volumes of input. The statements in this second group are known as DAZZLER control statements, and are all optional.

All statements in both groups contain a number of validated fields. The validation rules for each field are explained in the discussion of the individual statements.

You should note that the first three positions of each statement contain an identifying title. Invalid titles, and thus the information on the statement concerned, are immediately rejected.

Blank statements are ignored.



Note: All PSBs used for the DAZZLER program must have been created with either the PSBGEN statement option `LANG=ASSEM` or `LANG=COBOL`. Alternatively, if the PSB had been created with `LANG=PLI`, you must specify `LANG=COBOL` as DAZIFP parameter.

Appendix A shows an example of DAZZLER usage.

DL/I Statements

The following table lists the DL/I statements.

Statement	Function
HEADER Statement	A compulsory statement which contains the DL/I call and other important related parameters. It must be the first statement within the set (see below).
IOA Statement	An optional "input/output area" statement which, when present, causes the existing I/O area to be overwritten.
SSA Statement	An optional "segment search argument" statement which, when present, causes a specified segment search argument to be overwritten.
SUMMARY Statement	An optional statement which, when present, produces a tabular summary of all segment names and how many occurrences there are in a given PCB.

Sets

Each individual DL/I call requires that a set of statements be input — a set being defined as a group of statements beginning with a `HEADER` statement and ending with either the statement preceding the next `HEADER` or `SUMMARY` DL/I statement, `JUMP` DAZZLER control statement, or with the end of the file, as appropriate. A `SUMMARY` statement is considered to be a set in itself.

In other words, a set can consist of:

- only a `HEADER` statement,
- a `HEADER` statement followed by any combination of other DL/I or DAZZLER control statements (any `COMMENT` statements encountered will be ignored), or
- only a `SUMMARY` statement.

If the `HEADER` statement of a set is rejected by the validation routines, all the remaining statements in the same set will be rejected as well.

The following pages describe the individual statements, their layouts and the validation routines appropriate in each case.

HEADER Statement

```

....+....1....+....2....+....3....+....4....+....5....+....6....+
*HD call nnn cc pp ss iiii +++++ ssslssslssslssslssslssslssslsssl

```

The following pages explain the individual entries on the statement in detail.

Statement Field	Position	Status*	Description
*HD	1-3	C	The identifier for the HEADER statement.
call	5-8	C	The kind of DL/I call. This is validated against an internal table and must be four characters long, plus trailing blanks if appropriate. e.g. "DLET", "REPL", "GU", "GHNP", etc.
nnn	10-12	O	The number of times the set is to be run. The range is 1-999, with leading zeros as appropriate. The default is 001. See also the "cc" field.
cc	14-15	O	Any non-blank entry overrides any value entered in the "nnn" field. The set is repeated until the status code returned by DL/I corresponds exactly to the entry made here. An entry of "???" instructs DAZZLER to stop repeating the call as soon as a non-blank status code is returned.
pp	17-18	C	The PCB number to be used for this call.
ss	20-21	O	The number of SSAs in the call. The range is 00 to 15 (a leading zero must be provided if necessary). The default is 00.
iiii	23-27	O	If this parameter is omitted, or if "00000" is entered, no further validation is performed and any further coding in this statement is treated as a printable comment. If a five-figure number (including leading zeros) is entered, this value is used as the initial value for an iterative overwrite of the IOA. All remaining fields are then used to specify this overwrite.
+++++	29-33	O	A five-figure number with leading zeros which will be used as the increment for the iterative overwrite initiated by the entry in "iiii". 00001 is the default.
sssl...etc	35-66	O/C **	Each occurrence of "sssl" represents the start position (SSS) and the length ("l") of the iterative overwrite of the IOA. The "sss" entry must include leading zeros if necessary and be in the range 1-256. The "l" entry must be in the range 1-5. In addition, the end position calculable from the two entries must be less than 257. A minimum of 1 and a maximum of 8 "sssl" entries may be provided. Blank entries are ignored.

* C = compulsory, O = optional

** Optional unless an entry has been made in "iiii", in which case compulsory.

Example of the Iterative Overwrite Facility

Where *nnn* =004, *iiii* contains 00100, *++++* contains 00035, and the following *ssl* entries are provided

```

ssl|ssl|ssl|ssl|ssl|ssl|ssl|ssl|
+....4....+....5....+....6....+
02040352    0615    0013    1111

```

then the results of the coding are:

IOA Positions	1st Iteration	2nd Iteration	3rd Iteration	4th Iteration
20,21,22,23	0100	0135	0170	0205
35,36	00	35	70	05
61,62,63,64,65	00100	00135	00170	00205
1,2,3	100	135	170	205
111	0	5	0	5

BACK Call

In addition to DL/I calls you can specify *BACK* as the call in the *HEADER* statement. This results in an Adabas BT (Backout Transaction) call which removes all data base modifications performed during the user's current logical transaction. The *BACK* call can only be used, if *DAZZLER* runs as an online batch application, i.e. with *BMP*, *MPS* or *SDB* as first *DAZIFP* parameter. To prevent conflicts with the automatic *ET* functionality, it is recommended to specify *ET=NO* as *DAZIFP* parameter or in the *ADL* Parameter Module. The *BACK* call can not be used while *DAZZLER* runs against *DL/I*.

IOA and SSA Statements

```

....+....1....+....2....+....3....+....4....+....5....+....6....+
tit form ssl 11 ddddddddddddddddddddddddddddddddddddddddddddddddddd

```

Statement Field	Position	Status*	Description
tit	1-3	C	Two settings are possible: IOA - Indicates that the I/O area is to be overwritten. Snn - Indicates that an SSA is to be overwritten. The value "nn" specifies the SSA concerned and must be in the range 1-15, with a leading zero if necessary. The entry in "nn" is validated for being not greater than the number of SSAs coded on the <i>HEADER</i> statement.
form	5-8	O	A four-character entry with a trailing space if necessary. Indicates the format for the rewrite to the program. The possible values are:

Statement Field	Position	Status*	Description
			CHAR Characters
			HEX Hexadecimal data
			PCK Positive packed decimal number
			PCK+ Positive packed decimal number
			PCK- Negative packed decimal number
			The default is CHAR.
sss	10-12	C	A three-figure number with leading zeros, if necessary, which specifies the start position for the overwrite. It must be in the range 1-256.
ll	14-15	C	Two-digit string, with a leading zero if necessary, specifying the length of the overwrite. It is dependent on the entry made under "form". If this specifies character format, the "ll" entry determines the number of bytes to be overwritten. If it specifies non-character format, it specifies the number of digits to be read from the statement. The entry must be in the range 1-50, except for packed decimal data, in which case the maximum is 18. The calculable end position for the overwrite must be less than 257.
dddd ...	17-66	C	The data to be used in the overwrite are entered here, left-justified. Overwrites in hexadecimal format cause the program to validate the data for being in the ranges 0-9 or A-F and for being an even number of digits. Packed decimal format causes validation for the range 0-9.

* C = compulsory, O = optional

The I/O area and the SSAs are only overwritten when this is specifically requested. If a subsequent call requires some or all of the same SSAs or the same data in the I/O area, it is not necessary for these to be recoded. You should note that the I/O area may well change as the result of a DL/I call.

SUMMARY Statement

```
....+....1...
```

```
SUMMARY pp
```

Statement Field	Position	Status	Description
pp	9-10	Compulsory	The required PCB number. The range is 01-10. A leading zero must be provided if necessary.

This statement causes the program to produce a table showing, for any one PCB, all the retrievable segment names and how many occurrences of each there are. The statement is NOT a part of a set as defined earlier, but rather can be considered a set in itself. When a SUMMARY statement has been encountered and validated as correct, the program issues an unqualified GU to establish position on the first segment and then issues successive GN calls until a status code of "GB" is returned. The table is then printed out and the program continues with the next statement. A SUMMARY statement

may appear anywhere in the job stream and will also be recognized as ending a previous HEADER statement set.

DAZZLER Control Statements

Statement	Function
PRINT CONTROL PARAMETER	An optional statement which provides run control parameters. It comes at the front of the input stream.
COM(MENT)	This statement enables users to make comments which will not subsequently be printed. It can appear anywhere in the job stream, and is ignored as soon as it is encountered. It is used to help understand the test stream.
JUMP	This statement tells DAZZLER to ignore all following statements until the corresponding LABEL statement is read. It allows users to jump down the stream and skip some or all statements.
LABEL	This statement is the partner of a JUMP statement. It places a label in the job stream to indicate the point at which the program is to restart processing statements. If no prior associated JUMP statement exists, the LABEL statement is ignored.

PRINT CONTROL PARAMETER Statement

If the PRINT CONTROL PARAMETER statement is omitted, dates are printed with the format "DD/MM/YY" and a page throw occurs before each new set of statements.

The layout of the statement is as follows:

```
....+....1....+...
PARddtccssssppee
```

Statement Field	Position	Status*	Description
PAR	1-3	C	The identifier for the statement.
dd	4-5	C	The date print field. Two values are possible: US - Dates are printed in the format MM/DD/YY. AM - Dates are printed in the format DD/MM/YY. If the PRINT CONTROL PARAMETER statement is not used, then the date will automatically have the format DD/MM/YY.
t	6	O	If an entry is made here, the automatic page throw for every new call is suppressed.
cc	7-8	O	Specifies what is to be printed. Permissible entries are: PA Print all.

Statement Field	Position	Status*	Description										
			<table border="1"> <tr> <td>PN</td> <td>Print no results from the calls.</td> </tr> <tr> <td>PF</td> <td>Print from a specified call number onwards.</td> </tr> <tr> <td>PC</td> <td>Print no results until a non- blank status code is returned: after this, print the number of calls specified by the "pp" field (see below).</td> </tr> <tr> <td>PS</td> <td>Print slots (e.g. print every 20th call).</td> </tr> <tr> <td colspan="2">The default is PA.</td> </tr> </table>	PN	Print no results from the calls.	PF	Print from a specified call number onwards.	PC	Print no results until a non- blank status code is returned: after this, print the number of calls specified by the "pp" field (see below).	PS	Print slots (e.g. print every 20th call).	The default is PA.	
PN	Print no results from the calls.												
PF	Print from a specified call number onwards.												
PC	Print no results until a non- blank status code is returned: after this, print the number of calls specified by the "pp" field (see below).												
PS	Print slots (e.g. print every 20th call).												
The default is PA.													
sssss	9-13	O	Specifies the start call number when PF or PS have been entered in the "cc" field. If entered, the start call number must be a five-figure number with leading zeros if necessary.										
pp	14-15	O	Two uses are possible: Specifies the number of calls to be printed after a PC or a PF. The value entered must be a two-figure number with a leading zero if necessary. An entry of 99 means that all remaining calls will be printed. For the PS parameter, specifies the number of calls either side of the selected call number to be printed.										
eee	16-18	O	Used only with the PS parameter. Specifies the number of calls to be skipped when slots are to be printed. It is the repeating increment and must be a three-figure number (including leading zeros where necessary).										

* C = compulsory, O = optional

COMMENT Statement

```
.....+.....1.....+...
COM
```

Positions 1-3 must contain COM. This statement will be ignored by the program and is provided so that users can increase the legibility of their test streams.

JUMP and LABEL Statements

```
.....+.....
JUMP 111
LABEL111
```

This pair of statements is provided so that users can ignore calls in the test stream. As soon as the program reads the JUMP statement ("JUMP" in columns 1-5), it ignores all subsequent statements until the matching LABEL statement ("LABEL" in columns 1-5) is found. The match is performed on columns 6-8 (inclusive) of both statements. The three characters must match exactly for the program to restart. If no LABEL statement exists for a JUMP statement, no further calls will take

place. If no JUMP statement exists for a LABEL statement, then the latter is ignored. Note that a JUMP statement forces an end of set condition (see earlier in this section).

z/OS JCL Requirements for DAZZLER

The following table lists the data sets used by the Test utility, DAZZLER.

