ADL Conversion Utilities for DBDs and PSBs

For the Adabas Bridge for DL/I to be able to process a DL/I call, it must be aware of the original DL/I structures and the rules according to which they were defined, and how the data is structured under Adabas. This information is in ADL control blocks stored in the ADL directory file. The control blocks are created during the DBD/PSB conversion process, which uses the original DL/I DBD and PSB sources as input. These are assembled and then processed further by the CBC utility.

For an application program to run against ADL, both the PSB used and the DBDs which it references must be run through the DBD/PSB conversion process. This section describes the procedure for generating the necessary ADL control blocks from the DBD/PSB macro source.

This chapter covers the following topics:

- Conversion - Overview
- Conversion - PSBs and all DBDs except Index DBDs
- Conversion - Physical DBDs
- Conversion - Index DBDs
- Conversion - Logical DBDs
- Conversion - Logically Related Physical DBDs
- Conversion - HD Databases
- Control Statements for the CBC Utility
- CBC Utility Output
- z/OS Requirements
- z/VSE Requirements

Conversion - Overview

To convert a DL/I DBD or PSB into an ADL DBD or PSB, you must perform the following steps:

All PSBs and DBDs except Primary Index DBDs

Step	Description
Step 1	Assemble and link edit the original DL/I DBD or PSB source.

Step 1

Assemble and link edit the original source of the DL/I DBD or PSB using the ADL macros provided in the Source Library on the installation tape. You may use the sample JCL contained in the members `ADLDPC1` (z/OS) or `ADLDPC1.J` (z/VSE) for this. For the special link-edit requirements of an HD database, see the corresponding topic later in this section.

In addition to the DL/I keywords you can add the `"PRINT"` keyword to the DBD macro or to the first PCB macro of a PSB.

PRINT=GEN	generates the full assembler listing,
PRINT=NOGEN	is the default and generates the short output.

All PSBs and Physical and Logical DBDs

Step	Description
Step 2	Run the CBC utility once for each DBD/PSB.

Step 2

Run the CBC utility once for each DBD and PSB processed in Step 1. For z/OS, you may use the sample JCL contained in the members `ADLDPC23` for physical DBDs (contains Step 3 as well) or `ADLDPC2` for PSBs and logical DBDs. z/VSE users should consult members `ADLDPC23.J` or `ADLDPC2.J` respectively.

Sample JCL/JCS requirements for the CBC utility can be found at the end of this section.

