IKAN ALM IDD

Release 1.0

February 2019

IKAN Development N.V.
Kardinaal Mercierplein 2
2800 Mechelen
BELGIUM

Table of Contents

CHAPTER 1
Terminology

IDD Entity	An entity in the IDD that identifies a software component which can be a Process, Map, Dialog, Record, Table, Module, File, Schema, Subschema or Adsa Application
Using Entity	An entity that uses another software component, also called a Parent.
Used Entity	An entity that is used by another software component, also called a Child.
Entity File	The file that contains one line per IDD entity in the build that identifies its Type, Name and Version, and the relationship of such an entity to other IDD entities.
Source File	The file that contains the source of one (1) or more IDD entity(ies) that need(s) to be migrated to the next IDD in the life-cycle. There will be several 'source files', i.e., one per IDMS utility that handles the sources. So, we will have one source file to be handled by Idmsdddl (Records, Programs, Modules, Processes, Tables), one for Schema sources, one for Subschema sources and one for Map sources.
Child	An entity that is used by one or more other entity(ies).
Parent	An entity that is using one or more Child(ren).
Source IDD	An IDD from where you want to migrate, i.e., to extract, IDD components
Target IDD	An IDD to which you want to migrate, i.e., to extract, IDD.

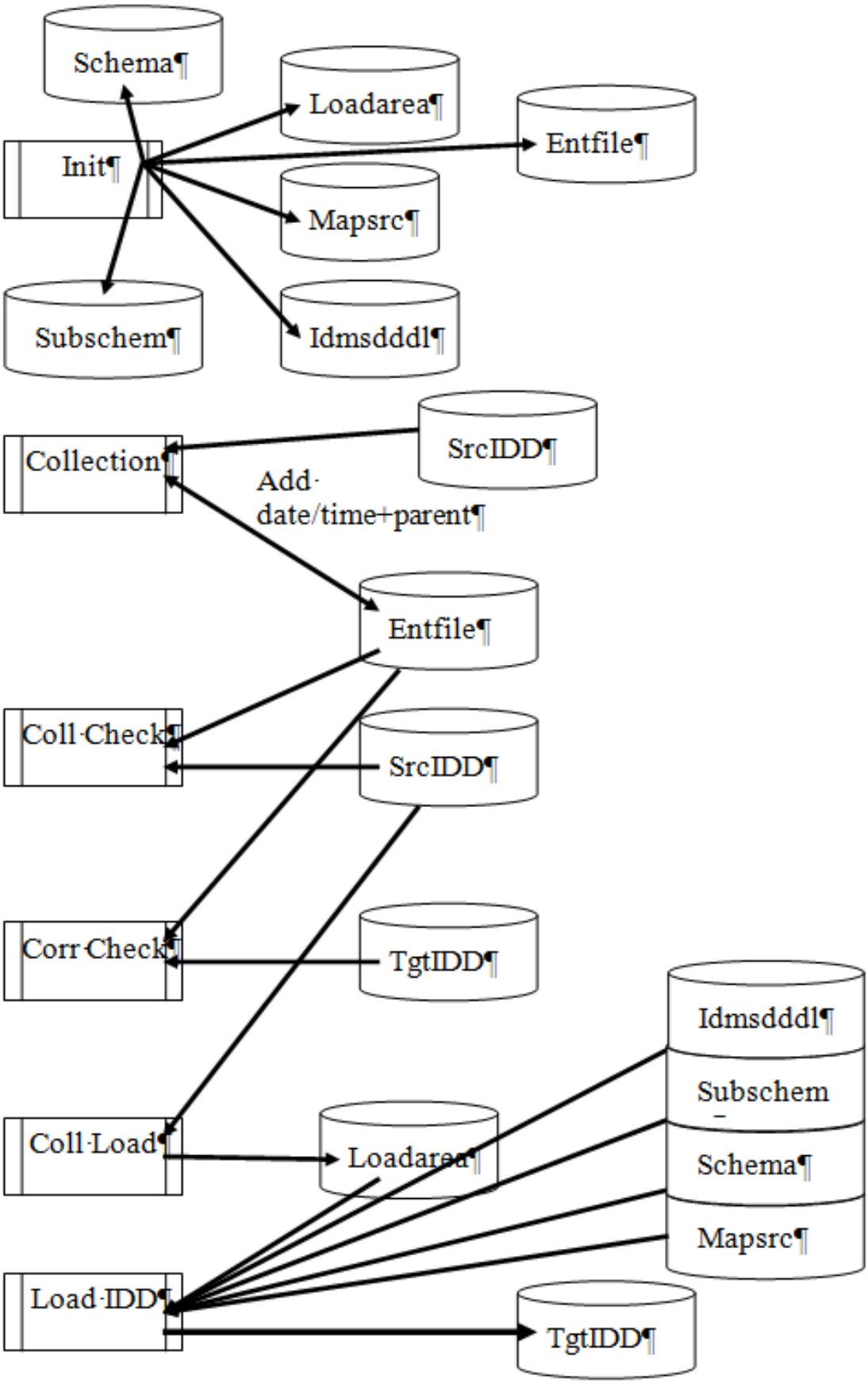
Example:
A dialog is a Parent and uses a Map (child) and/or Records(child) and/or Adso Processes (child).
A record is a Child and is used by a Map and/or a Dialog and/or an Adsa Application.

