ADL Installation Verification Package

This chapter covers the following topics:

- Introduction
- DL/I Terms
- Example Database
- Adabas Terms
- DL/I terms versus Adabas terms
- IVP Sample JCL
- Conversion of the Example Database
- DL/I Applications for the Installation Verification Package
- DDMs for the Installation Verification Package
- Example Database Application
- Other Natural Objects of the Installation Verification Package
- Tuning the ADL Installation Verification Package

Introduction

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

The ADL IVP consists of the following parts:

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

- DDM definitions

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

DL/I Terms

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

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

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

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

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

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

Example Database

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

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

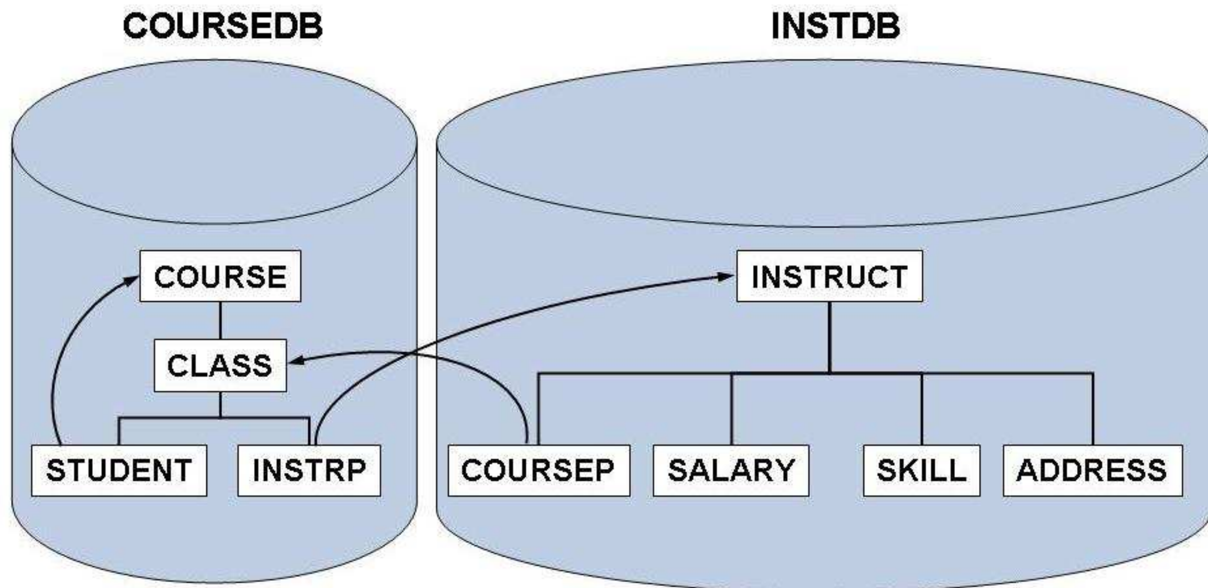


Figure 1: Example Database

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

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

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

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

Adabas Terms

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

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

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

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

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

DL/I terms versus Adabas terms

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

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

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

IVP Sample JCL

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

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

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

Conversion of the Example Database

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

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

Run the sample jobs in the following sequence:

1. IVPDPC1: Assemble and link-edit all DBD and PSB sources. Only the primary index DBD definitions are not assembled, because ADL does not need them.

2. IVPDPC2: Convert the PSBs and the logical DBDs. The DBID and FNR of the ADL directory on which the definition is stored should have already been defined during the ADL Installation. Note that a logical DBD or a PSB can be converted, even if the corresponding physical DBD is not yet converted.

3. IVPDPC23: Convert the databases INSTDB and COURSEDB. The GENSEG statement for the STUDENT segment is used to store the student data in a different file than the course and class data. This job generates the ADACMP cards for all three files and the macro cards for the Adabas User Exit 6.

4. IVPDPC4A/C: Assemble and link-edit the User Exit 6 cards for COURSEDB and INSTDB respectively, which have been generated in the previous step. Note that there is one User Exit 6 per database, even if the segments are distributed over several files.