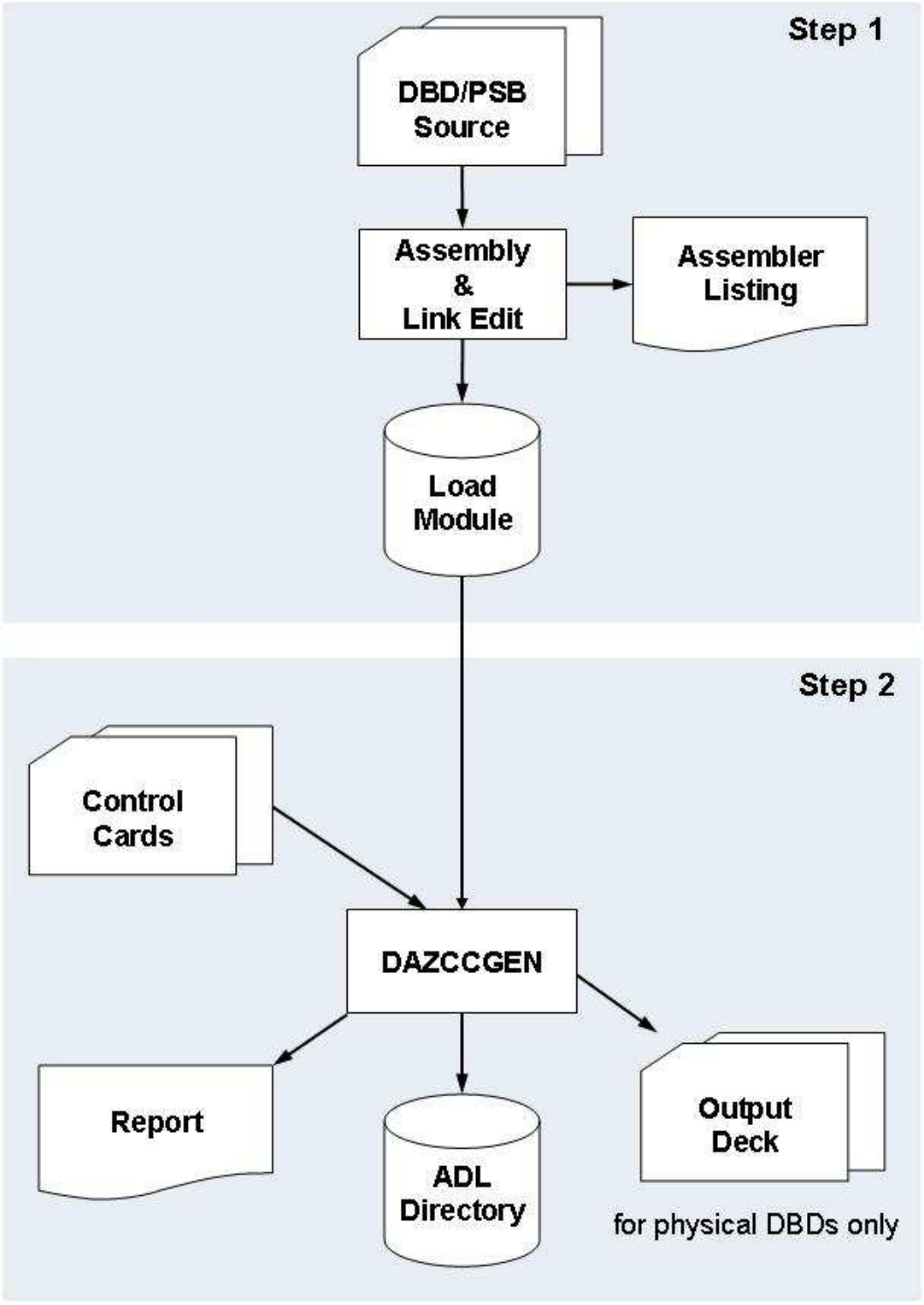
The following input must be provided for each DBD or PSB:

- The load module produced in Step 1;
- One or more control statements, as described later in this section.

The following output is produced:

- An ADL DBD or PSB entry in the ADL directory file;
- The `ADACMP` statements for the Adabas file(s) to be used to store the converted data (for physical DBDs only);
- The Adabas User Exit 6 extensions to be used during initial loading of the Adabas file(s) (for physical DBDs only);
- A report describing the original DL/I structure, the new Adabas structure, and the relationship between the two (for DBDs only).

DBD/PSB Conversion Steps 1 - 2


Step 1
DBD/PSB Source
Assembly & Link Edit
Assembler Listing
Load Module
Step 2
Control Cards
DAZCCGEN
Report
ADL Directory
Output Deck
for physical DBDs only

Physical DBDs Only

Step	Description
Step 3	Create input decks.
Step 4	Assemble and link-edit the Adabas User Exit 6 extension.

Step 3

The output deck created by the CBC utility contains several members (ADACMP control statements and the User Exit 6 extensions for each Adabas file). If the CBC utility has generated output control statements (see the description of the `UTI` parameter in *ADL Parameter Module* in the *ADL Installation* documentation), you can use one of the IBM utilities `IEBUPDTE` (z/OS) or `LIBR` (z/VSE) to create the members. The control statement output data sets generated by the CBC utility are named according to the following conventions:

```
xfffff
```