CHAPTER 2
Process

The complete process should have the following functions/parts:

- Initialization (allocate permanent files etc.)
- 'Collection' of parent information and date/time of both child and parent.
- 'Collection check' to check whether child-parent relations identify errors in the sequence of modifications in the IDD.
- 'Correlation check' to check whether the 'entity file' contains migration stoppers.
- 'Collect Loads' to copy the required load Area Modules.
- 'Load Idd' to load the IDD sources (if required; this is implementation-specific) and the Load Area

Modules into the target IDD.



2.1. Collect Information

The relationship between Children and Parents is stored in the 'Entity File', together with the date and time information of both the children and their parents.

- Collect Process-Dialog relationship (between an Adso Process and a Dialog)
- Collect Map-Dialog relationship
- Collect Record-Dialog
- Collect Record-Map
- Collect Record-Application (Adsa)
- Collect Record-Table
- Collect Record-Schema
- Collect Table-Map

For every child-parent relationship we need to pick up the date and time of the last modification/creation.

This means that we need to create a file on the mainframe that will contain information for all the entities in the build. The information needed is the Type, Name and Version of that entity.

Example:

We have two Dialogs and one Process. Initially, the file will contain the following information:

```
TYPE=PROCESS NAME=<process name> VERSION=<process version>
TYPE=DIALOG   NAME=<dialog1 name> VERSION=<dialog1 version>
TYPE=DIALOG   NAME=<dialog2 name> VERSION=<dialog2 version>
```

The Version, Date and Time of the entity can be obtained from the source of the entity in the VCR.

To populate the file we need to do the following:

1. Transfer a single IDD entity source to a sequential file on the mainframe.
2. Submit a job with the following steps:
    - Read the sequential file and extract the IDD Type, Name, Version, Date and Time from the source and write the information to a 'temp file'. The only exception here is the type=Map. The Date and Time are not present in the source of the entity.
    - Process the 'temp file' and read the IDD starting with the Type, Name and Version of the IDD entity and navigate through the IDD to pick up Parent relationships and extract the Type, Name, Version and Date/Time of the Parent entity.
    - Add the Date/Time of the Child entity and the Type, Name, Version and Date/Time of every Parent entity to the 'Entity file' (no replace)
    - Add the source to the 'source file' (no replace)

Note: This sequence should be repeated for every IDD entity source file in the build.

To extract the Type, Name, Version, Date and Time from the sequential file we need the type of the source to be passed as a parm to REXX IKANSTRT.

The REXX program IKANSTRT will call the REXX program IKANDDDL if the type identifies a source that is handled by the IDMS utility IDMSDDDL (for types Process, Record, File, Module, Tables). IKANSTRT is run with TYPE=IDMSDDDL.