5. IVPDBC6A/B/C: Initial load of the Adabas files *a*, *b*, and *c*, respectively. Each job consists of the following steps: Delete the file (if it already exists), compress the data, and load the file. Since we have not unloaded any data before, we now load an empty file. In this case, User Exit 6 is not required. Moreover, we do not need to establish logical relationships in an empty file, which is normally done by running the DAZELORE utility.

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

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

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

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

6. IVPINVA: Create additionally descriptors for file a. We make the field AC (CLASSNO) a descriptor and create a superdescriptor S1 (CCK-CLASS), build up by the fields PC (PCK-COURSENO) and AC (CLASSNO). With the help of this superdescriptor we can easy read the class data in the hierarchical sequence.

7. IVPINVB: Create additionally descriptors for file b. We make the field AA (SURNAME) a descriptor and create a superdescriptor S1 (CCK-CLASS), build up by the fields PC (PCK-COURSE) and PB (CLASSNO). With the help of this superdescriptor we can easy read the student data in the hierarchical sequence.

8. IVPINVC: Create additionally descriptors for file c. For each of the dependent segments SALARY, SKILL and ADDRESS we create a superdescriptor (S2 - S4), which is build up by the field PA (PCK-INSTRUCT) and a part of the segment data. With the help of these superdescriptors we can easy read the segments data in the hierarchical sequence. Additionally we create the superdescriptors S1 (CCK-COURSEP) and S5 (CCK-INSTRP) to reflect the hierarchical view of the COURSEP and the INSTRP segments.

DL/I Applications for the Installation Verification Package

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

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

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

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

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

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

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

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

DDMs for the Installation Verification Package

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

```
DBDname-segmentname
```

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

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

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

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

Example Database Application

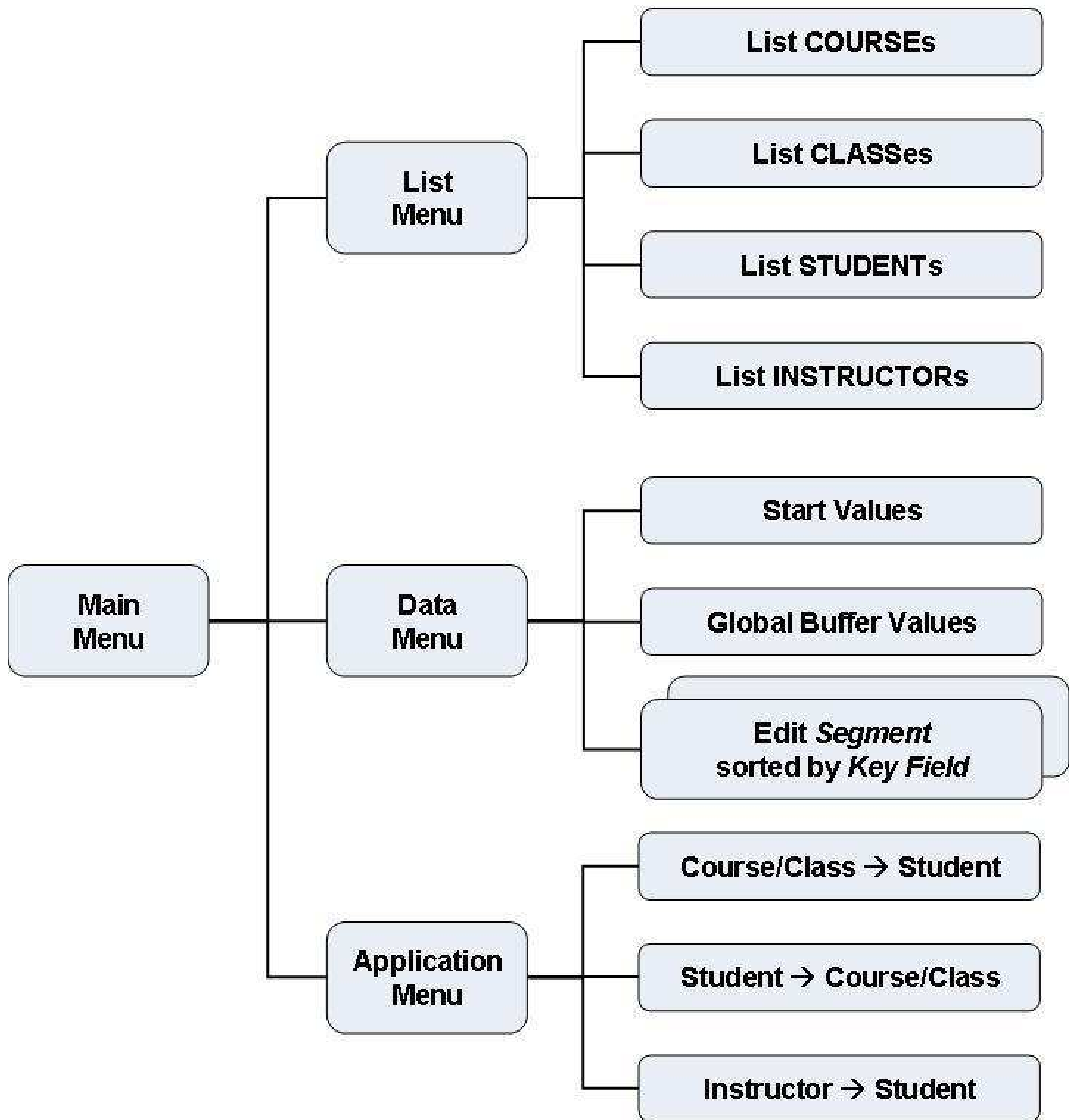
Logon to the Natural library SYSADLIV and catalog all sources by using the Natural CATALL utility.