DDname	Medium	Description
CFILE	Reader	Control input for DAZZLER.
PRNTR	Printer	Report.

Example

```
//          EXEC PGM=DAZIFP,PARM='DLI,DAZZLER,psbname'
//STEPLIB DD DSN=ADL.LOAD,DISP=SHR
//          DD DSN=ADABAS.LOAD,DISP=SHR
//DDCARD  DD *
ADARUN PROGRAM=USER,...
//DAZOUT1 DD SYSOUT=X
//*
//*  DAZZLER DATASETS
//*
//PRNTR   DD SYSOUT=X
//CFILE   DD *
*HD GU           01
*HD GN           GB 01
/*
```

z/VSE JCS Requirements for DAZZLER

The following table lists the files used by the Test utility, DAZZLER.

DTF	Logical Unit	Medium	Description
CFILE	SYS007	Reader	Control input for DAZZLER.
PRNTR	SYS006	Printer	Report.

The control input for the batch monitor (DAZIFP) and for ADARUN is read from SYSIPT. Where the control input for DAZZLER itself is also to be read from SYSIPT (by assigning SYS007 to SYSIPT), the control statements must be specified in the following order:

```
DLI,DAZZLER,psbname,...          input for ↵
DAZIFP
/*
ADARUN DB=dbid,M0=MULTI,PROGRAM=USER,...  input for ADARUN
/*
*HD GU ....                       ↵
input for DAZZLER
.
.
/*
```

Example

```
// ASSGN SYS006,SYSLST
// ASSGN SYS007,SYSIPT
// EXEC DAZIFP
DLI,DAZZLER,psbname
/*
ADARUN PROGRAM=USER,...
/*
*HD GU          01
*HD GN          GB 01
/*
```

12

Managing ADL Files

▪ Overview	140
▪ Reorganization with ADL Utilities (Pseudo Mixed Mode)	140
▪ Reorganization with ADL Utilities (Normal Mode)	143
▪ Reorganization with Adabas Utilities	145
▪ Reorganization with User-written Programs	146
▪ Change of DL/I Segment Definitions	147
▪ Change of DL/I Field Definitions	148
▪ Change of the Data Layout	149
▪ Re-establishment of the Hierarchical Sequence	153
▪ Change of DL/I PSB Definitions	153
▪ Change of the ADL Structure	153
▪ Change of Adabas DBIDs and File Numbers	154
▪ Change of the Adabas File Layout	156

This chapter covers the following topics:

Overview

When a DL/I data base has been converted into ADL files, the need for modifications of these files may arise. The reason for this could be changes in the DL/I data base structure as well as changes in the Adabas files layout of the ADL files. Also, ADL files need to be maintained for reorganization, backup and recovery reasons in the same or similar way as normal Adabas files. During reorganization of an Adabas data base, ADL files may be assigned to Adabas DBIDs and file numbers different from those assigned when the files were created.

The first three sections in this chapter describe the procedures applicable when modifying ADL files, in particular the reorganization of ADL files.

- by ADL utilities,
- by Adabas utilities,
- or by user-written programs.

The remaining sections list potential changes and outlines which of the procedures should be used to perform the change. The following issues are explained in details:

- how to change the DL/I structures like the PSB, segment or field definitions;
- how to change the data layout of the I/O area;
- how to re-establish the hierarchical sequence of the migrated data;
- how to change the ADL structure by splitting a DBD over several files,
- how to change the Adabas structure like the DBID, file number, group or field definitions.

Reorganization with ADL Utilities (Pseudo Mixed Mode)

Some of the changes described later in this chapter require a logical unload and reload of the data. Logical unloads and reloads should be performed using the standard ADL Unload and Reload utilities. The procedure is similar to that used to convert the original DL/I data base (see the section ADL Conversion Utility in the *ADL Conversion* documentation).

At the conversion of the original DL/I data, ADL runs in “mixed mode”, i.e. DL/I calls ADL which calls the ADL Unload utility DAZUNDLI.

When the structure of the converted data changes, DAZUNDLI maps the old to the new structure. For this task it must be aware of both structures simultaneously. But the ADL directory supports only one version of a DBD. Therefore “pseudo mixed mode” was introduced to ADL. In the pseudo

mixed mode, DAZUNDLI accesses two ADL environments, containing two interface programs DAZIFP, two batch nuclei DAZNUCB and two ADL directories, one containing the old, the other containing the new structure.

In a first step, a copy of the (“standard”) ADL environment is created by copying the ADL batch interface program DAZIFP, the batch nucleus DAZNUCB, and the ADL directory. This ADL-copy-environment reflects the old structure; the standard ADL environment is used to convert the new structure. In “pseudo mixed mode”, the ADL-copy-environment calls the standard ADL environment which calls the ADL Unload utility DAZUNDLI.

The following steps must be performed:

Step	Description
Step 1	Copy ADL programs
Step 2	Create the ADL unload PSB
Step 3	Copy the ADL directory
Step 4	Modify and convert the DBD
Step 5	Create the ADL reload PSB
Step 6	Unload the data in pseudo-mixed mode
Step 7	Load the Adabas file(s)
Step 8	Re-establish logical relationships

These steps are explained in detail below. For more information about the ADL CBC utility see the section *ADL Conversion Utilities for DBDs and PSBs* in the *ADL Conversion* documentation.

Step 1: Copy ADL programs

In the ADL load library, make a copy of “DAZIFP” named “DAZIFPX”, and a copy of “DAZNUCB” named “DAZNUCX”.

Step 2: Create an ADL Unload PSB

Create a PSB containing two PCBs. Both PCBs should be based on the original DBD and should reference all segments. The first PCB will be used for reading the original data.

Run the PSB through the CBC utility.

Step 3: Copy the ADL directory

Copy the ADL directory file using standard Adabas utilities. This “copy-directory” contains the old structure including the Unload PSB and will not be changed. The source directory (“standard directory”) will be used for the new structure definitions.

Step 4: Modify and Convert the DBD

Modify the DBD sources according to your requirements.

Run the DBD through the CBC-utility. Ensure that the new structure is written into the standard directory.

Step 5: Create the ADL Reload PSB

Create a PSB containing two PCBs. Both PCBs should be based on the modified DBD and should reference all segments. The second PCB will be used for loading the modified data.

The Reload PSB must have the same name as the Unload PSB created in step 2. If both PSBs look the same, you can omit this step.

Run the PSB through the CBC utility. Ensure that the new structure is written into the standard directory.

Step 6: Unload the Data in Pseudo-Mixed Mode

Unload the data by running the ADL Unload utility, DAZUNDLI. As mentioned above, DAZUNDLI is executed in pseudo mixed mode.

DAZIFP is used to read the source data. The old structure definitions are taken from the copy-directory, i.e. it acts like DL/I in mixed mode. DAZIFP calls the application program DAZIFPX.

DAZIFPX writes the data with the new layout (as defined in the standard directory) to a sequential file.

The control statement must have the following layout:

```
EXEC DAZIFP
DLI,DAZIFPX,psbname,FNR=c-dir,...      input for DAZIFP
UNL,DAZUNDLI,psbname,FNR=s-dir,...     input for DAZIFPX
ADARUN DB=dbid,...                     input for ADARUN
MODE=CHECKNUM                          input for DAZUNDLI
```

where *psbname* is the name of the unload/reload PSB, *c-dir* is the file number of the copy directory with the old structure and *s-dir* is the file number of the standard directory with the new structure.

Under z/OS the input for DAZIFP is read with the PARM keyword, the input for DAZIFPX is read from DAZIN2, the ADARUN input is read from DDCARD, and the DAZUNDLI input from DAZIN1. Under z/VSE all control input statements are read from SYSIPT.

The unloaded data will be stored on a sequential file (DAZOUT3/DAZOT3D). See the section *ADL Conversion Utility* in the *ADL Conversion* documentation for more information on the DAZUNDLI utility.

Step 7: Load the Adabas File(s)

Each Adabas file used to store the converted data is loaded separately using the standard Adabas utilities, ADACMP and ADALOD. In other words, Step 7 needs to be run once for each of the Adabas files used to store the data.

Step 8: Re-Establish Logical Relationships

If the DBD is involved in logical relationships, the steps above must be performed for each DBD logical related. Finally, each logical child segment needs to be connected to its logical parent. This is done by performing Steps 8 and 9 of the section *ADL Data Conversion Utilities* in the *ADL Conversion* documentation.

Reorganization with ADL Utilities (Normal Mode)

If the new structure is the same as the old structure, the logical unload and reload of the data can be performed in "normal mode" using the standard ADL Unload and Reload utilities. The procedure is a simplified version of the pseudo mixed mode procedure described in the section above.

The copy of the ADL environment is not needed. DAZUNDLI is executed as a "normal mode" batch program (see *Normal Mode Batch Execution* in the section *Batch Installation and Operation*).

The following steps must be performed:

Step	Description
Step 1	Create the ADL unload PSB.
Step 2	Unload the data in normal mode.
Step 3	Load the Adabas file(s).
Step 4	Re-establish logical relationships.

These steps are explained in detail below. For more information about the ADL CBC utility see the section *ADL Conversion Utilities for DBDs and PSBs* in the *ADL Conversion* documentation.

Step 1: Create an ADL Unload PSB

Create a PSB containing two PCBs. Both PCBs should be based on the DBD and should reference all segments. The first PCB will be used for reading the data. The second PCB will be used for loading the data.

Run the PSB through the CBC utility.

Step 2: Unload the Data in Normal Mode

Unload the data by running the ADL Unload utility, DAZUNDLI. As mentioned above, DAZUNDLI is executed in normal mode. Note that this differs from the unloading of a native DL/I data base, which is executed in “mixed mode”.

The control statement must have the following layout:

```
EXEC DAZIFP
UNL,DAZUNDLI,psbname,...      input for DAZIFP
ADARUN DB=dbid,...           input for ADARUN
MODE=CHECKNUM                input for AZUNDLI
```

where *psbname* is the name of the unload PSB.

Under z/OS the input for DAZIFP is read with the PARM keyword, the ADARUN input is read from DDCARD, and the DAZUNDLI input from DAZIN1. Under z/VSE all control input statements are read from SYSIPT.

The unloaded data will be stored on a sequential file (DAZOUT3/DAZOT3D). See the section *ADL Data Conversion Utilities* in the *ADL Conversion* documentation for more information on the DAZUNDLI utility

Step 3: Load the Adabas File(s)

Each Adabas file used to store the converted data is loaded separately using the standard Adabas utilities, ADACMP and ADALOD. In other words, Step 3 needs to be run once for each of the Adabas files used to store the data.

The sequential file produced by Step 2 is taken as input for the ADACMP, as is the FDT and the user exit 6 generated at the conversion of the DBD

The process of loading the data using Adabas utilities is identical to the process described under Step 6 of the section *ADL Data Conversion Utilities* in the *ADL Conversion* documentation.

Step 4: Re-Establish Logical Relationships

If the DBD is involved in logical relationships, the steps above must be performed for each DBD logical related. Finally, each logical child segment needs to be connected to its logical parent. This is done by performing Steps 8 and 9 of the section ADL Data Conversion Utilities in the *ADL Conversion* documentation.

Reorganization with Adabas Utilities

You may use any Adabas utility for the maintenance of ADL files. But one important restriction must be considered: Never alter or implicitly change the ISN of an ADL record.

ADL uses the Adabas ISN to maintain the hierarchical structure of the data base as defined in the DL/I DBDs. To alter the ISN of a record would thus destroy the physical pointer to its child records.

The ISNs of records in ADL files must be preserved by specifying the option `USERISN=YES` for the Adabas utilities `ADACMP` (edit/compress /decompress data) and `ADALOD` (file loading).

Some of the changes described later in this section do not require a logical unload and reload of the data with ADL utilities. The Adabas utilities can be used instead, which are considerably faster. The following steps must be performed:

Step	Description
Step 1	Unload the data
Step 2	Convert the (changed) physical DBD
Step 3	Generate an user exit 6
Step 4	Load the data

These steps are explained in detail below.

