CICS Installation and Operation

This chapter covers the following topics:

- Overview CICS Installation
- Prerequisites for z/OS CICS Installation
- Prerequisites for z/VSE CICS Installation
- Important Note for CICS Users
- Generating the Runtime Control Tables
- Tuning and Maintaining the Runtime Control Tables
- z/OS Requirements
- z/VSE Requirements
- Activating and Controlling the ADL Interfaces
- CALLDLI Interface
- Consistency Interface
- User Exit

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=dsnname,.....,
//           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=dsnname,....,
//           DCB=(BLKSIZE=nnnn,LRECL=132,DSORG=PS,RECFM=FB)
//DAZOUT5 DD DSN=dsnname,.....,
//           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:

```
SVCPARM 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 sectione *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

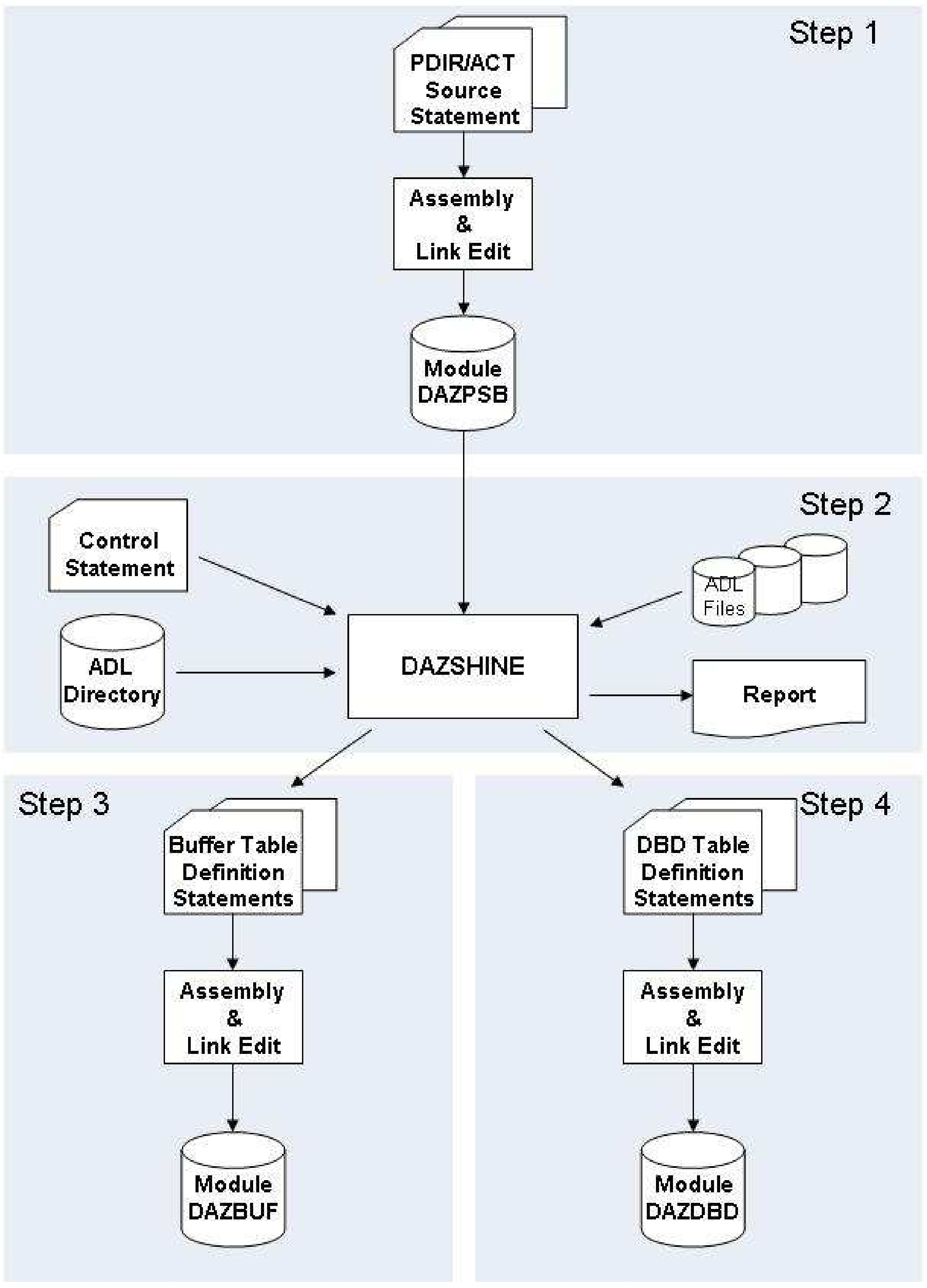
Step 1
PDIR/ACT Source Statement
Assembly & Link Edit
Module DAZPSB
Step 2
Control Statement
ADL Directory
DAZSHINE
ADL Files
Report
Step 3
Buffer Table Definition Statements
Assembly & Link Edit
Module DAZBUF
Step 4
DBD Table Definition Statements
Assembly & Link Edit
Module DAZDBD

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 BUFMAC macro with TYPE=FINAL must be present as the last statement before the `END` statement.