The REXX program IKANSTRT will call the REXX program IKANMAP if the type identifies a source as a type Map. IKANSTRT is run with TYPE=MAP.

The REXX program IKANSTRT will call the REXX program IKANDIA if the type identifies a source as a type Dialog. IKANSTRT is run with TYPE=DIA.

The REXX program IKANSTRT will call the REXX program IKANADSA if the type identifies a source as a type Adsa Application. IKANSTRT is run with TYPE=ADSA.

The REXX program IKANSTRT will call the REXX program IKANSUBS if the type identifies a source as a type Subschema. This REXX program should extract the Schema name and version as well. IKANSTRT is run with TYPE=SUBSCHEMA.

The REXX program IKANSTRT will call the REXX program IKANDDDL if the type identifies a source as a type Schema. IKANSTRT is run with TYPE=SCHEMA.

The REXX programs mentioned before will write the Type, Name, Version, Date and Time information to the 'temp file'.

The REXX program IKANSTRT will call the REXX program IKANMAP if the type identifies a source as a type Map, but will not collect the date and time information.

The date/time information of the child for all types, except type=Map, is found by reading the source of the entity. The date and time is identified by the lines

```
*+       DATE CREATED IS      03/03/09 (mm/dd/yy)
*+       TIME CREATED IS      12240559 (hhmmssmm)
*+       DATE LAST UPDATED IS 05/29/09 (mm/dd/yy)
*+       TIME LAST UPDATED IS 14355854 (hhmmssmm)
```

If the entity has been created (1st time), the 'LAST UPDATED' lines will not show.

The date/time information of the child for type=Map and the Type, Name, Version, Date and Time of the parent is extracted by the COBOL program IKANIDD. For every child-parent combination the program adds an entry to the 'Entity File'.

Depending on the child, we need to navigate different paths. The relationships (and record definitions) in the IDD have been described in the IDMS Manual 'Dictionary Structure Reference Guide'. The relationships between the children and the parents can be viewed in the manual 'CA-IDMS Dictionary Diagram'

Type=Process, relation is Process-Dialog.

Read the MODULE-067 record and keep on reading until the MOD-NAME-067 and MOD-VER-067 matches the input and the LANG-067 equals 'PROCESS'. Then pick up the DATE-LU-067 and TIME-LU-067. If the date field is blank, pick the DATE-CREATED-067 instead.

Next, read the member record MODLST-055 and then read the owner record PROG-051. The Name of the parent is in PROG-NAME-051, the Version is in PROG-VER-051 and the Date is in DATE-LU-051 (or DATE-CREATED-051 if LU is empty). The Time should be set to '99999999'. To make sure we have a dialog, the PROG-FLAG1-051 field must contain X'10' and the PROG-FLAG3-051 field must contain X'04'

Then read the next member record MODLST-055 (if applicable) and pick up the next prog-051 record. Repeat this until there are no more member records MODLST-055 (end of set).

Type=Map, relationship is Map-Dialog

Read the MAP-098 record and keep on reading until the MAP-NAME-098 and MAP-VER-098 matches the input. Then pick up the DATE-LU-098 or MAP-DATE-098 or DATE-CREATED-098 (whatever date is most recent) and MAP-TIME-098 (if spaces, we should set an error, because the map has not been compiled yet).

Next, read the member record PROGMAP-126 and then read the owner record PROG-051. The name of the parent is in PROG-NAME-051, the Version is in PROG-VER-051, the Date is in DATE-LU-051 (or DATE-CREATED-051 if LU is empty) and the Time should be set to '99999999'. To make sure we have a dialog, the PROG-FLAG1-051 field must contain X'10' and the PROG-FLAG3-051 field must contain X'04'.

Then read the next member record PROGMAP-126 (if applicable) and pick up the next prog-051 record. Repeat this until there are no more member records PROGMAP-126 (end of set).