Step 1: Unload the Data

Use the Adabas utility `ADACMP` with the `DECOMPRESS` function to unload and decompress the data of the ADL file. The Adabas utility is described in the *Adabas Utilities* documentation.

Step 2: Convert the (Changed) Physical DBD

Run the (changed) DBD against the ADL CBC-utility (see the section *ADL Conversion Utilities for DBDs and PSBs* in the *ADL Conversion* documentation). Use the original DBD name for the new structure. This step can be omitted, if there is no change in the DBD definition.

Step 3: Generate an User Exit 6

Write, assemble and link-edit an ADACMP user exit 6, which maps the data from the old structure to the new structure. Special care has to be taken for the ADL internal fields, which should remain untouched in most cases. You may use the member ADLEX06 (z/OS) or ADLEX06.A (z/VSE) in the ADL source library as an Assembler skeleton for an ADACMP user exit 6. For more information on the ADACMP user exit 6 see the *Adabas DBA Reference* documentation.

Step 4: Load the Data

The data is loaded into Adabas with the Adabas utilities ADACMP and ADALOD. The sequential file produced in Step 1 and the ADACMP cards generated by the CBC utility in Step 2 are used as input for ADACMP. This utility calls the user exit 6 from Step 3.

Note, that you can not use the standard ADL user exit DAZUEX06, if the data have been unloaded with Adabas utilities. And vice versa, if the data have been unloaded with ADL utilities like DAZUNDLI, the ADL user exit must be used rather than your own.

Reorganization with User-written Programs

Some specific changes in the database structure can not be handled by standard utilities. In general, you have to follow the same steps as described in the section *Reorganization with ADL Utilities (Pseudo Mixed Mode)* for details on unloading/reloading data. In this case the DAZUNDLI utility is replaced by your own unload program. The related unload PSB should contain two PCBs as described in the section mentioned above. The second PCB should specify the processing option 'L' (LOAD), and the ADL LOAD parameter should be set to 'UTILITY', so that the data is written to a sequential file. The insert call which writes the data to the sequential file, must supply the data in the new hierarchical sequence. The SSA for the call must be unqualified and should only refer to the segment name corresponding to the data.

Change of DL/I Segment Definitions

ADL does not store any information about the child segments in the respective parent data. Therefore, the addition or deletion of dependent segment types does not effect the parent segments. But, since the segment number is part of the ADL internal pointer filed, all segments following in the hierarchy are effected when a segment type is inserted or deleted.

Adding a new Segment Type

When a new segment type is added to the end of the DBD, no other segment is affected.

The following steps are required:

- Add the new segment definition to the DBD source and run it through the ADL CBC utility. If the data of the new segment should reside on a separate Adabas file, add a `GENSEG` statement to the input cards of the CBC utility.
- Modify the Adabas file definitions. If the data is on a separate file, use the `ADACMP` and `ADALOD` utilities to allocate an empty Adabas file. Otherwise, add the new group/field definitions to the existing `FDT`, for example with the Adabas Online System (AOS).

When the new segment is not the last segment in the DBD a logical unload and reload of the data is required. This will supply the ADL internal pointers with the new segment numbers. Follow the steps described in the section *Reorganization with ADL Utilities (Pseudo Mixed Mode)* earlier in this documentation.

Deleting a Segment Type

If an entire segment type is to be deleted, the procedure to follow depends on the hierarchical position of the segment. Additionally, if data is stored for this segment, the related storage space has to be released.

If the segment is the last segment of the DBD and the data is stored on a separate Adabas file, follow these steps:

- Delete the segment definition from the DBD source and run it through the ADL CBC utility. Take care that no `GENSEG` statement refers to the deleted segment.
- Delete the Adabas file related to the segment with the `DELETE` function of the Adabas utility.

If the segment is the last segment of the DBD and no data is stored for this segment, it is sufficient to run the modified DBD definition through the ADL CBC utility. The group and field definitions related to the segment remain in the Adabas `FDT`, but because of the Adabas compression there is no storage possessed by that field. It is sufficient to eliminate these fields during the next reorganization.

Another situation is given, if the segment to be deleted is supplied with data and if it is stored with data of other segments on one Adabas file. If the segment is the last one in the DBD, follow the steps described in the section *Reorganization with Adabas Utilities*. If it is not the last one, a logical unload and reload is required as described in the section *Reorganization with ADL Utilities (Pseudo Mixed Mode)*. Leave out the corresponding sensitive segment from the first PCB of the unload PSB.

Moving a Segment Type

Moving a segment type to another hierarchical position is a special task which can not be handled by standard utilities. Refer to the section *Reorganization with User-written Programs* for details.

Increasing the Segment Length

If you want to increase the overall length of a segment, the modified DBD definition must run through the ADL CBC utility. Depending on the last Adabas field in the group which corresponds to the segment, further action may be necessary:

- If the length of this field can be increased so that the entire new segment is covered by the whole group, this should be done using the Adabas Online System (AOS). The maximum length of an Adabas field is restricted to 253 bytes. Note, that the FDT generated by the ADL CBC can differ from the one defined by AOS. But this is no problem as long as the overall group length is the same as the segment length.
- If the new length of the last Adabas field would exceed 253 bytes, you have to follow the steps described in the section *Reorganization with Adabas Utilities*. In this case you can use the FDT generated by the ADL CBC utility to reload the data.

Change of DL/I Field Definitions

DL/I corresponds with the application on a segment level. Field definitions affect the application only for sequence fields, sensitive fields and fields which are referred to in secondary indices. All other 'normal' field definitions are meaningless to the application. This means, the application can work with copy codes which are totally different from the layout defined by the DBD definition. Therefore, these 'normal' DL/I field definitions can be changed without the need to modify your application, as long as they still fit into the segment.

ADL uses the DL/I field definitions to generate the initial layout of the corresponding Adabas file. See the section *Conversion of the Data Structure - General Considerations* in the *ADL Conversion* documentation for more information on the generated Adabas file layout. As with DL/I, the DL/I field definitions do not affect the DL/I application, when it runs against ADL. But it concerns the Adabas file layout in that way that the Adabas compression and the Natural programs which access the migrated data, as described in the section mentioned above.

Therefore, the DL/I field definitions should be checked carefully before the conversion or at reorganization time.

If you want to change the DL/I field definitions after the migration, you should follow the steps outlined in the section *Reorganization with Adabas Utilities* earlier in this section. There is no need for an user exit 6, as long as the data layout remains the same. If the data layout should be modified too, refer the next section. All these reorganization steps are only required, if the Adabas file layout should match the DL/I definitions. Since there is no need for this correspondence, you may modify the DL/I definitions only using the ADL CBC utility or on the other hand you may modify the Adabas file layout only, as described later.

Adding Key Fields and Logical Relationships

If you want to define a new sequence field or a new secondary index, you must follow the steps outlined in the section *Reorganization with ADL Utilities (Pseudo Mixed Mode)* earlier in this documentation. The same is true, if you want to add a new logical relationship to the DBD definitions.

Change of the Data Layout

The data layout of a segment is usually described in the copy code of the I/O area in the applications. A *data field* is a part of the I/O area and is of a specific meaning to the application. It can match a DL/I field definition but as described earlier there is no need for this.

If changes have been made to the data layout and this new structure is to be reflected in the ADL directory (for the Natural access or for the ADL Online System) the modified DBD definitions have to be run through the ADL Conversion Utilities as described in the section *ADL Conversion Utilities for DBDs and PSBs*. This step is required if the overall segment length has been changed or sequence or secondary index fields have been moved or modified. In all other cases this step is optional.

Changing the Length of a Data Field

If you want to increase the size of a data field, the corresponding Adabas file description is important. There is one special case to be regarded:

- The end of a data field coincides with the end of an Adabas field and the increased length of this Adabas field does not exceed 253 bytes. In this case use the Adabas Online System to increase the length of that Adabas field.

In all other cases follow the steps outlined in the section *Reorganization with Adabas Utilities* earlier in this section.

If you want to decrease the size of a data field there is again one special case to be regarded:

- The end of a data field coincides with the end of an Adabas field, the length of the decreased field is greater than zero, and the part to be omitted contains all null values. In this case use the Adabas Online System to decrease the length of that Adabas field.

In all other cases follow the steps outlined in the section *Reorganization with Adabas Utilities* earlier in this section.

Changing the length of data fields can affect the overall segment length and the start position of all following fields. Therefore, check whether the DBD definitions have to be modified as well, as described at the beginning of this section.

Changing the Length of a Sequence Field

If you want to change the length of a **root sequence field**, perform the following steps:

- Modify the DBD and run it through the ADL CBC utility.
- If the DBD is involved in logical relationships, run all connected DBDs through the ADL CBC utility.
- Use AOS to increase the length of all modified fields (compare the old ADACMP cards with the new ones). These fields are the root sequence field (usually "AA") and the corresponding concatenated key fields (usually "PA" or "QA") in all involved Adabas files.
- If the field is involved in secondary indices, you must unload and reload the data with ADL Utilities. See the section *Reorganization with ADL Utilities (Normal Mode)*. If the data is not involved in secondary indices, unload and reload of the data is not required.

If you want to change the length of a **dependent sequence field**, perform the same steps as for a root sequence field. But you must always unload and reload the data with ADL Utilities to build up the new internal pointer field (Z1).

Adding and Deleting Data Fields

A new data field can be added by increasing the length of the previous data field as described above. Note, that in case you do not need to reorganize the data with Adabas utilities, the ADACMP cards generated by an ADL CBC run of the modified DBD does not match the current used FDT.

Deleting a data field can be reached by decreasing the size of the data field to the length of null. This is also outlined in the previous section.

Changing the Data Position

If a data field should be moved to another position in the I/O area the procedure described in section *Reorganization with Adabas Utilities* must be followed. During this reorganization other modifications can be performed like changing the field or segment lengths, add or delete fields, etc. as described above.

Examples

Increase the length of a field

In the Instructor database 'INSTDB' there is a segment ADDRESS which describes the address of the instructor. The segment was originally defined in the following way:

```
SEGM  NAME=ADDRESS ,PARENT=INSTRUCT ,BYTES=60
FIELD  NAME=ZIPCODE ,BYTES=4 ,START=1
FIELD  NAME=CITY ,BYTES=16 ,START=5
FIELD  NAME=STREET ,BYTES=40 ,START=21
```

The Adabas group and fields corresponding to this segment and its elements are named SE, AG, AH and AI respectively. The length of the Adabas fields are the same as those of the corresponding DL/I fields.

Now the zip code should be increased from 4 to 5 bytes and the city field from 16 to 20 bytes.

In a first step you have to modify the DBD source:

```
SEGM  NAME=ADDRESS ,PARENT=INSTRUCT ,BYTES=65
FIELD  NAME=ZIPCODE ,BYTES=5 ,START=1
FIELD  NAME=CITY ,BYTES=20 ,START=6
FIELD  NAME=STREET ,BYTES=40 ,START=26
```

Run the modified DBD against the ADL CBC utility as described in the section *ADL Conversion Utilities for DBDs and PSBs* in the *ADL Conversion* documentation . This ensures that the new segment length is known to ADL. Use the Adabas Online System to increase the length of the Adabas fields AG and AK from 4 to 5 bytes and from 16 to 20 bytes, respectively. From the view of the database management the work is finished. Of course, now the applications have to be modified to reflect the new layout of the I/O area for the ADDRESS segment. Depending on the task it can be required to edit the data, for example in case of zip code data moving from 4 digits to 5 digits.

Adding a new field

The new field NUMBER containing the house number has to be added to the ADDRESS segment. The field should be 6 bytes long and be of type character. The easiest way is to increase the STREET field from 40 to 46 bytes. The application can use these additional 6 bytes as the house number field.