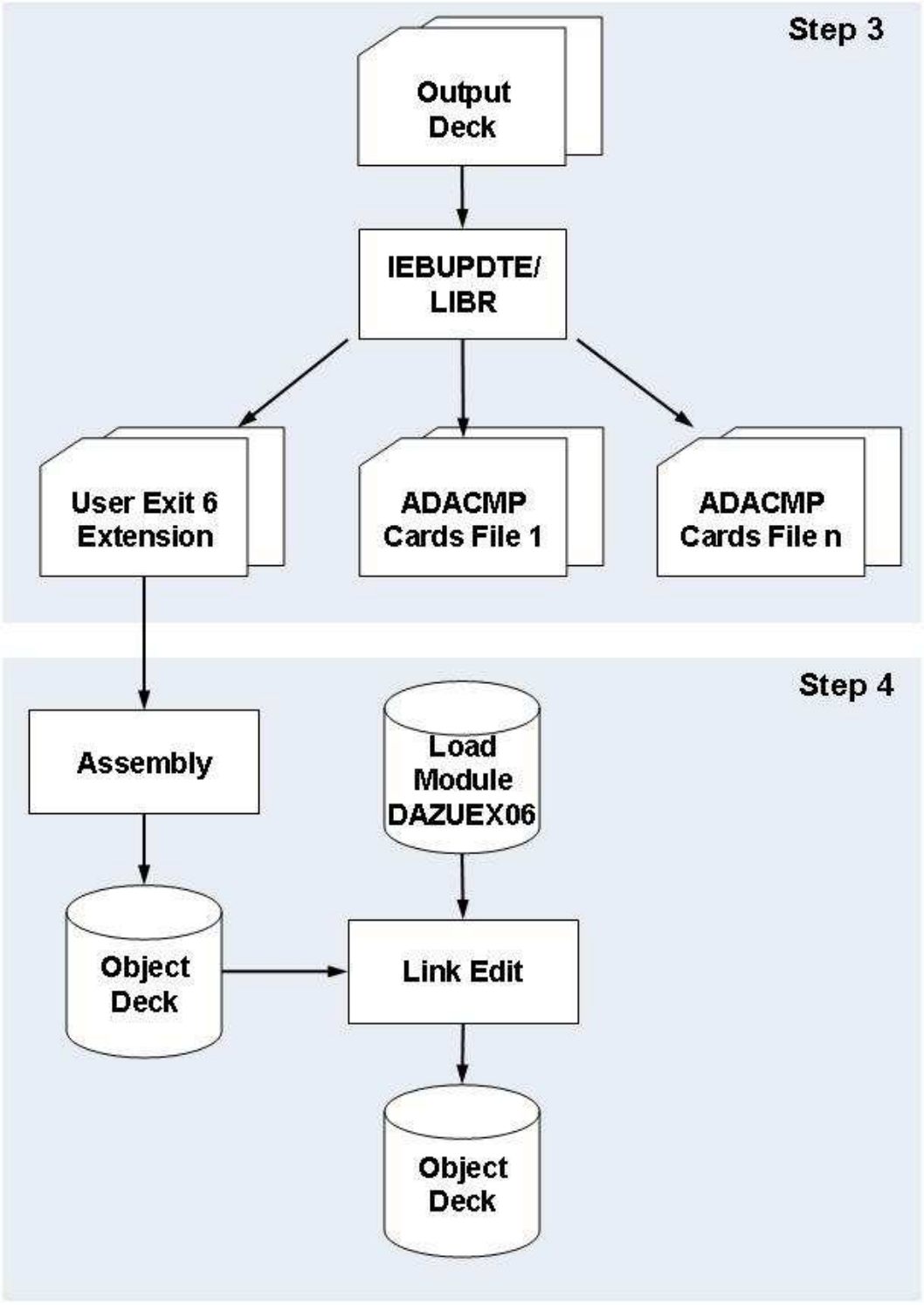
The individual identifiers are as follows:

Identifier	Explanation
x	W for ADACMP control statements I for the Adabas User Exit 6 Extension
fffff	The five-digit number of the Adabas file used to store the root segment data.

Step 4

Assemble the Adabas User Exit 6 Extension (i.e. the output from Step 3) and link edit this with the fixed part, DAZUEX06. You may use the sample JCL in the Source Library member ADLDPC4 (z/OS) or ADLDPC4.J (z/VSE) as an example.

DBD/PSB Conversion Steps 3 - 4


Step 3
Output Deck
IEBUPDTE/ LIBR
User Exit 6 Extension
ADACMP Cards File 1
ADACMP Cards File n
Step 4
Assembly
Load Module DAZUEX06
Object Deck
Link Edit
Object Deck

The control blocks to be converted are of different types, such as PSBs, physical DBDs and logical DBDs. Not all the steps in the DBD/PSB conversion process need to be performed for all control block types. The table below provides an overview of which steps are required for which control blocks and is followed by more detailed explanations.