Type=Record, relationship is Record-Map, Record-Dialog, Record-Adsa, Record-Table

Read the RCDSYN-079 record and keep on reading until the RSYN-NAME-079 and RSYN-VER-079 matches the input. Then read the owner record SR-036 and pick up the DATE-LU-036 or DATE-CREATED-036 (whatever date is most recent) and TIME-LU-036.

Next, read member (from RCDSYN-079) record RCDCOPY-063 and, from there, read the owner record PROG-051.

If PROG-FLAG3-051 equals x'20', then the parent is a Map.

If PROG-FLAG3-051 equals x'02', then the parent is a Table.

If PROG-FLAG3-051 equals x'14' or x'04', then the parent is a Dialog.

If PROG-FLAG1-051 equals x'01', then the parent is an Adsa Application.

The name of the parent is in PROG-NAME-051, the Version is in PROG-VER-051, the Date is in DATE-LU-051 or in DATE-CREATED-051 whatever is most recent and the Time should be set to '99999999'.

Then read the next member record RCDCOPY-063 (if applicable) and pick up the next prog-051 record. Repeat this until there are no more member records RCDCOPY-063 (end of set).

Type=Record, relationship is Record-Schema

Read the RCDSYN-079 record and keep on reading until the RSYN-VER-079 matches the input. Then read the owner record SR-036 and pick up the DATE-LU-036 or DATE-CREATED-036 (whatever date is most recent) and TIME-LU-036.

Next, read member (from RCDSYN-079) record SRCD-113 and from there read the owner record S-010. The name of the parent is in S-NAM-010, the Version is in S-SER-010, the Date is in DATE-LU-010 (or DATE-CREATED-010 if LU-010 is spaces), and the Time is in TIME-LU-010 (or TIME-CREATED-010 if LU-010 is spaces).

If S-NAM-010 equals 'IDMSNTWK' or 'NON IDMS' we do not process the Schema.

Type=Table, relationship is Table-Map

Read the PROG-NAME-051 record and keep on reading until the PROG-VER-051 matches the input and the PROG-FLAG3-051 is x'02'. Version is in PROG-VER-051, Date is in DATE-LU-051 (or DATE-CREATED-051 if LU is empty), Time should be set to '99999999'.

Then read the next member record PROGMAP-126 (if applicable) and pick up the owner record MAP-098 record. The name of the Map is in MAP-NAME-098, the Version is in MAP-VER-098, date is in DATE-LU-098 or DATE-CREATED-098 or MAP-DATE-098. If the date in DATE-LU-098 is more recent than the date in MAP-DATE-098 we should set the time to '99999999', if not we take the time from MAP-TIME-098.

Repeat reading the next member record PROGMAP-126 and the owner MAP-098 until there are no more PROGMAP-126 records left (end of set).

Type=Subschema, relation is Subschema-Schema

The info supplied by REXX program IKANSUBS has already picked up the name of the Schema to which the Subschema is connected. So in this case we just need to extract the date and time of Subschema and Schema.

Read the SS-026 record and keep on reading until the S-NAME-026 matches the Schema name from the input and the S-SER-026 matches the Schema version from the input. The Date is in DATE-LU-026 or DATE-CREATED-026 if more recent. The Time is TIME-LU-026 or TIME-CREATED-026 if DATE-LU-026 is spaces. The S-SER-010 is also applicable for the Subschema Version.

Next, read the owner record S-010 that identifies the Schema. The Date is in DATE-LU-010 or DATE-CREATED-010 if more recent. The Time is TIME-LU-010 or TIME-CREATED-010 if DATE-LU-010 is spaces.

Schema

Schema is always a parent, so we just add Date and Time for this type.

Load Area Information

The Load Area Module is considered to be a parent for types Table, Dialog, Adsa Application, Subschema and Map. The type for a Load Area Module will be 'loadarea'.

To get the Load Area date/time info, the record LOADHDR-156 needs to be read.