The new DBD definition looks like this:

```
SEGM  NAME=ADDRESS , PARENT=INSTRUCT , BYTES=71
FIELD  NAME=ZIPCODE , BYTES=5 , START=1
FIELD  NAME=CITY , BYTES=20 , START=6
FIELD  NAME=STREET , BYTES=46 , START=26
```

Run the new definition against the ADL CBC utility. Use the Adabas Online System to increase the length of the Adabas field AI from 40 to 46 bytes. Note, that the new part of the I/O area (field NUMBER) is initially filled with blanks.

Move fields

The field STREET (including the NUMBER data field) of the example above should be moved to the start of the ADDRESS segment. The new DBD definition should look like:

```
SEGM  NAME=ADDRESS , PARENT=INSTRUCT , BYTES=71
FIELD  NAME=NUMBER , BYTES=6 , START=1
FIELD  NAME=STREET , BYTES=40 , START=7
FIELD  NAME=ZIPCODE , BYTES=5 , START=47
FIELD  NAME= CITY , BYTES=20 , START=52
```

First unload and decompress the corresponding Adabas file, then run the modified DBD definition against the ADL CBC utility. To generate an user exit 6 you may use the Assembler skeleton ADLEX06 from the ADL source library. Make sure that all fields which should not be reorganized are moved unchanged from the input to the output record. Assuming that register 5 points to the start of SE group (segment ADDRESS) in the input record and register 6 to the same in the output record, the following Assembler statements will reorganize the fields:

```
MVC  0(6,6) , 65(5)      MOVE NUMBER
MVC  6(40,6) , 25(5)     MOVE STREET
MVC  46(5,6) , 0(5)      MOVE ZIPCODE
MVC  51(20,6) , 5(5)     MOVE CITY
```

Assemble and link-edit the user exit. Then load the data back into Adabas with the ADACMP and ADALOD utilities. The ADACMP utility must use the newly created user exit and the new ADACMP cards produced by the ADL CBC utility.

Re-establishment of the Hierarchical Sequence

At an initial load the migrated data is stored in Adabas in the hierarchical sequence. Continuous insertion and deletion of data distorts this sequential order and eventually may decrease the access performance as described in section *Performance Considerations* in this documentation. Therefore, it may be advantageous to re-establish the hierarchical sequence. This can be achieved by following the steps outlined in the section *Reorganization with ADL Utilities (Normal Mode)* earlier in this documentation.

Change of DL/I PSB Definitions

If any changes have been made to the DL/I PSB definitions, the modified PSB must be run against the ADL CBC utility as described in the section *ADL Conversion Utilities for DBDs and PSBs* in the *ADL Conversion* documentation.

Ensure that the old definition are deleted from the directory before converting the new one.

The following PSB definitions can be modified: the processing option, the key length, the positioning, or the processing sequence. In addition, PCBs, sensitive segments or sensitive fields can be added or deleted to/from the PSB. After adding new elements to a PSB which is contained in the ADL CICS table DAZPSB, the ADL CICS tables DAZDBD and DAZBUF must be regenerated as described in the section *CICS Installation and Operation* in this documentation.

Change of the ADL Structure

At DBD conversion time it is defined on which Adabas files the segments should be stored. Later, it may become necessary to change these definitions. If you want to move or copy the ADL files to a new DBID/FNR combination, refer to the section *Change of Adabas DBIDs and File Numbers* later in this section.

Splitting a DBD

If the segments of a DBD should be distributed over several Adabas files, the data must be unloaded and reloaded as described in the section *Reorganization with ADL Utilities (Pseudo Mixed Mode)* earlier in this documentation. Although the DBD definition did not change, it is necessary to generate a new DBD reflecting the new structure. Use the GENSEG keyword for the ADL CBC utility to specify, on which files the segments should be stored.

Change of Adabas DBIDs and File Numbers

The DBID and file number assigned to the segment types of a database during the DBD conversion process is stored in the ADL directory. If you want to assign to an ADL file an Adabas DBID or file number different from the one defined at database conversion time, you must run the DBD against the ADL Control Block Conversion with the new DBID / FNR combination.

If the DBID used to store the data is the same as the DBID of the ADL directory, it is recommended to omit the DBID parameter with the GENDBD function. See the example [Creating a mirror database](#) later in this section for details.

At the ADL Control Block Conversion a logical ID is defined for the DBD. This logical ID is used by ADL as a part of the physical pointers reflecting the hierarchical structure of the data base. Related segments are found based on the values stored in the physical pointers. If you want to assign to an ADL file an Adabas DBID or file number different from the one defined at data base conversion time, you must specify the same logical ID as with the original conversion.

For synchronization reasons it is important that all DBIDs referenced by all PCBs of a PSB are identical. If ADL detects that this is not the case, it will abnormally terminate the application with the error code "ADL0329".

Examples

Moving an ADL file within an ADL environment

Assume a DL/I database named "ADLDBD1" has been assigned to Adabas DBID 2 and file number 96 and was converted with LOGID=1. After reorganization of the system this file should be assigned to file number 10 on the same DBID. You would have to code the following statement at the control block conversion:

```
GENDBD NAME=ADLDBD1,DBID=2,FNR=10,LOGID=1
```

If the ADL directory is also on DBID=2, you can omit the DBID keyword (recommended). Note that LOGID=1 is the default value. Therefore you can shorten the statement as follows:

```
GENDBD NAME=ADLDBD1,FNR=10
```

Copying an ADL file within an ADL environment

Assume a DL/I database named "ADLDBD2A" has been assigned to Adabas DBID 2 and file number 20 and was converted with LOGID=1.

```
GENDBD NAME=ADLDBD2A,DBID=2,FNR=20,LOGID=1
```

Later the data has been unloaded and reloaded to file number 21 with Adabas utilities. Now it is desired to access both files with one DL/I or with one Natural application through the ADL CALLDLI or ADL Consistency Interface, respectively.

In order to do this, perform the following steps:

- Copy the DBD source and use a new DBD name, e.g. “ADLDBD2B”.
- Run the DBD definitions of “ADLDBD2B” through the ADL Control Block Conversion. Specify the new DBID/FNR combination. Use the same LOGID as with the original DBD and store it on the same ADL directory.

```
GENDBD NAME=ADLDBD2B,DBID=2,FNR=21,LOGID=1
```

Copying an ADL file to another ADL environment

Assume a DL/I database named “ADLDBD3” has been assigned to Adabas DBID 2, file number 11 and LOGID=1.

```
GENDBD NAME=ADLDBD3,DBID=2,FNR=11,LOGID=1
```

A new ADL environment has been created with its own ADL directory file and ADL nucleus modules. The data of “ADLDBD3” has been copied to DBID 3 and file number 22 with Adabas utilities. To access this data in the new environment perform the following steps:

- Run the DBD definitions of “ADLDBD3” against the ADL Control Block Conversion. Store the definitions on the new directory file. Use the same LOGID as in the original DBD. You can use the same DBD name. Specify the new DBID and FNR at the GENDBD statement.

```
GENDBD NAME=ADLDBD3,DBID=3,FNR=22,LOGID=1
```

Creating a mirror database

Assume the ADL directory file and all the ADL files, which are referred in this directory, are stored on DBID 1. Moreover at the conversion the DBID parameter was never specified for the GENDBD/GENSEG function.

To create a mirror database on DBID 2, unload the ADL directory file and all ADL files with Adabas utilities from DBID 1 and reloaded them to DBID 2, assigning the same file numbers as before. Take care that the Adabas utilities preserve the User ISNs.

There is no need to run any of the DBD definitions against the ADL Control Block Conversion again. If no DBID is specified in the ADL directory for an ADL data file, ADL assumes that the data uses the same DBID as the directory.

You can create a new ADL batch nucleus with the new DBID specified in the ADL parameter module to access the new environment in batch. But you can also use the same ADL batch nucleus for both environments. When you start a batch program, you just have to specify the DBID of the ADL directory file as a dynamic parameter for DAZIFP as described in the section *Batch Installation and Operation* in this documentation. Thereupon ADL will treat this DBID as the physical DBID of all ADL files.

For CICS you have to create a new ADL CICS nucleus specifying the DBID of the new directory file with the DAZPARM macro.

Root-only databases

A root-only database is a database, which does not have any dependent segment type. Therefore the LOGID parameter which is stored with dependent segments is meaningless. When you copy or move a root database follow the rules described above for normal databases.

Change of the Adabas File Layout

All modifications described in this section concern the Adabas side, only. Changes which affect the DL/I side as well are described in the previous sections.

The need to change the Adabas field layout of ADL files may originate from requirements of new Natural or Adabas applications. Also, to obtain an increase in the Adabas compression rate or performance might be a reason for modifications. You may change the Adabas layout of an ADL file after the data base conversion within certain limitations.

Changing Adabas Group Definitions

The overall length of the Adabas groups generated for each segment type must not be altered. A group length can only be altered as a result of a changed DL/I segment length as described earlier.

The Adabas group name corresponding to a DL/I segment is defined at the DBD conversion. If the group name should be altered, follow the steps outlined in the section *Reorganization with Adabas Utilities* earlier in this section. Use the ADANAME keyword of the GENSEG function when running the ADL CBC utility to specify the new group name. Note that the change of one group name can affect the group names of the following segments. There is no need for a user exit 6 at the ADACMP run.

Changing Adabas Field Definitions

The following rules apply when attempting to alter the Adabas field definitions:

You must not alter

- the ADL system fields (ZO .. Z8),
- the ADL generated PCK field and the descriptors corresponding to secondary index fields (both are identified by a clear comment in the ADL generated input source for ADACMP),
- the Adabas field corresponding to a sequence field.

You can alter

- all field definitions inside the Adabas groups corresponding to segments, provided the overall group length remains the same. This regards the number of the fields inside the group, the length and the format of each field. Note, however, that ADL is not aware of the types of these Adabas fields and thus errors due to invalid data supplied for numerical fields can occur. Follow the steps outlined in the section *Reorganization with Adabas Utilities* earlier in this section. There is no need for an user exit 6 (step 3), if the data layout has not been changed and as long as the DBD has not been touched, step 2 can also be omitted. After unloading the data, modify the ADACMP cards manually as required.

You can add

- any new Adabas field definitions outside of the Adabas group corresponding to DL/I segments,
- any definitions of descriptors, superdescriptors, etc., regardless whether the related fields have been generated by ADL or not. If a descriptor is defined as `UNIQUE` in ADL files, be aware that your DL/I ISRT calls may receive the status 'II' (segment already exists). This happens because the CALLDLI Interface automatically transforms an Adabas response code 198 into a DL/I status code 'II'.



Important: Please note that new Adabas fields defined outside the DL/I segment (group) are not unloaded at the reorganization with ADL utilities.

13 Performance Considerations

■ Introduction	160
■ Data Processing and Pointers	161
■ Initial Load and Reload the Database	163
■ Using the Adabas Multifetch Feature	164
■ Insert and Delete	166
■ Splitting a DBD	167
■ Field Definitions	167
■ Enqueue Logic	168
■ CICS/Batch Communication	168
■ Application Program and DB Design	168

This chapter covers the following topics:

Introduction

The performance of DL/I applications depends on various factors:

- The environment: The hardware, how the data is physically stored, the operating system, the TP monitor, the number of users on the system, the number of users accessing the database management system, whether there are batch jobs running at the same time as online applications and more.
- The database structure: The *ACCESS* method used, the number of hierarchical levels, the complexity of the structure including logical relationships and secondary indices, the distribution of the database over different Adabas files and more.
- The application: The way it retrieves the data: direct (GU calls) or sequential (GN, GNP calls), the amount of data for insert, update or delete requests, the complexity of the SSAs, the fields used for qualifying the call and more.

Further on, there are some side-effects related to the performance which should also be considered:

- The disk space required to store the data.
- How often the database has to be reorganized.
- How flexible the database system is, that is, how much effort it takes to change the layout of the database.
- How the database system affects the loading of the TP monitor.
- What tools are available to debug performance-critical applications.