SIZE=	*nnn* or *nnn* K	the size of the slots in bytes or kilobytes (K).
NUM=	*nnn*	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

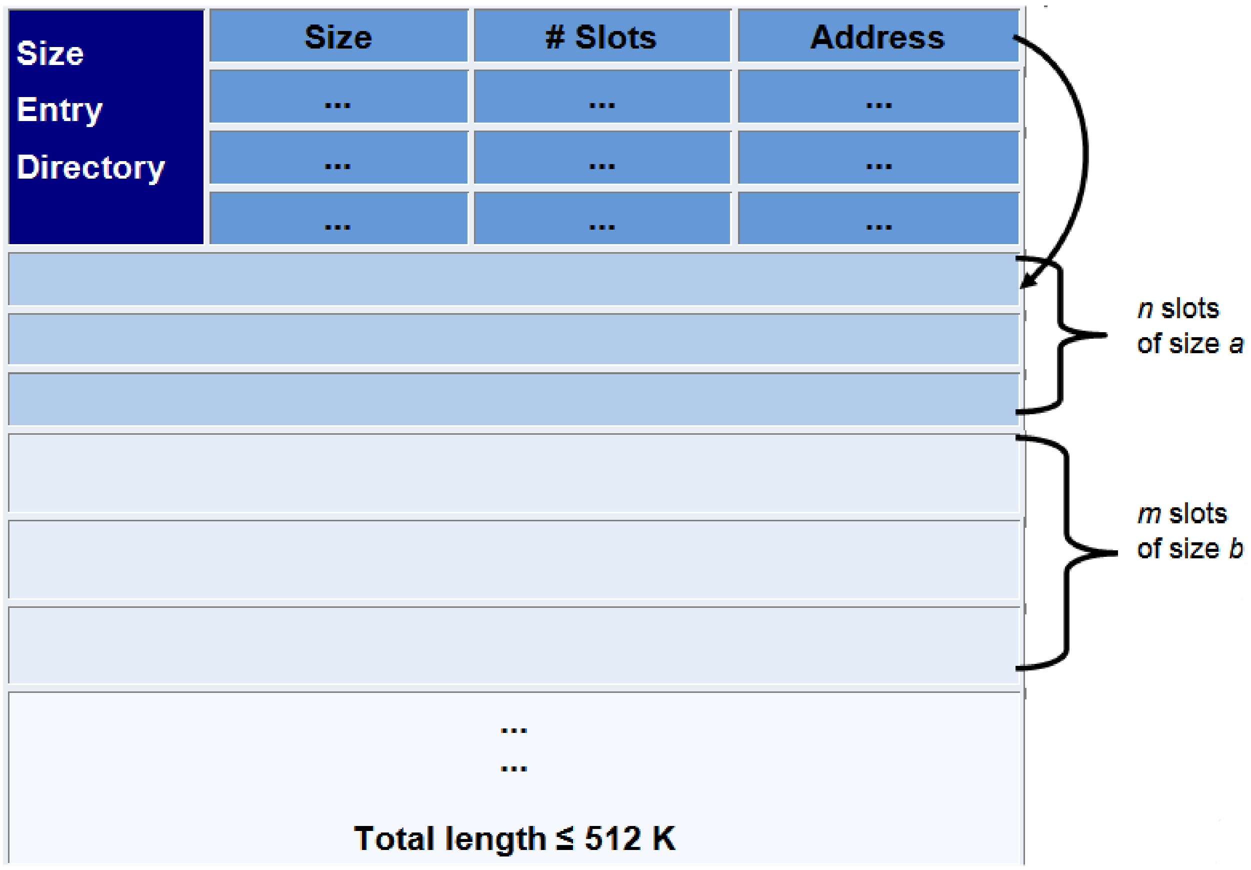

Figure 2b: DAZBUF Buffer Table

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.