Loadhdr-Modname-156 is the name of the load area module

Loadhdr-Vers-156 is the version of the load area module

Loadhdr-Date-156 is the creation date of the load area module

Loadhdr-Time-156 is the creation time of the load area module

2.2. Content Entity File

The 'entity file' has the following layout:

Pos1-15	TYPE=<childtype>
Pos16	Blank
Pos17-53	NAME=<childname>
Pos54	Blank
Pos55-66	VERSION=<childversion> with <childversion> being 4 characters (version 1 becomes 0001).
Pos67	Blank
Pos68-75	Last modification date of child(ccyymmdd)
Pos76	Blank
Pos77-84	Last modification time of child(hhmmss)
Pos85	Blank
Pos86-100	TYPE=<parenttype>
Pos101	Blank
Pos102-138	NAME=<parentname>
Pos139	Blank

Pos140-151	VERSION=<parentversion> with <parentversion> being 4 characters (version 1 becomes 0001)
Pos152	Blank
Pos153-160	Last modification date of parent(ccyymmdd)
Pos161	Blank
Pos162-169	Last modification time of parent(hhmmss)
Pos170-300	Blank

Copybook Entfile:

```
01   Entfile.
02   Ent-child-type-literal        PIC X(05) value 'TYPE='.
02   Ent-child-type                PIC X(10).
02   FILLER                        PIC X(01) value spaces.
02   Ent-child-name-literal        PIC X(05) value 'NAME='.
02   Ent-child-type                PIC X(32).
02   FILLER                        PIC X(01) value spaces.
02   Ent-child-ver-literal         PIC X(08) value 'VERSION='.
02   Ent-child-ver                 PIC X(04).
02   FILLER                        PIC X(01) value spaces.
02   Ent-child-date                PIC X(08).
02   FILLER                        PIC X(01) value spaces.
02   Ent-child-time                PIC X(06).
02   FILLER                        PIC X(01) value spaces.
02   Ent-parent-type-literal       PIC X(05) value 'TYPE='.
02   Ent-parent-type               PIC X(10).
02   FILLER                        PIC X(01) value spaces.
02   Ent-parent-name-literal       PIC X(05) value 'NAME='.
02   Ent-parent-type               PIC X(32).
02   FILLER                        PIC X(01) value spaces.
02   Ent-parent-ver-literal        PIC X(08) value 'VERSION='.
02   Ent-parent-ver                PIC X(04).
02   FILLER                        PIC X(01) value spaces.
02   Ent-parent-date               PIC X(08).
02   FILLER                        PIC X(01) value spaces.
02   Ent-parent-time               PIC X(06).
02   FILLER                        PIC X(115) value spaces.
```

2.3. Collection Check

This process can be submitted by the build when all IDD types have been processed by the 'collection' process. We set errors in case a Child has been modified more recently (later) than the last modification of the Parent. These checks are executed for the following relationships:

- Record-Dialog
- Record-Adsa application
- Record-Map The check Record-Map is not executed. The reason for this is the fact that the map compiler only changes the date and time information of the last modification in case of a 'critical change'. For more information, refer to Appendix A.4 of the “IDMS_Mapping_Facility” manual.
- Process-Dialog

- Record-Schema
- Map-Dialog
- Dialog-Loadmodule
- Adsa Application-Loadmodule
- Map-Loadmodule
- Subschema-Loadmodule
- This check can be executed by the REXX program IKANCHCK. Return code = 100 will be set when the check finds out that there is an 'out of sync' condition.

2.4. Correlation check

IDD entities cannot be migrated to a target IDD if that entity in the target IDD is connected to a map and/or a Schema. This is called 'not-IDD-owned'. The migration requires that the parent entities (Schema and/or Map) are also migrated.

To do the correlation check we need to run the following job:

1. Sort the 'entity file' on 'TYPE=REC' in column 1-8 into 'entity file2'.
2. Run Program IKANIDD which reads the sorted entity file and adds the parent information for this file based on the target IDD.
3. Sort the 'entity file' once more on 'TYPE=REC' in column 1-8 into 'entity file3'.
4. Compare 'entity file2' with 'entity file3'.
5. Any parent entity in 'entity file2' that is not present in 'entity file3' should set the build in error.

2.5. Collect Loads

This process will Punch the corresponding Load Area Module for every Child of Type=Map, Type=Dialog, Type=Adsa, Type=Table and Type=Subschema in the 'entity file'.