When the DL/I application runs against migrated data, two Software AG products are involved: ADL and Adabas. For the tuning of Adabas, you should refer to the appropriate Adabas documentations, such as, the *Adabas DBA Reference* documentation. It helps you to optimize your physical database layout (disk usage), the buffer pool management, the usage of Multifetch and more. You may use Adabas facilities, such as Review, Adabas Statistics Facility and AOS/Adabas Manager to debug performance-critical applications. You should always keep in mind that, from the Adabas point of view, your DL/I application is like any other application making Adabas direct calls.

The current section explains the tuning of ADL. It describes in detail how the performance is affected by

- direct and sequential data processing,
- the layout of the ADL internal pointer fields,
- unload and reload the data with ADL utilities,

- initial load of HDAM databases,
- Adabas Multifetch (Prefetch) feature,
- the Multifetch table (MFT parameter) and the Record Buffer Extension (RBE parameter),
- the last-call save area (LCS parameter),
- mass insert by user application,
- deletion of data,
- splitting the segments of a DBD over several Adabas files,
- Adabas Fastpath,
- the field definitions,
- the enqueue logic (ET parameter),
- the communication between Adabas and the CICS or batch region,
- the way your application accesses the data.

Data Processing and Pointers

Three DL/I commands are available to retrieve data:

- Get Unique (“GU”) for direct access of specific data,
- Get Next (“GN”) for sequential read of the whole database,
- Get Next in Parent (“GNP”) for sequential read of a part of the hierarchy.

For the sequential read commands, DL/I supplies pointers which reflect the sequence of the processing:

- The hierarchic forward pointer which points to the hierarchical “next” segment, that is, the segment which will be retrieved at the next unqualified “GN” command.
- The physical child first and the physical twin forward pointers. These pointers are used when processing a “GNP” command.

For the direct access of data, DL/I treats the root segment differently from the others.

- The root segment is handled by the appropriate ACCESS method, for example, VSAM if this was specified at the DBD generation.
- Dependent segment data is searched by running through the pointers, that is, for each level, the physical child first pointer is used to get the first occurrence of the segment type wanted, and then the physical twin forward pointer is repeatedly used until the specified data occurrence is reached.

In order to reflect the hierarchical structure, ADL mainly uses one field, the ADL physical pointer field Z1 which is defined as a unique Adabas descriptor. In fact it is a “descriptor” rather than a “pointer” because it describes its own position instead of pointing to other data. In addition to the Z1 field, the root sequence field is defined as a descriptor too (the “root descriptor”).

The layout of the ADL physical looks like:

Field	Parent Logical ID	Segment Number	Parent ISN	Segment Sequence Field	Appendage
Length	1	1	4	variable	variable

The Logical ID and segment number are each 1 byte long, the Parent ISN is 4 bytes long, the length of the sequence field is defined by the DBD definition and the appendage has a variable length. If the segment sequence field is not unique or does not exist at all, the appendage is generated to make the Z1 field unique. The Z1 field can be up to 126 bytes long. To minimize the space requirements, an appendage as short as possible is used.



Note: The layout of the ADL physical pointer field has been changed with ADL 2.3. The former DBID (1 byte) and FNR (1 byte) have been replaced by the Logical ID (1 byte); and the parent ISN has been increased from 3 bytes to 4 bytes. ADL 2.3 uses only one layout of the ADL physical pointer where the segment number is in front of the parent ISN corresponding to the ADL 2.2 “SEQ=SEG” layout.

For the direct access of data, ADL treats the root segment differently from the others (like DL/I):

- On the root level, ADL makes use of the root descriptor.
- For dependent segments, the Z1 field is used.

For each level, an L3 or L9 call is issued, depending whether the data is wanted (for path call or lowest level) or not. The start value for these calls is built up for dependent segments from the already known Logical ID and ISN of the parent and from the requested segment number. If the SSA searches for a specific sequence field value (with “greater” or “equal”), this value is included in the start value too. In this case, the value on that level can be accessed directly; no run through a pointer queue is required.

Sequential reading of the root segment data is handled by sequential reading of the root descriptor. Sequential reading through the hierarchy cannot be translated into one Adabas sequence. For each (sensitive) segment, there is one Adabas sequence (one command ID). To read the first occurrence in a twin chain, ADL makes a value start call, where the sequence field part of the Z1 field in the value buffer is supplied with a minimal value.

To retrieve the next occurrence in the twin chain, ADL continues reading in the Adabas sequence. Since there is no information which says how many records are in the twin chain, ADL has to check whether the retrieved record belongs to the current twin chain. This is done by checking whether the Logical ID, segment number and ISN match those requested. This has the effect that ADL reads one record too many at the end of a twin chain. This additionally-read record is used by the last-call save area as described later.

Initial Load and Reload the Database

The data is initially loaded to the ADL files with the Adabas utilities `ADACMP` and `ADALOD`. The physical sequence in which it is loaded is defined by the unload utility, usually `DAZUNDLI` (see the section *ADL Data Conversion Utilities* in the *ADL Conversion* documentation for more information). This utility reads the data in the hierarchical sequence, that is, for each root, it reads all dependent segments before it accesses the next root. Therefore, the loaded data is in the hierarchical sequence too. This means that, if a root segment is on one data block, then the dependents will be on the same block, the next root also, and so on, depending on how many records can be stored on one data block.

An application which reads the data in the hierarchical sequence can therefore issue many DL/I calls before the next physical I/O is required. This is, of course, only true as long as the data is in the hierarchical sequence. When new data is inserted and old data deleted, it comes more and more out of that sequence. If you notice a considerable decrease in performance, you should think about restoring the database. Restoring with Adabas utilities makes no sense, because Adabas has no idea about the hierarchy which is inherent in your data. The easiest way is similar to the initial load: unload the database with `DAZUNDLI` and reload it with `ADACMP` and `ADALOD`. In this case, `DAZUNDLI` runs as a normal (that is, not mixed mode) application, as described in the section *Managing ADL Files* in this documentation. This brings the data back to the hierarchical sequence.

Additional considerations should be made when initially loading a randomized database, such as, an HDAM database. Root segments are retrieved by DL/I in the randomized sequence. When reading with ADL, they are accessed in the sequence of the root sequence field. If these two sequences do not match, you may notice poor performance for sequential reads. In this case, you should load the data in the root key sequence, as described in the section *Unloading a HDAM Database* in the *ADL Conversion* documentation.

The unload utility, `DAZUNDLI` (or `DAZREFOR`), also assigns the Adabas ISNs which are used when the data is loaded into the Adabas file. Therefore, after an initial load or restoring of the data, the ISNs reflect the hierarchical sequence too. The sequence of the ISNs is only important when the last-call save area (LCS) is used (described below). This feature works best when the ISNs are in the hierarchical sequence. Thus, if the LCS is used, restoring of the data with ADL utilities should be considered if the ISNs are out of the hierarchical sequence for the most part.

Using the Adabas Multifetch Feature

The Adabas Multifetch feature reduces the communication overhead between the application program and the Adabas nucleus for sequential reads. See the *Adabas Operations* documentation for more information. The Multifetch feature is similar to the Prefetch feature. If not otherwise stated, we use Multifetch as a synonym for Multifetch and Prefetch.

The Multifetch feature is activated for batch applications by specifying the `ADARUN PREFETCH=YES` parameter. The number of sequences (command IDs) which are multifetched is defined by the keywords `PREFTBL` (total buffer length) and `PREFSBL` (single buffer length), exactly `PREFTBL` divided by `PREFSBL`. This number has to be evaluated very carefully. Multifetching “bad” sequences can considerably decrease the performance. For example, an L3 sequence with many value-restarts (“V” option) is bad, because Multifetch throws away the data of the corresponding buffer at every restart. The Multifetch buffers are occupied by the sequences which are used first, that is, the first CID uses the first buffer, the second the next one, and so on. You can exclude specific command/file number combinations from Multifetch by specifying the `PREFXCMD` and `PREFXFIL` keywords. Note that an Adabas OP (open) command refreshes all the buffers.

The ADL parameters `RBE` and `MFT` can be used to restrict the number of multifetched records for a specific `PCB/SENSE` combination. (See the section *ADL Parameter Module* in the *ADL Installation* documentation for a detailed description.) Here we have to distinguish between Multifetch and Prefetch. The `MFT` can only be used with the Multifetch feature and, in this case, you are recommended to use it instead of the `RBE`.

With the Multifetch Table (MFT), you can explicitly specify the number of multifetched records. For an unspecified `PCB/SENSE` combination, the maximum number of records will be multifetched. If you want to multifetch not the maximum number but a smaller amount, you should specify an MFT entry for this `PCB/SENSE` combination. Because of the double buffering of Multifetch, Adabas will return twice as many records as you specify. To minimize the amount of returned data for a specific sequence you should specify a value “1”, which will result in 2 returned records.

With the Record Buffer Extension (`RBE`) parameter, you can increase the record- buffer length for specific `PCB/SENSE` combinations. This has the effect that fewer records fit into the the ISN buffer for the corresponding Prefetch call. Thus, the fewer records you want to prefetch, the bigger the corresponding `RBE` entry should be. To retrieve the maximum number of records for a specific `PCB/SENSE` combination, you should omit the corresponding `RBE` entry.

The use of Multifetch for migrated data requires knowledge of the application and of how ADL translates the DL/I calls into Adabas calls. With the ADL trace facility, you can obtain this information (see the section *Debugging Aids - ADL Trace Facility*. It lists you the DL/I calls of the application and the resulting Adabas calls. From this list, you can see, for example, that a GU call is translated into an Adabas “L3” with value-start. Therefore the corresponding CIDs are bad candidates for Multifetch. On the other hand, if a root segment is read sequentially (GN calls), there is one value-start at the first call and then no other restart. Therefore, you can multifetch this sequence.

With the last-call save area (LCS), the record last retrieved is saved. This feature can be activated by specifying the LCS parameter, as described in the section *ADL Parameter Module* in the *ADL Installation* documentation. If an application program reads the same record twice or more, only one Adabas call will be issued by ADL because it finds this value in the LCS. But the main importance of the LCS is when reading sequentially through dependent segments.

As described earlier, ADL makes a value-start call at the beginning of each twin chain. Then it reads sequentially through the chain until it finds one record that does not belong to the chain. Now, under certain circumstances, it can happen that this additionally read record is the first record of the next twin chain of the same segment type. In this case, the LCS preserves ADL for reading the same record twice. Since this saved call was the call with the value-start, a sequential read (GN or GNP) through a dependent segment type is translated into sequential Adabas reads without intermediate restarts. Therefore, you can use the Multifetch feature for dependent segments too. The remaining questions are: which record does ADL find after the end of a twin chain and under which condition is this record the first of the next twin chain for the same segment type? The sequence in which ADL retrieves the data is determined by the layout of the ADL physical pointer fields, which is described earlier in this section.

The Z1 physical pointer field sorts the data in the sequence “segment number / parent ISN”. In general, the record found after the end of a twin chain is from the same segment type with the next higher parent ISN. If this ISN belongs to the next parent, the record found is the first of the next twin chain. Therefore, the ISNs should match the hierarchical sequence. This is true after an initial load of the data or after a restore. If the ISNs are mainly out of the hierarchical sequence, you should reestablish the hierarchical sequence as described in the section [Managing ADL Files](#) in this manual. If you do not want to reestablish the hierarchical sequence by any reason, adjust the number of multifetched records with the MFT or RBE parameter or avoid multifetching those sequences at all.

Summary

- Do not use Multifetch for direct reading (“GU”).
- Use Multifetch for sequential reading of root-segment data.
- Use the last-call save area (LCS) and Multifetch for sequential reading of dependent data.
- Do not use Multifetch for sequential reading of dependent data if you do not use the last-call save area (LCS).
- Adjust the number of multifetched/prefetched records with the MFT or RBE parameter, respectively.
- Check with the ADL trace facility or with Adabas utilities whether the data and ISNs are still in the hierarchical sequence. If required, restore the database with ADL utilities, so that the ISNs match the hierarchical sequence again.