Type of PSB/DBD	Step 1	Step 2	Step 3	Step 4
Physical DBD	YES	YES *	YES *	YES
Primary Index DBD	NO	NO	NO	NO
Secondary Index DBD	YES	NO	NO	NO
Logical DBD	YES	YES	NO	NO
PSB	YES	YES	NO	NO

* Note that the sample JCL(JCS) in member ADLDPC23 (ADLDPC23.J) enables both steps to be run in a single job.

It is recommended to convert all PSB and logical DBD control blocks once, whether the physical DBDs referenced by them are already converted or not. ADL will automatically detect when a physical DBD is not yet converted and will direct the corresponding database call to DL/I.

This prevents you from monitoring which PSB in detail references which DBD. The only thing you have to do when you convert a further database is to convert the corresponding DBD control block. In a CICS environment, you will have to repeat some of the steps described in the topic *Generating the Runtime Control Tables* in the *ADL Interfaces* documentation.

Conversion - PSBs and all DBDs except Index DBDs

Assemble and link edit all DBDs, except for primary index DBDs, and all PSBs using the ADL macros provided in the ADL Source Library on the installation tape. You may use the sample JCL(JCS) contained in the members ADLDPC1 (ADLDPC1.J) for this. Primary index DBDs and PSBs which do not contain DB-PCBs are not used by ADL and may, therefore, be omitted from the DBD/PSB conversion process. After this, run all DBDs and PSBs except secondary index DBDs through the CBC utility.

If you do not intend to convert all DL/I databases at once, you need to convert only those DBDs describing the databases being converted. Likewise, you need to convert only those PSBs which reference converted data.

PSBs that reference both converted and non-converted databases are used in the mixed mode environment.

Conversion - Physical DBDs

If a physical DBD is to be run through the CBC utility more than once, you must delete it from the ADL directory file first. This must be done to ensure that the latest version of the DBD is taken (i.e. the version in the Load Library and not the one stored in the ADL directory file).

You may delete control blocks for a physical DBD stored in the ADL directory file using the DELDBD function described later in this section, or with the ADL Online Services.

For physical DBDs, Steps 3 and 4 must also be performed in order to complete the conversion process.

Conversion - Index DBDs

All index DBDs, except for the primary index DBD for `HIDAM` or `HISAM` databases, need to be run through the assembly and link edit process described in Step 1. No further steps need to be performed.

The index DBD control blocks are automatically processed and stored in the ADL directory file when the physical DBD for which the secondary index has been defined is run through the CBC utility. This means that you must run all secondary index DBDs through the assembly and link edit process described in Step 1 before you can successfully run a DBD containing secondary indices through the CBC utility. Each secondary index definition will result in the generation of an Adabas descriptor as part of the Adabas file description for the file containing the source segment data.

If an index DBD contains user data fields, Adabas fields will automatically be generated as part of the Adabas file description of the file containing the source segment data.

Once stored in the ADL directory file, the control blocks for an index DBD may be deleted like any other DBD. If you delete a physical DBD with the ADL Online Services, you should also delete all related index DBDs in order to avoid problems which could arise during regeneration.

Where a physical DBD containing secondary indices is to be run through the CBC utility more than once, the ADL control blocks for the secondary index DBDs will be deleted and re-created automatically.

Conversion - Logical DBDs

You must run all logical DBDs through the assembly and link edit process as described in Step 1 and then through the CBC utility as described in Step 2. The only parameter required for the CBC utility is the name of the logical DBD to be converted. All other parameters will be ignored.

No `ADACMP` statements or Adabas User Exit 6 extensions are generated for logical DBDs. This is because logical DBDs are based on database(s) which are described by physical DBDs, and these have to be run through the complete conversion process separately. All `ADACMP` statements and Adabas User Exit 6 extensions needed are generated then.

Once stored in the ADL directory file, the control blocks for a logical DBD may be deleted in the same way as any other DBD.

Conversion - Logically Related Physical DBDs

Logically related physical DBDs reference each other using definitions of logical child and logical parent segment types. When processing a physical DBD, the CBC utility needs access to all other physical DBDs which are logically related to the DBD being processed. The CBC utility first tries to locate a related DBD in the ADL directory file. If this fails, the CBC utility tries to load the related DBD as a load module from the Load Library. If this also fails, the CBC utility reports an error.