The output is saved for the build and should be the input for every next deploy.

2.6. Load IDD

This process will only be run during a deploy process. The result of the 'Collect Loads' process will be added in the target IDD Load Area.

If the customer wants the IDD sources to be loaded in the target IDD, all the IDD source entities have to be added in different steps. One step for every utility.

These steps are described below.

Usually a number of components is migrated at the same time (within one JCL step) using the IDMS utility IDMSDDDL. In such a job we might have a mix of component types like Records, Elements, (Adso) Processes, Programs, Maps, Modules, Files, Tables. As they can be interrelated, we encounter a chicken/egg problem. Example:

We have input like this.

```
REPLACE PROGRAM PGOTESDI
USING MAP PGOTESMA.
REPLACE MAP PGOTESMA.
```

In the example above we have the situation that the program is added first, but the map to which it likes to connect is not there yet as it is in the next line.

This situation can be solved by running this particular step twice to make sure that all related components are present.

The disadvantage is that the first job returns a return code of 8 which is not desirable. This can be solved by using the REXX program IKANCTL which is capable of resetting a return code to rc=4 when the maximum allowed return code (as specified in the maxrc=xx) is reached.

```
//IDD01 EXEC PGM=IKJEFT01,
//     PARM='%IKANCTL PGM=IDMSDDDL DICT=<dictname> CV=<cv> MAXRC=8'
//     COND=(4,LT)
```

In the example above the return code of 8 will be reset to return code=4.

The general flow is:

- Delete Subschemas
- Delete Schemas
- Delete Maps
- Add Records (and other idmsdddl stuff) twice
- Add Maps
- Add Schemas
- Add Subschemas

REXX program IKANDDDL

This program will process sources of the following format:

```
<action>
<iddtype> NAME IS <iddname> VERSION IS <version>
<iddtype specification>
<more iddtype specification>
```

The <action> may be ADD, MOD(IFY) or REP(LACE) and should be in Line 1 but is of no further interest. This might be used in the future to make sure that the correct action is used.

The <iddtype> starts on the second line and must be followed by 'NAME IS'. If that is the case, the 4th word identifies the name of the IDD entity. The 5th word should be 'VERSION' and might be situated on the next line. If word 6 equals 'IS' then word 7 is the version of this entity, else it is word6.

<iddtype> may be:

PROCESS, RECORD, FILE, TABLE or MODULE.

REXX program IKANMAP

This program will process sources of the following format:

```
PANEL <panelname> VERSION      1 DEVICES = (24X80, 32X80, 43X80, 27X132).
PFLD  OLMPF-0001 AT (   1,   6) ATTRIBUTES = (NUMERIC,PROTECTED) NODELIMIT
VALUE = ((   1) 'CB-ONLINE TEST SYSTEEM ENDEVOR').
<one or more pfld definitions>
MAP <mapname> VERSION     <mapversion> PANEL     <panelname> VERSION     1
USING ( (<recname1>   <ver1>) (<recname2> <ver2>)) RESET UNLOCK NOALARM NOPRT NLCR
NONPAGEABLE ON EDIT ERROR SOUND NOALARM.
MFLD OLMPF-0001 LITERAL.
<more mfld definitions>
```

The only thing that is important here is to find out the name of the Map. When you encounter the literal 'MAP' as word 1, then word 2 is the Map name and word 4 is the Map version.

REXX program IKANDIA

We assume that the 'source' of a Dialog is the output from an Adsorpts run for the dialog as the dialog itself has no source in the IDD.