Insert and Delete

A DL/I insert request ("ISRT") is translated by ADL into an Adabas N1 call. A special case is given when loading the database. For the initial load during the conversion process, the data which have been unloaded by ADL utilities are loaded into Adabas with the Adabas utilities ADACMP and ADALOD. ADACMP requires the ADL user exit DAZUEX06.

Another way of initially loading data is using insert calls by a user application against a PCB with PROCOPT=L. This insert call must be qualified with one SSA which specifies the segment to be inserted but no other qualification. The data must be inserted in the hierarchical sequence. How such an insert call is treated by ADL depends on the setting of the LOAD parameter.

If LOAD=DIRECT is specified, an Adabas N1 call is issued, as with any "normal" insert against a PCB with PROCOPT=I.

If LOAD=UTILITY is specified, the data is written to a sequential file. It has the same layout as the data generated by the ADL data conversion utilities. It is loaded into Adabas with ADACMP (with the ADL user exit) and ADALOD.

Which of these two methods should be used depends on the amount of data to be loaded. If it is only a small amount of data, the easiest way is to load it directly into Adabas. If, however, there is a lot of data to be loaded, you are recommended to use the Adabas utilities, because the utilities are considerably faster than a direct insert.

A DL/I delete request ("DLET") is translated by ADL into Adabas E1 calls for the specified segment occurrence and all its dependents. Normally, DL/I does not really delete the data; it sets a flag which indicates that the record is "deleted". When retrieving the data, DL/I has to run through such "deleted" data, because the pointers are still there. Therefore, the DL/I database has to be reorganized to get rid of that garbage and to release the corresponding storage.

When Adabas deletes a record, the storage is released immediately. This takes more time than just setting a flag and you may notice an increased time, especially for mass deletes by user applications. On the other hand, there is no running through "deleted" data and no requirement for a reorganization to release the storage.

Splitting a DBD

With the GENSEG statement of the ADL CBC utility, you can distribute the segments of a DBD to multiple Adabas files.

The following reasons for splitting a DBD are related to performance.

- If the application accesses some segment types and not others, splitting will reduce the number of physical I/Os. This is because more of the accessed data fits on one data block, since the information not requested is no longer stored with it.
- If some of the segments contain constant data like tables, which are often read but not updated, you can use Adabas Fastpath especially for these files.
- For some requests, you can use Adabas utilities if the segment is separated. For example, if you want to delete all the data of a dependent segment type, you can refresh the corresponding Adabas file.

But splitting a DBD can also have disadvantages:

- The initial load will consume more time, because the ADL user exit always has to process the whole DBD data.
- If your application accesses segments that reside on different Adabas files, it probably requires more physical I/Os. If it reads, for example, one root segment occurrence and all its dependents, at least one I/O is performed for each Adabas file. As long as your application reads sequentially through the data, this will not lead to any problem. This is because Adabas will find the subsequent data in the buffer pool and therefore, the total number of I/Os will not increase. If, however, it reads the data out of sequence, the number of I/Os is multiplied by the number of Adabas files accessed.

Field Definitions

The fields are primarily defined in the DL/I DBD source. As described in the section *Conversion of the Data Structure - General Considerations* in the *ADL Conversion* documentation, you can change the default Adabas field specifications before or during the conversion process. One reason for such a modification can be better compression. The more the data is compressed, the fewer I/Os are required. However, the compression itself also needs some time. Therefore, if you add new field definitions for “better” compression, but, in fact, there is nothing to compress, you keep Adabas occupied with unnecessary work

If a Natural program accesses all the data of one segment, the corresponding format buffer (FB) contains all Adabas fields that build up the Adabas group related to the segment. More field definitions require more time for the format buffer translation. This is especially the case if the

call has to be handled by the ADL Consistency Interface, since the format buffer translation of ADL is not as sophisticated as of Adabas.

Enqueue Logic

Batch applications which run in multi-user or shared-database (so called “online batch”) mode should supply their own enqueue logic. Nevertheless, you can run a program in online batch, although it was originally designed to run in normal mode. ADL guarantees the integrity of the data by putting each accessed root record into hold. If the application does not make checkpoints, the Adabas hold queue will soon be full. For this situation, ADL offers the automatic-ET feature, which is controlled by the ET parameter described in the *ADL Installation* documentation, section *ADL Parameter Module*. Because each ET call consumes some time, you should set the ET parameter to a value right below the critical value where an Adabas hold queue overflow would occur.

Under CICS, ADL puts each accessed root record into hold, as in online batch. It is released as soon as the next root record is accessed if there was no update on that hierarchy. Here, the ET parameter helps you reduce the number of RI calls. The records not released remain in hold status until the next syncpoint or terminate call is issued. This may lead to situations where other users have to wait for such records or, fatally, to deadlock situations. If, however, your application avoids such situations, you can adjust the ET parameter to save the time for the RI calls.

CICS/Batch Communication

Usually, DL/I runs in the same region/partition as the user application. This saves interregion communication. On the other hand, it loads the online system, especially when big buffers are required on smaller machines. In addition, if batch applications access online databases, the CICS system has to forward the calls to the “online” DL/I. Adabas runs in its own region/partition (if not in single user mode), Therefore, you have the overhead of the interregion communication, but the online system is relieved of the database management system and, of course, no batch request must be handled by CICS.

Application Program and DB Design

A good programming style is a general task and not directly related to a conversion. However, some problems become obvious when you are using features of ADL or Adabas. If, for example, you see one DL/I call in the ADL trace which is translated into myriads of Adabas calls, you should not hesitate to take a look into your application program or into the DB design. Changes in this area would help DL/I as well as ADL. In addition, if your application makes one million calls too

many, you can tune your database system as much as you like, but you will never get rid of these superfluous calls in this way.

The examples that follow are far from complete. Most of them have been seen on real sites. They should give you an idea of what can be done wrong and how to make things work better.

Examples

Non-descriptor Search

Description:	In a qualified SSA, a normal field is used for searching, that is, no sequence field and no secondary index field.
Effect:	This is a "non-descriptor search". For a root segment, the range of the search extends from the first record to the end of the database. For dependent segments, it is restricted to the children of one specific parent occurrence. The search stops if a record fulfils the qualification; otherwise, it runs until the end of the range is reached.
Action:	Use, as far as possible, the sequence field or an already existing secondary index. Otherwise, define a new secondary index which specifies the field from the SSA as search field.

Wrong Key

Description:	In a qualified SSA for a root segment, the primary key is used, but the PCB specifies an alternate processing sequence.
Effect:	This is also a non-descriptor search, as in the previous example. If a "PROCSEQ" statement is specified in the PCB definition, the only usable key field is the corresponding secondary index field. All the other fields, including the primary key, are treated like non-descriptors.
Action:	Use a PCB without an alternate processing sequence.

Second-level Qualification

Description:	The SSA specifies two levels: the first level is not qualified but the second level is qualified.
Effect:	This is a non-descriptor search on the first level. It stops when the correct second-level segment is found or at the end of the database.
Action:	Define a secondary index on the second-level segment and use this in the call.

Retrieve same Data

Description:	The application program issues a path call over two levels. The first level always specifies the same segment occurrence.
Effect:	The first-level segment occurrence has to be read for each call.
Action:	Once the first level is read, avoid reading it again. You can specify only the lowest-level segment or make no path call.
Note:	If you use the last-call save area (LCS), the record is saved in the LCS buffer. For the repeated read, the value of the LCS is taken and the overhead is minimized.

Concatenated Segment

Description:	In a concatenated segment, the sequence or any other field of the destination parent is used for the qualification in the SSA.
Effect:	This is a non-descriptor search.
Action:	The only key field you can use for concatenated segments is the sequence field of the logical child segment.

No Sequence Field

Description:	A qualified SSA is used for a dependent segment, but this does not have a sequence field.
Effect:	This is a non-descriptor search.
Action:	Define a (non-unique) sequence field for the field used in the SSA.

Read in Sequence

Description:	The application reads the data in the sequence of the sequence field, but it uses qualified direct reads, such as: GU ROOT (SQF = 1) GU ROOT (SQF = 2) GU ROOT (SQF = 3) This can be, for example, in an HDAM database, where the randomized sequence does not match the key-field sequence.
Effect:	Each call is translated into an Adabas call with value-start. Multifetch can not be used. In addition, if there are gaps in the data, a lot of unnecessary calls are issued.
Action:	Read the data with GN calls and an unqualified SSA, such as GN ROOT Under this condition, you can use Multifetch. This also works for HDAM databases, since ADL always returns the data in the sequence of the sequence field.

Segments without Data

Description:	An application makes unqualified GN or GNP calls and some of the sensitive segments in the PCB are never supplied with data.
Effect:	ADL tries to find records belonging to the empty segments. For each parent occurrence, one Adabas call is issued, in order to realize that there is no data of that segment type under this parent.
Action:	Use qualified calls or do not specify the sensitive segments in the PCB.
Note:	If you use the last-call save area (LCS), the next record is saved in the LCS buffer. When reading the empty segment, ADL realizes from the value saved in the LCS that there is no data for this segment and the overhead is minimized.

14 Recovery and Restart Procedures

■ Introduction	174
■ Extensions and Restrictions	174
■ Adabas User Types	175
■ Automatic ET Calls Issued by ADL	179
■ Restart/Recovery Logic under ADL	180
■ ADL Actions for Basic and Symbolic Checkpoints	185
■ How to Restart a Batch Program	186

This chapter covers the following topics:

Introduction

This section describes how the Adabas Bridge for DL/I (ADL) handles DL/Icheckpoint calls and, in the case of an abnormal end to a program, how the database can be recovered and the application program restarted.

Note that the recovery and restart logic and procedures for Natural applications or programs using direct Adabas calls are not affected by ADL.

In DL/I, an application program may set checkpoints using the `CHKP` call. This results in the confirmation of all changes to the database made by the program up to this point. In addition, under z/OS, the `XRST` routine allows the use of “symbolic checkpointing”, with which user data may be saved for each checkpoint. These data can later be used to restart the program, should it have terminated abnormally.

The ADL reproduces the full functionality of the DL/I `XRST` and `CHKP` calls and thus provides complete restart and recovery capabilities.

Adabas provides two basic methods of checkpointing:

- ET (“end of transaction”) logic, intended to be used for updating files concurrently, and
- C1 checkpoint calls, intended to be used by programs which use files exclusively.

Both the `ET` and `C1` checkpoint methods are used by ADL.

Extensions and Restrictions

Symbolic checkpointing is also possible under z/VSE, thus allowing full recovery/restart capabilities.

The following restrictions apply to ADL:

- The use of time stamps instead of checkpoint identifiers is not supported.
- OS/VS checkpoint requests are not supported.
- (for IMS/TP BMP only) “LAST” may not be used to indicate that the program is to be restarted from the last issued checkpoint. Instead, establish the checkpoint identifier of the last issued checkpoint using the messages on the `DAZOUT1` file, and then supply this for the `XRST` call.

Adabas User Types

As mentioned above, the restart/recovery logic used by Adabas in a given situation mainly depends on how an application program accesses the data base. Three user types are distinguished: access-only users (ACC), exclusive file users (EXU) and ET logic users (UPD). ADL uses all of these: which type will be used in a given situation depends on the following features:

- the operational environment;
- the mode in which the application program is run (the first three characters in the parameter statement);
- how the files in the data base are accessed (read only, update), as determined by the PSB specified for the program; and
- the method of (DL/I) checkpointing used (without checkpointing, basic checkpointing or symbolic checkpointing).

The following tables show how ADL applies the Adabas user types and checkpoint methods, e.g. the last column in the first table means

- 1st table: "STA" is specified for DAZIFP, i.e. the program runs as "stand-alone".
- Last block "Update": The program makes updates.
- Last column in block "Symb": The program uses symbolic checkpointing.

