Managing ADL Files

This chapter covers the following topics:

- Overview
- Reorganization with ADL Utilities (Pseudo Mixed Mode)
- Reorganization with ADL Utilities (Normal Mode)
- Reorganization with Adabas Utilities
- Reorganization with User-written Programs
- Change of DL/I Segment Definitions
- Change of DL/I Field Definitions
- Change of the Data Layout
- Re-establishment of the Hierarchical Sequence
- Change of DL/I PSB Definitions
- Change of the ADL Structure
- Change of Adabas DBIDs and File Numbers
- Change of the Adabas File Layout

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 it 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.