This program will process sources of the following format with lrecl=133 or 121:

It will search for the literal ' DIALOG...:' to determine the name of the dialog (next word) and search for the literal ' VERS.:' to identify the version number (next word) of the dialog. The date and time are extracted by copying the words that follow the literal ' DATE.:' and 'TIME.:'

Example of Adsorpt output:

```
DIALOG...: SN0I01DI SCHEMA...: W35001SC  ONLINE MAP:  INPUT MAP.:  OUTPUT MAP:
    VERS.: 0001      VERS.: 0010    VERS.: 0000    VERS.:      VERS.:
    DATE.: 02/26/09  SUBSCHEMA: SN0510SU     DATE.:    DATE.:    DATE.:
    TIME.: 001459               TIME.:       TIME.:    TIME.:
    FDB SIZE:  5300  ENTRY PT: PREMAP    INPUT LABEL:        OUTPUT LABEL:
    ACCESS MOD: SN0I01DI          SUSPENSE FILE:
     **** OPTIONS ****
     SYMBOL TABLE....: NO  DIAGNOSTIC TABLES.: NO
     COBOL MOVE......: NO  ACTIVITY LOGGING..: YES
     MESSAGE PREFIX..: DC  RETRIEVAL LOCKING.: YES
     AUTOSTATUS......: YES
      SQL DATE........: SQL COMPLIANCE....:
     SQL TIME........:
     PAGEABLE MAP....: NO  PAGING OPTION.....:
      PAGING UPDATE...: PAGING AUTODISPLAY:
     PAGING BACKPAGE.:
          RECORD...: ADSO-STAT-DEF-REC               VERS.: 0001  AUTO-STATUS
          RECORD...: ADSO-APPLICATION-GLOBAL-RECORD  VERS.: 0001        WORK
          RECORD...: SN0I01RS                        VERS.: 0001        WORK
          RECORD...: W01001RS                        VERS.: 0001        WORK
          RECORD...: PA4020RS                        VERS.: 0001        WORK
      DECLARATION MOD: * * NO DECLARATION PROCESS  * *    VERS.:
      PREMAP PROCESS: SN0I01PP                            VERS.: 0001
```

REXX program IKANSUBS

This program will process sources of the following format:

```
SUBSCHEMA NAME IS <subschemaname> OF SCHEMA <schemaname> VERSION IS <version>
<action>
<subschema specification>
<more subschema specification>
```

The <action> will be ADD and should be in Line 1 but is of no further interest.

The literal 'OF' for the Schema is optional, so is the literal 'IS' for the Version.

REXX program IKANSCHM

This program will process sources of the following format:

```
<action>
SCHEMA NAME IS <schemaname> VERSION IS <version>
<schema specification>
<more schema specification>
```

The <action> will be ADD and should be in Line 1 but is of no further interest.

The <schemaname> starts at the second line and must be followed by 'NAME IS'. If that is the case, the 4th word identifies the name of the Schema. The 5th word should be 'VERSION' and might be situated in the next line. If word6 equals 'IS' then word 7 is the version of this entity, else it is word 6.

CHAPTER 3
Installation

Note: Before you install the IDD support modules, make sure you installed the IKAN ALM Common Utilities. During the installation of the IDD support modules these utilities are used in the installation procedures.

1. Extract the .zip file to a folder from where you want to install the IKAN ALM IDD support modules.

   The extract operation will create 6 files for the mainframe in the target folder.

Pc File	Mainframe File	Description
Cobol.seq	Hlq.COBOL.SEQ	COBOL Programs
Cntl.seq	Hlq.CNTL.SEQ	Installation JCL
Cntl.include.seq	Hlq.JCLINCL.SEQ	JCL Include members
Copybook.seq	Hlq.COPYBOOK.SEQ	COBOL Copybooks
Rexx.seq	na	
Rexx.bin	Hlq.REXX.SEQ	REXX Programs

   It will also create a folder zOS which will contain several files that are needed to set up the build and deploy process of IKAN ALM. See Implementation of zOS Support on page 16.

2. Upload the files to the mainframe, with the exception of the Rexx.seq file.

   The file Rexx.bin should be uploaded with the binary option. The default high level qualifier (HLQ) for the mainframe files used is ´IKANALM.IDD001´. If you pick your own high level qualifier make sure that you change the installation files accordingly.