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

`ADBXP`*`ntm`*

where

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



The Example Database Application SYSADLIV

SYSADLIV Main Menu

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

 Consistency: Active            - Main  Menu -             Program: ADBXP



                            Listings             4
                            Applications         5
                            Data Editor          7
                            Quit                 .


                            Option :




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

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

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

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

SYSADLIV List Menu

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

                               - List  Menu -               Program: ADBXPA



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




 Press a PF-KEY!


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

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

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

SYSADLIV Application Menu

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

                            - Applications  Menu -         Program: ADBXPB



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





 Press a PF-KEY!


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

The Applications Menu provides you with the following functions:

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

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

SYSADLIV Data Menu

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

                               - Data  Menu -            Program: ADBXPC


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

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

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

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

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

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

SYSADLIV Example Database Editor

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

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

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

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

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

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

The PF-keys provide the following functions:

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

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

Other Natural Objects of the Installation Verification Package

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

- The program DEL-COUR deletes a COURSE segment occurrence. Before it performs the deletion, it checks, whether the COURSE has dependent segments.
- The program INS-STUD inserts a new student record. Before the insert, it verifies whether the chosen COURSE/CLASS path exists.

- The programs UPD-COUR and UPD-STUD update a COURSE and a STUDENT record. These programs modify neither the sequence field nor the PCK fields. The UPD-STUD program updates the secondary index source fields, while the secondary index descriptor field (XA) is handled by the Consistency.

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

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

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

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

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

- The program LFILE sets the Natural `LFILE` parameter for the ADL directory. It reads the new DBID and FNR from the input.
- The subprogram SETLFILE sets the Natural `LFILE` parameter for the ADL directory. Use function=3 and specify the DBID and FNR parameters (each 5 byte numeric) as required. The subprogram returns a 2 byte integer response.

Tuning the ADL Installation Verification Package

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

- Hierarchical Sequence
- Last Call Savearea (LCS)
- ADARUN Multifetch Feature

Hierarchical Sequence

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

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

Last Call Savearea (LCS)

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

ADARUN Multifetch Feature

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

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

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

Performance Test Streams

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

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

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

Number of DL/I calls: 426

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

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

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

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

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

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

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

Number of DL/I calls: 311

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

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