Under these conditions, ADL runs the program as Adabas "EXU" user (indicated in the third line), it issues no ETs (forth line) but uses "C1" commands (fifth line) for the ADL checkpoint method.

Normal Batch Programs

Two situations may be distinguished:

- where "STA" has been specified as the first positional parameter for DAZIFP
- where "DLI", or "IMS" has been specified as the first parameter for DAZIFP

With "STA"

		Read Only			Update		
Checkpointing		Without	Basic	Symb	Without	Basic	Symb
Adabas User Type		ACC	ACC	EXU	EXU	EXU	EXU
ADL Checkpoint Method	ET		No	No		No	No
	C1		Yes	Yes		Yes	Yes

With "DLI", or "IMS"

		Read Only			Update		
Checkpointing		Without	Basic	Symb	Without	Basic	Symb
Adabas User Type		ACC	ACC	UPD	EXU	EXU	UPD
ADL Checkpoint Method	ET		No	Yes		No	Yes
	C1		Yes	Yes		Yes	Yes

The advantage of using the "STA" parameter for normal batch programs is that no ET calls have to be issued. However, other application programs also using symbolic checkpoints cannot run concurrently. This is because STA programs use the ADL directory file as exclusive (EXU) users.

SDB, BMP (z/OS) or MPS (z/VSE) Programs

		Read Only			Update		
		Without	Basic	Symb	Without	Basic	Symb
Checkpointing							
Adabas User Type		ACC	UPD	UPD	EXU	UPD	UPD
ADL Checkpoint Method	ET		Yes	Yes		Yes	Yes
	C1		No	No		No	No

CICS or IMS/TP Message Region

		Read Only			Update		
		Without	Basic	Symb	Without	Basic	Symb
Checkpointing							
Adabas User Type		ACC	UPD	-	EXU	UPD	-
ADL Checkpoint Method	ET		Yes	-		Yes	-
	C1		No	-		No	-

Automatic ET Calls Issued by ADL

Adabas ET logic allows concurrent processing of one or more files by more than one user. All records which are updated by a program are held in a queue and may not be updated by any other user until the program issues an Adabas ET call. This results in a confirmation of all updates made by the program and the release of the held records. However, the use of ET logic for programs (e.g. batch programs) which use files as exclusive users or which do not update these files is not recommended.

If ET calls are not issued often enough, the Adabas hold queue for update records may overflow, resulting in a fatal error. On the other hand, ET calls should not be issued too frequently, in order to avoid unnecessary buffer flushes.

ADL includes an option to issue ET calls automatically. The actual procedure performed depends on the program type and the operational environment. Automatic ETs will be issued in the following cases:

- **Normal Batch and SDB, BMP (z/OS) or MPS (z/VSE) Programs**

ADL issues ET calls automatically. The ET parameter provided when creating the ADL parameter module (see the section *ADL Parameter Module* in the *ADL Installation* documentation) or as a keyword for *DAZIFP*, determines the number of changes which can be made to the root segment position in the database (summarized over all PCBs in the PSB) before an ET call is automatically issued. For some batch programs it may be desirable not to issue ET calls at all. This can be achieved by specifying "ET=NO".

■ **IMS/TP**

ADL automatically issues an ET call each time the application program issues a GU call on the I/O PCB.

■ **CICS Environment**

Under CICS, all users are ET logic users and ADL will automatically issue an ET call in two situations:

- where the application program issues an explicit termination call, and
- every time an implicit termination call is taken (i.e. at the end of a task).

When a program terminates abnormally, an Adabas BT call is issued and all database modifications made by this program are backed out.

If OPENRQ=YES has been specified when assembling the parameter module DAZPARM, ADL will issue an Adabas "OP" each time a scheduling call is received, and a "CL" for the termination call.

Restart/Recovery Logic under ADL

The figures on the following pages explain how ADL uses restart/recovery logic for different program types and operational environments.

Normal Batch Programs

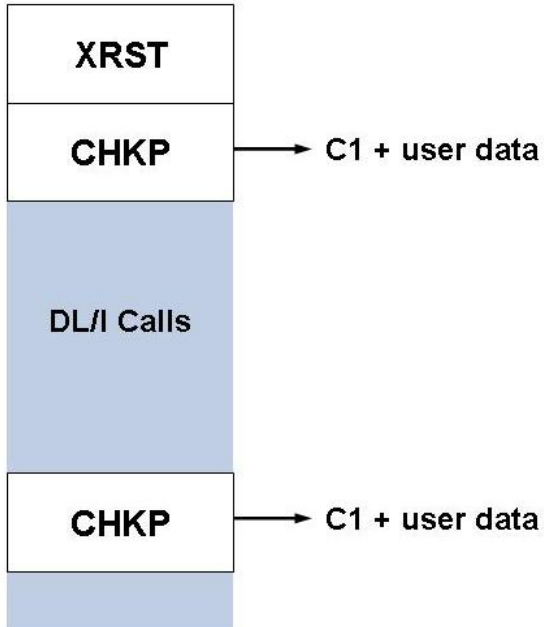
Two situations may be distinguished:

- with "STA" as the first positional parameter specified for DAZIFP
- with "DLI" or "IMS" specified as the first parameter for DAZIFP

These two cases are described in the tables below.

With "STA"

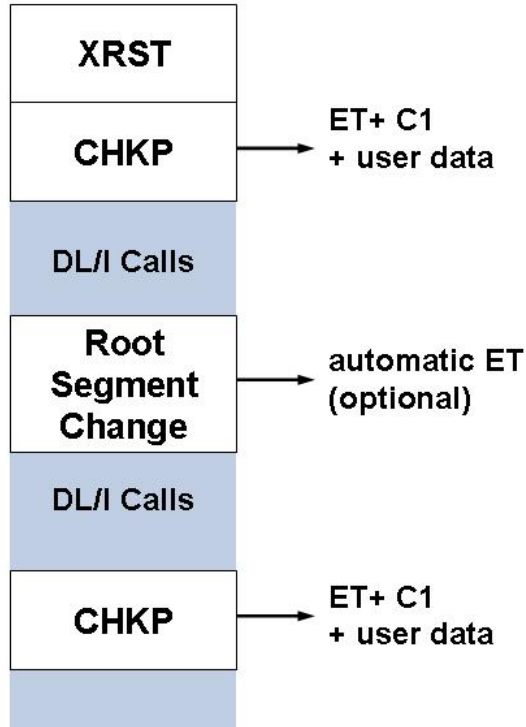
Symbolic Checkpoint



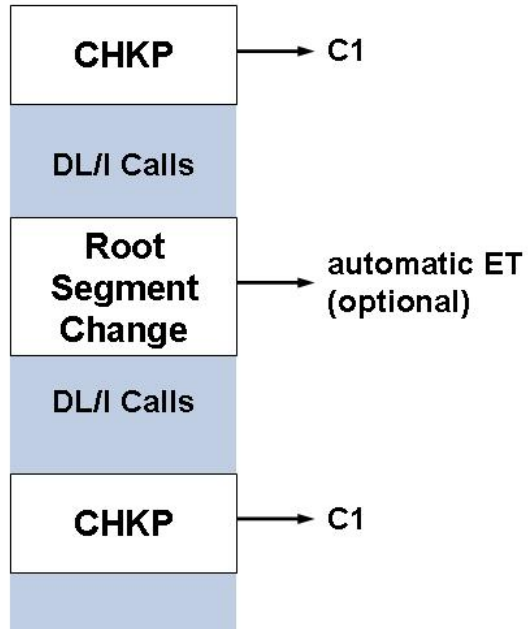
In the case of basic checkpointing, it is recommended that DLI or IMS be used instead of STA as the first parameter for DAZIFP to avoid the ADL directory file being in exclusive mode.

With "DLI", or "IMS"