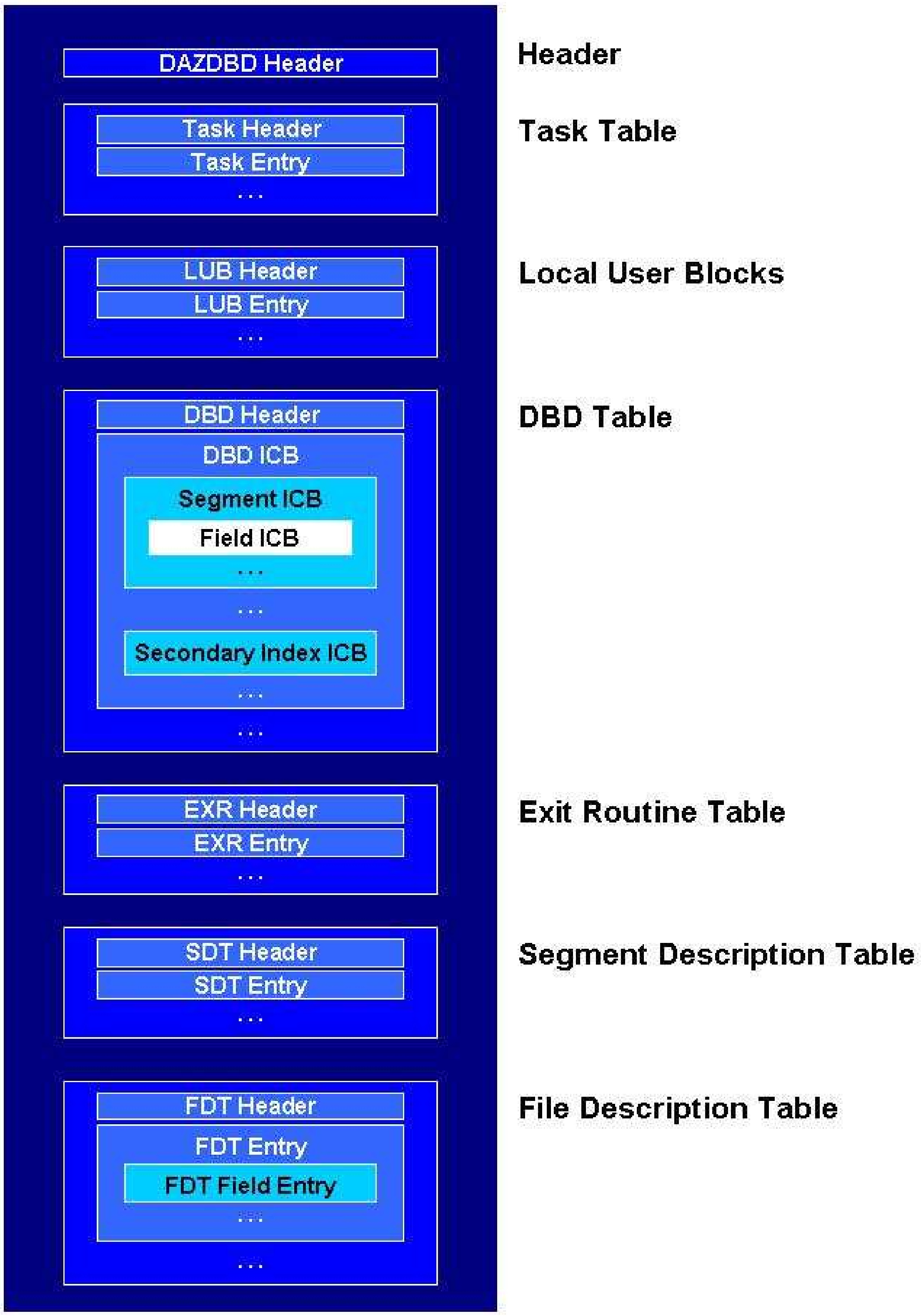
DAZDBD Header
Header
Task Header
Task Entry
. . .
Task Table
LUB Header
LUB Entry
. . .
Local User Blocks
DBD Header
DBD ICB
Segment ICB
Field ICB
. . .
. . .
Secondary Index ICB
. . .
. . .
DBD Table
EXR Header
EXR Entry
. . .
Exit Routine Table
SDT Header
SDT Entry
. . .
Segment Description Table
FDT Header
FDT Entry
FDT Field Entry
. . .
. . .
File Description Table

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.