This means that before you can successfully run a DBD containing logical relationships through the CBC utility, you have to run all other related physical DBDs through the assembly and link edit process described in Step 1 at the beginning of this section.

Running a physical DBD containing logical relationships through the CBC utility stores both the physical DBD being processed and all other logically related physical DBDs in the ADL directory file. However, this does not complete processing of these ADL control blocks; you still have to run each one of them separately through the CBC utility. You can see whether processing has been completed for all DBDs referenced by the DBD being run through the CBC utility by consulting the list provided at the end of the CBC utility report.

If a physical DBD is logical related to other DBDs, you must specify the `LOGID` parameter at the `GENDBD` function for a unique identification of the DBD's data.

If a physical DBD is to be run through the CBC utility more than once, you must first delete it and all other logically related physical DBDs from the ADL directory file. Then re-convert them all again (without further intermediate deletion). This ensures that the latest version of the DBDs is used (that is, the version in the Load Library and not that stored in the ADL directory file).

Conversion - HD Databases

For `HD` databases an alternate sequence is defined by the `ACCESS` macro, which replaces the secondary index DBDs. When such a DBD source is assembled, the result is one module containing the physical DBD as well as the secondary index DBDs. Since ADL expects the secondary index DBDs on separate members, this module must be split up, as described below. The remaining steps 2 - 4 of the conversion can be performed in the common way, as described earlier in this section.

z/OS Requirements

Add the following entries to the link-edit for the physical DBD:

```
ENTRY   dbdname
REPLACE     secname-1
.
.                   for each secondary index
.
REPLACE     secname-n
INCLUDE     OBJMOD
NAME    dbdname(R)
```

for each secondary index add the following entry:

```
ENTRY   secname-x
REPLACE dbdname
REPLACE secname-1
.
.                   all secondary indices, but not secname-x
.
REPLACE     secname-n
INCLUDE     OBJMOD
NAME        secname-x(R)
```

z/VSE Requirements

Assemble the DBD with option "DECK" to create an OBJ module and then add the following entries to the link-edit for a physical DBD:

```
PHASE dbdname,*,NOAUTO
INCLUDE dbdname,(dbdname)
```

and for each secondary index add the following entry:

```
PHASE secname-x,*,NOAUTO
INCLUDE dbdname,(secname-x)
```

Control Statements for the CBC Utility

As mentioned earlier, you must provide a control statement for each DBD and PSB processed using the CBC utility. Each control statement must have the following format:

```
function p1,p2... comments
```

The individual parameters are as follows:

Parameter	Explanation
function	A function keyword. It must start in the first column of a statement.
p1,p2...	Parameters for the function given. They must follow the function and be separated from it by at least one blank. No blanks should be left between parameters.
comments	Comment statements. Comments can be made on statements by leaving at least one blank after the last parameter, or by inserting an asterisk ("*") in the first column of a statement.

The various function keywords are as follows:

Function	Parameters
GENDBD	NAME=DBD-name, LOGID=logical-Id, DBID=dbid, FNR=file-number, TYPE=conversion-type, CONSI=YES/NO
GENSEG	NAME=segment-name, LOGID=logical-Id, FNR=file-number, ADANAME=Adabas-short-name, BACKW=YES/NO
DELDBD	NAME=DBD-name
GENPSB	NAME=PSB-name
DELPSB	NAME=PSB-name

These functions are described in more detail in the following topics:

- GENDBD Function

- GENSEG Function
- DELDBD Function
- GENPSB Function
- DELPSB Function

GENDBD Function

The `GENDBD` function initiates the processing of the DBD to be converted. Its parameters are as follows:

Parameter	Description
NAME	The name of the ADL DBD to be converted. It can be between 1 and 8 alphanumeric characters long; the first character must be a letter. The name specified must be the same as the original DL/I name of the DBD.
LOGID	(for physical DBDs only) The default logical ID of the DBD. If the DBD is involved in logical relationships, the `LOGID` given must be different from the `LOGIDs` of the other DBDs. Possible values: 1 - 25 Default: 1
DBID	(for physical DBDs only) The Adabas database ID for the database which will be used to store the converted data. If it is omitted, the `DBID` of the ADL directory will be used. If the `DBID` is the same as the `DBID` of the ADL directory, it is recommended to omit the `DBID` parameter. This eases the creation of mirror databases. Possible values: 1 - 65535 Default: none
FNR	(for physical DBDs only) The default Adabas file number for the file which will be used to store the converted data for all segment types not affected by a `GENSEG` function. Possible values: 1 - 65534 Default: none
TYPE	(for physical DBDs only) This parameter is used to indicate whether the database being processed is to be converted to Adabas or not. Possible values: `ADA` the database will be converted `DLI` the database will reside under DL/I Default: `ADA`
CONSI	(for physical DBDs only) This parameter specifies whether the converted DBD may be accessed by Natural/Adabas applications via the ADL Consistency Interface or not. The assignment can be modified later using the ADL Online Services. The `CONSI` parameter must be set to `YES` if the DBD is accessed by both DL/I and Natural/Adabas applications. For more information, refer to the section Consistency DBD Maintenance in the section *ADL Online Services* in the *ADL Interfaces* documentation . Possible values: YES = DBD used by ADL Consistency Interface NO = DBD not used by ADL Consistency Interface. Default: `YES`

Note:
If the DBD was originally converted with ADL version 2.2 or before, the LOGID should specify the same value as the ADL 2.2 FNR parameter.
The SEQ (processing sequence) parameter of ADL 2.2 has become obsolete. The new layout of the ADL internal pointer field has the same performance advantages as the previous "SEQ=SEG" setting.
To avoid conflicts with the logical files settings (LFILE) of Natural, it is recommended not to use DBID=255.

GENSEG Function

The GENSEG function is used to control DBD conversion for a DL/I segment. It may be specified for each segment of a physical DBD but not for virtual logical child segments. If no GENSEG function is specified for a particular segment, then the default parameter values are used.

The function parameters are explained below.

Parameter	Description
NAME	The name of the segment in the DBD currently being processed. This may be between 1 and 8 alphanumeric characters long. The first character must be a letter.
LOGID	The logical ID of the segment. The number given will be used as default for all dependent segment types. If the DBD is involved in logical relationships, the LOGID given must be different from the LOGIDs of the other DBDs. Possible values: 1 - 255 Default: LOGID of parent segment or LOGID of GENDBD function for the root segement
FNR	The Adabas file number for the file which will be used to store the converted data of this segment type. The number given will be used as the default for all dependent segment types. Possible values: 1 - 65534 Default: none
ADANAME	The Adabas short name to be used as the name of the Adabas field group which will store the segment data. If the ADANAME parameter is not specified, then ADL generates the Adabas group names automatically.
BACKW	Used to specify whether the Adabas descriptor used for internal reading backwards is to be maintained for the segment being generated. This is done in order to optimize retrieval of the last segment occurrence in a twin chain. Possible values: YES = indicates that the descriptor will be maintained; NO = indicates that it will not. The descriptor is maintained automatically where the physical insert rule is either HERE or LAST.

Note:
If the DBD was originally converted with ADL version 2.2 or before, the LOGID should specify the same value as the original FNR parameter. Otherwise it is not required to specify LOGIDs for segments even if the DBD data is saved on multiple Adabas files.

DELDBD Function

The DELDBD function deletes an existing DBD from the ADL directory file.

The function parameter is as follows:

Parameter	Description
NAME	The DL/I name of the DBD to be deleted. This may be 1 to 8 alphanumeric characters long. The first character must be a letter.

GENPSB Function

The `GENPSB` function initiates processing of the PSB specified.

The function parameter is as follows:

Parameter	Description
NAME	The DL/I name of the PSB to be converted. This may be 1 to 8 alphanumeric characters long. The first character must be a letter.

DELPSB Function

The `DELPSB` function deletes an existing PSB from the ADL directory file.

The function parameter is as follows:

Parameter	Description
NAME	The DL/I name of the PSB to be deleted. This may be 1 to 8 alphanumeric characters long. The first character must be a letter.

CBC Utility Output

The CBC utility produces up to four different types of output:

1. *ADL control blocks on the ADL directory file.*

   The CBC utility stores and updates the ADL DBDs and PSBs on the ADL directory file.