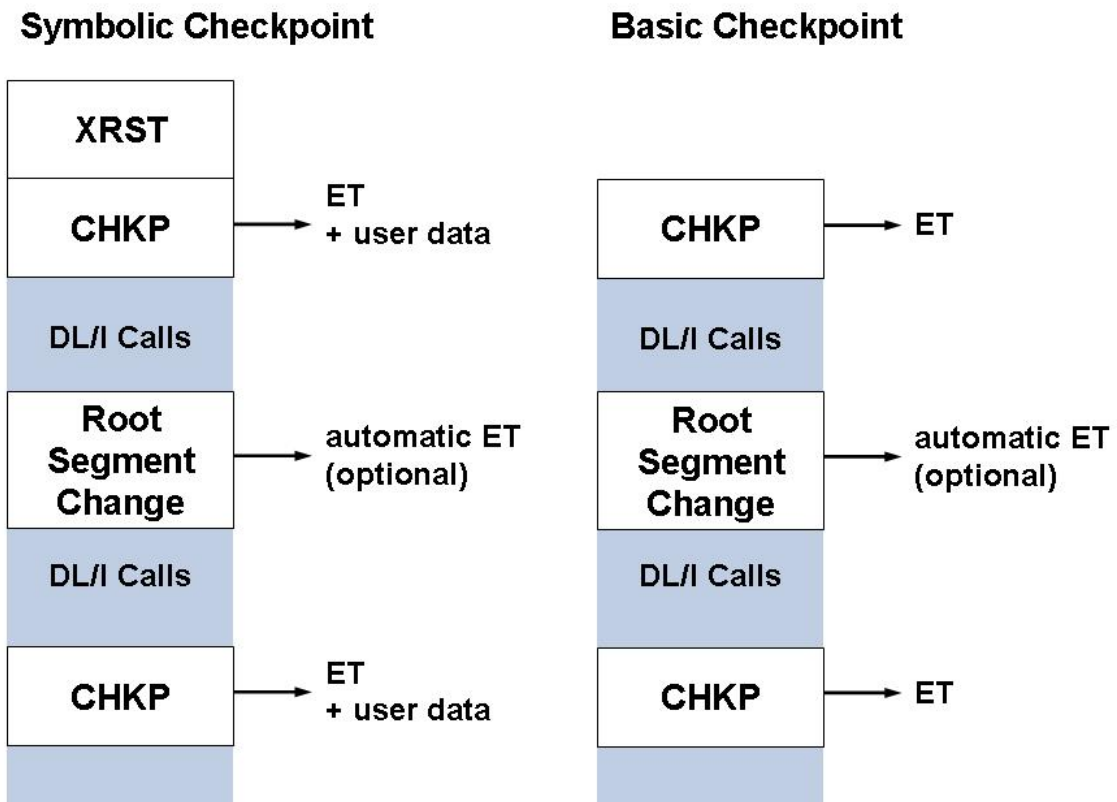
Symbolic Checkpoint



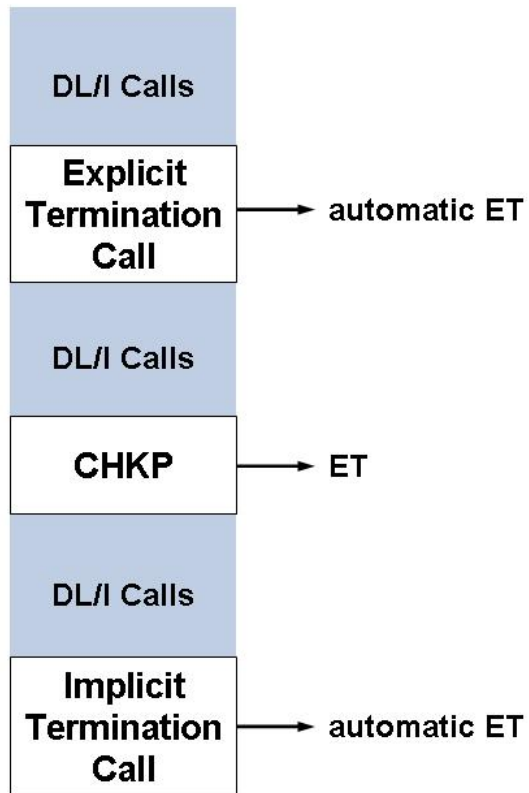
Basic Checkpoint



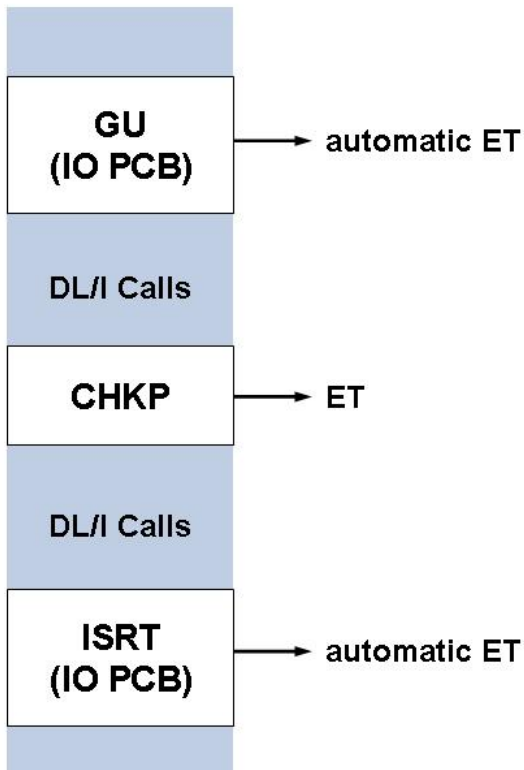
SDB, BMP (z/OS) or MPS (z/VSE) Programs



CICS



IMS/TP Message Region



ADL Actions for Basic and Symbolic Checkpoints

Normal Batch and SDB/BMP/MPS Programs

With a **CHKP** call, the following message is written to the **DAZOUT1** file:

```
'DAZCHKP - CHECKPOINT WRITTEN. DATE: dd/mm/yy TIME: hh/mm/ss FOR PROGRAM progname
WITH CHECKPOINT-IDENTIFIER chkpiden'
```

where

<i>dd/mm/yy</i>	is the actual date,
<i>hh/mm/ss</i>	the actual time,
<i>progrname</i>	the name of the application program,
<i>chkpiden</i>	the eight-character checkpoint identifier as passed to the CHKP call.

If the symbolic checkpoint call is used, up to seven data areas are stored on the ADL directory file. The checkpoints on the latter are identified by the name of the program and the eight-character checkpoint identifier.

Finally, ADL issues an ET and/or a C1 call. In the Adabas checkpoint table, the checkpoints associated with the C1 call are identified by the checkpoint name "CHKP". Please refer to the *Adabas Utilities* documentation for details on how to obtain and read the Adabas checkpoint table.

If a program ends normally, the corresponding checkpoint entries on the ADL directory file will be deleted. If a program ends abnormally, these entries will be deleted after the program has been restarted. An abnormally ended program which is never restarted will thus produce garbage in the directory file. In order to clear the ADL directory file, run a dummy program containing only an XRST call. This dummy program must have the same name as the abnormally terminated program.

Alternatively, you can list and delete the checkpoint entries in the ADL directory with the ADL Online Services as described in the section *The ADL Online Services* in this documentation.

CICS and IMS/TP

ADL issues an ET call for each CHKP call.

In all cases, the current position in the data base will be deleted after a CHKP call. In the case of a restart, the position is *NOT* reestablished after an XRST call.

How to Restart a Batch Program

To restart a batch program, perform the following five steps:

Step 1

Use the output of the application program on the DAZOUT1 file to find the checkpoint identifier and the date and time of the checkpoint at which you want to restart your program.

Step 2

Find the corresponding protection log number and block number of the desired checkpoint in the Adabas checkpoint table. This table may be obtained using the Adabas ADAREP (report) utility. See the *Adabas Utilities* documentation for more details.

Step 3

Make a copy of the protection log file using the Adabas ADARES (restart) utility with either the COPY or the PLCOPY function, for single or dual protection logging respectively. The following ADARUN and ADARES statements are provided as an example (dual protection logging was active):

ADARUN input statement:

```
ADARUN PROGRAM=ADARES,MODE=SINGLE
```

ADARES input statement:

```
ADARES PLCOPY DUALPLD=3350
```

See the *Adabas Utilities* documentation for more details.

Step 4

Restore the entire data base or one or more of its files to the state in effect when the desired checkpoint was taken, using the Adabas ADARES utility with the BACKOUT function. The following ADARUN/ADARES input statements are provided as an example.

ADARUN input statement:

```
ADARUN PROGRAM=ADARES,MODE=SINGLE
```

ADARES input statement:

```
ADARES BACKOUT FILE=37,PLOGNUM=8,TOCP=CHKP,TOBLK=19
```

File 37 is restored to the checkpoint which is found on protection log file 8 in block 19 (all checkpoints issued by ADL are named CHKP). Please refer to the *Adabas Utilities* documentation and the *Adabas Operations* documentation for more details.

Step 5

Restart your program. The eight-character checkpoint identifier may be supplied either with the CPID keyword for DAZIFP or in the I/O area referenced by the XRST call in the application program. If both methods are used at the same time, the value provided with the CPID keyword will be used.

If the checkpoint is not found on the ADL directory file, or if the XRST call requests more data than have been previously stored with the CHKP call, ADL will issue an error message and abend the application program immediately.

15

Appendix A - DAZZLER Test Stream

```

COM *****
COM *
COM *      2)  The following example shows the use of the
COM *          iterative function on the header statement and the use
COM *          of both types of overwrite statement.
COM *          Reading from the left, the header statement specifies
COM *          the following:
COM *
COM *          Do an 'ISRT' call 9 times in succession using PCB1
COM *          with 1 SSA, iteratively increasing the number from
COM *          its start value of 100 by 100 and overwriting the
COM *          I/O Area in position 14 for a length of 5 and in
COM *          position 95 for a length of 4.
COM *
COM *          The next statement causes the first SSA to have the
COM *          segment name in positions 1 to 8 inclusive (all
COM *          segment names are 8 characters long and left-justified)
COM *          followed by 42 spaces. The default format (character
COM *          format) is used as no entry was made in positions
COM *          5 to 8 inclusive. For the same reason, this default
COM *          also applies to the remaining IOA statements.
COM *
COM *          These last four statements cause the I/O Area to be set
COM *          up for a total of 170 characters. The NNNNN entries
COM *          will be iteratively overwritten by the values from
COM *          the *HD statement.
COM *
COM *          Note that the order of the five statements following
COM *          the *HD statement is immaterial: the same results are
COM *          achieved no matter what order they are specified in.
COM *
COM *****
*HD ISRT 009      01 01 00100 00100 0145      0954
S01      001 50 ITEMID
IOA      001 50 ITEMID/MITEM/NNNN*.....
IOA      051 50 .....ITEMID/MMACD/*NNNN**
IOA      101 50 .....END OF
IOA      151 20 SEGMENT ITEMID --->:

COM *****
COM *
COM *      3)  The following example shows a single call
COM *          ('ISRT') using two SSAs.
COM *          The *HD statement specifies an insert using PCB1 and 2
COM *          SSAs.
COM *          The two SSAs and the IOA are created as explained
COM *          above in the second step.
COM *
COM *****
*HD ISRT      01 02

```

```
S01      001 50 ITEMID (MITEM = ITEMID/MITEM/00100*)
S02      001 50 LAGER
IOA      001 50 THIS IS A LAGER SEGMENT THAT CONTAINS NO SEQUENCE
IOA      051 50 FIELD WHATSOEVER - HOWEVER FOR INTERNAL CONTROL PU
IOA      101 50 RPOSES -0101- END OF SEGMENT LAGER --->:
```

```
COM *****
COM *
COM * 4) The following example shows a single call
COM * ('GHU') using two SSAs.
COM * The *HD statement specifies a 'GHU' using PCB1 and 2
COM * SSAs.
COM * The two SSAs are created as explained
COM * above in the second step.
COM *
```

```
*HD GHU      01 02
S01      001 50 ITEMID (MITEM = ITEMID/MITEM/00700*)
S02      001 50 PRODST (SPKEY = ITEMID-00700/SPKEY-701*****)
```

```
COM *****
COM * 5) POSITION AT BEGINNING AND PRINT THE LOT
COM *
COM * This is achieved by issuing an unqualified get
COM * unique which automatically retrieves the first
COM * segment in the data base and follows this with
COM * successive get nexts until the end of the data
COM * base is reached. This condition is indicated
COM * by a status code of 'GB'.
COM * The two *HD statements are translated as follows :
COM * Issue a 'GU' using PCB1 and then issue a 'GN',
COM * also using PCB1. Keep issuing a 'GN' call
COM * until a status code of 'GB' is returned in the
COM * PCB.
COM *
```

```
*HD GU      01
*HD GN      GB 01
```

```
COM *****
COM * LAST) PRINT THE NUMBER OF EACH TYPE OF SEGMENT
COM * ACCESSIBLE BY PCB3
COM *
COM * Internally, this statement will cause an unqualified
COM * GU on PCB3 to be issued, thus positioning at
COM * the beginning. Unqualified GNs are then issued
COM * until a status code of 'GB' is returned in the
COM * PCB. This is apparently the same as the previous
```



```

COM *      two *HD statements in combination (but on a different
COM *      PCB), but the SUMMARY statement does not print out
COM *      each segment as it reads it. Instead, it only
COM *      prints the summary table at the end.
COM *****

```

```
SUMMARY 03
```

```

COM *****
COM *
COM *      As all blank statements and comment lines are ignored by
COM *      DAZZLER, the identical results from the stream above
COM *      would be achieved by the stream which now follows.
COM *
COM *****

```

```
PARUS>PF1067501
```

```

*HD ISRT 009 01 01 00100 00100 0145 0954
S01 001 50 ITEMID
IOA 001 50 ITEMID/MITEM/NNNNN*.....
IOA 051 50 .....ITEMID/MMACD/*NNNN**
IOA 101 50 .....END OF
IOA 151 20 SEGMENT ITEMID --->:
*HD ISRT 01 02
S01 001 50 ITEMID (MITEM = ITEMID/MITEM/00100*)
S02 001 50 LAGER
IOA 001 50 THIS IS A LAGER SEGMENT THAT CONTAINS NO SEQUENCE
IOA 051 50 FIELD WHATSOEVER - HOWEVER FOR INTERNAL CONTROL PU
IOA 101 50 RPOSES -0101- END OF SEGMENT LAGER --->:
*HD GHU 01 02
S01 001 50 ITEMID (MITEM = ITEMID/MITEM/00700*)
S02 001 50 PRODST (SPKEY = ITEMID-00700/SPKEY-701*****)
*HD GU 01
*HD GN GB 01
SUMMARY 03

```


16

Appendix B - 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 Interfaces.

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.

17 Appendix C - z/VSE Dataset Usage

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 *1
DAZIN3D	SYS013	8196	8192	VB	PS	DISK *2
DAZIN3T	SYS013	8196	8192	VB	PS	TAPE *2
DAZOT3D	SYS013	8196	8192	VB	PS	DISK *2
DAZOT3T	SYS013	8196	8192	VB	PS	TAPE *2
DAZIN4D	SYS012	8196	8192	VB	PS	DISK *3
DAZIN4T	SYS012	8196	8192	VB	PS	TAPE *3
DAZIN5D	SYS014	1320	132	FB	PS	DISK*4
DAZIN5T	SYS014	1320	132	FB	PS	TAPE*4
DAZOT5D	SYS012	8196	8192	VB	PS	DISK *3
DAZOT5T	SYS012	8196	8192	VB	PS	TAPE *3
DAZOUT4	SYSxxx		80	F	PS	DISK *5

*1 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.

*2 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.

*3 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.

*4 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.

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

Index

B

Bridge for DL/I, vii