2. *Output deck containing ADACMP statements and the Adabas User Exit 6.*

   This output is only produced when the CBC utility is processing a physical DBD. The ADACMP statements for all files used to store the data and the Adabas User Exit 6 extension for loading the data into the Adabas file(s) are produced as one output deck. Each set of records in the deck is separated from the others by control statements. These can be interpreted by an IBM utility (LIBR for z/VSE and IEBUPDTE for z/OS) to create separate members in a library.

   Note that the generation of output control statements by the CBC utility can be suppressed by specifying the parameter UTI=(,N). See the chapter The ADL Parameter Module in the ADL Installation Manual for further details.

3. *Control print output.*

   A print file giving the input control statements for the CBC utility and the action taken. It has the following format:

   nnnnn FUN input statement ACT message

   where *nnnnn* is the number of the input statement.

   The input statements are followed by information on the run, including the number of report pages produced and the number of input statements.

4. *Report (physical and logical DBDs only).*

   In the case of physical DBDs, the report contains a complete overview of the DL/I and Adabas structures involved. A list of all DBDs referenced by the DBD just processed, i.e. secondary index DBDs and other physical DBDs related via logical relationships, is given at the end of the report. The list also indicates whether processing of the DBDs referenced has been completed or not.

z/OS Requirements

The following table lists the data sets used by the CBC utility `DAZCCGEN`.

DDname	Medium	Description
DAZIN1	Reader	Control input for the ADL batch monitor, DAZIFP.
DAZOUT1	Printer	Messages and codes.
DAZOUT2	Printer	Report.
DAZOUT4	Disk	ADACMP statements and Adabas User Exit 6 extension.

Example:

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

```
//         EXEC PGM=DAZIFP,PARM='UTC,DAZCCGEN'
//STEPLIB  DD DSN=ADLxxx.LOAD,DISP=SHR
//         DD DSN=ADABAS.Vnnn.LOAD,DISP=SHR
//DDCARD   DD *
ADARUN PROGRAM=USER,...
//DAZIN1   DD *
DELDBD NAME=COURSEDB
GENDBD NAME=COURSEDB,DBID=009,FNR=034
//DAZOUT1  DD SYSOUT=X
//DAZOUT2  DD SYSOUT=X
//DAZOUT4  DD DSN=&&DECK,DISP=(,PASS),UNIT=SYSDA,
//            SPACE=(80,(100,100),RLSE),
//            DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//*
//         EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD SYSOUT=X
//SYSUT2   DD DSN=ADLxxx.SOURCE,DISP=SHR

//SYSIN    DD DSN=&&DECK,DISP=(OLD,DELETE)
```

z/VSE Requirements

The following table lists the files used by the CBC utility, DAZCCGEN.

DTF	Logical Unit	Medium	Description
DAZIN1	SYSIPT	Reader	Control input for the ADL batch monitor, DAZIFP.
DAZOUT1	SYSLST	Printer	Report, messages and codes.
DAZOUT2	SYS011	Printer	Report. *
DAZOT3D	SYS013	Disk	Report. **
DAZIN3D	SYS014	Disk	Report. **
DAZOUT4	SYSxxx	Disk	ADACMP statements and Adabas User Exit 6 extension.

* 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 written 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 the CBC utility itself are all read from SYSIPT. The control statements for this must be specified in the following order:

```
UTC,DAZCCGEN,,...                          input for DAZIFP
/*
ADARUN DB=dbid,MO=MULTI,PROGRAM=USER,... input for ADARUN
/*
DELDBD NAME=dbd                            input for the CBC utility
GENDBD ...
.
/*
```

Example:

The following is an example of a job to execute the CBC 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 SYS014,volser,......
// EXEC DAZIFP
UTC,DAZCCGEN
/*
ADARUN PROGRAM=USER,..........
/*
DELDBD  NAME=COURSEDB
GENDBD  NAME=COURSEDB,DBID=9,FNR=34
/*
// DLBL IJSYSIN,'punchfile'
```

```
// EXTENT SYSIPT,volser
ASSGN SYSIPT,DISK,VOL=volser,SHR
// EXEC LIBR,PARM='ACCESS S=SAGLIB.ADL...'
/&
CLOSE SYSIPT,FEC
```