3. Run an IEBUPDTE job to create a JCL library into which the IKANALM.IDD001.CNTL.SEQ file will be expanded.

   Example:

```
//ADCDMSTA JOB (5145,00000,2233,T),'IKAN',
//         MSGLEVEL=(1,1),MSGCLASS=X,
//         CLASS=A,REGION=0M,
//         NOTIFY=&SYSUID
//*
//         SET TARGET=IKAN ALM.IDD001.CNTL
//         SET UNIT=SYSDA
//*
```

```
//* ADD MEMBERS INTO PDS
//************************************* //ADDCNTL EXEC PGM=IEBUPDTE,
//       PARM=NEW,
//       COND=(4,LT)
//SYSUT1   DD DUMMY
//SYSUT2   DD DISP=(MOD,CATLG,CATLG),,DSN=&TARGET,
//                 UNIT=&UNIT,SPACE=(CYL,(2,2,180)),
//                 LRECL=80,BLKSIZE=0,RECFM=FB,DSORG=PO,
//                 DSNTYPE=LIBRARY
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DISP=SHR,DSN=&TARGET..SEQ
```

4. Modify the member INSTALL in the JCL library created in the previous step to meet your dataset prefix choice.

   By default the member INSTALL will allocate new libraries for COBOL, Copybooks, REXX, JCL includes, Loadlib, Object library and a listing library.

   The datasets will be allocated as <prefix>.<type> with <prefix> being the general prefix for the datasets to be used, and <type> being COBOL for COBOL Programs, COPYBOOK for COBOL Copybooks, REXX for REXX programs, JCLINCL for Jcl include members, LOADLIB for Loadlibrary, OBJLIB for Object library and COBOL.LISTLIB for the Listing of COBOL program compilations.

   If you want to change the <type>, modify the SET statements accordingly. // SET COBOL=COBOL will set the <type> to COBOL for COBOL programs // SET COPYBOOK=COPYBOOK will set the <type> to COPYBOOK for COBOL Copybooks .

```
//  SET REXX=REXX          will set the <type> to REXX for REXX programs
//  SET JCLINCL=JCLINCL    will set the <type> to JCLINCL for Jcl include members
//  SET LOADLIB=LOADLIB    will allocate a Pds as <prefix>.LOADLIB
//  SET OBJLIB=OBJLIB      will allocate a Pds as <prefix>.OBJLIB
```

   The SET statements below indicate the uploaded sequential files for the several types. The dataset read will be <prefix>.<type>.SEQ

```
//  SET COBSEQ=COBOL.SEQ        UPLOADED Cobol
//  SET COPYSEQ=COPYBOOK.SEQ    UPLOADED Copybook
//  SET REXXSEQ=REXX.SEQ        UPLOADED REXX PROGRAMS
//  SET INCLSEQ=JCLINCL.SEQ     UPLOADED JCL INCLUDE
```

5. Modify the REXX program IKANCTL to specify a table of IDMS central version numbers and the corresponding sysctl files and dmcl names.

   The REXX program IKANCTL has been installed together with the IKAN Common Utilities. A table needs to be defined that describes all the supported Central Version numbers, their corresponding sysctl file names and their corresponding dmcl names. The table starts at about line 33 of the REXX program.

   See the next three lines below:

```
cv_table.1 = '20 ADCDMST.TEST.SYSCTL20 DMCLCV20'
cv_table.2 = '<cv_number2> <sysctl_file2> <dmcl_name2>'
cv_table.0 = 2        /* Adjust the number of supported cv_numbers */
```

   Every table entry (cv_table.x) contains 3 words:

   'cv_table.1 = '<cv_number1> <sysctl_file1> <dmcl_name1>' all separated by at least one space.

   The first word in a table entry (cv_number1) defines the cv number supported. The second word in a table entry (sysctl_file1) describes the name of the Sysctl file that is needed to communicate with the cv (in central mode) which is described in word 1. The third word in a table entry (dmcl_name1) describes the name of the dmcl load module that is needed when running in local mode.

```
'cv_table.0 = 2'
```

The statement above defines the number of entries in the table. In this example we defined 2 entries (default value).

For every combination of cv number, dictionary name and dmcl name, we need 1 entry in the table. So if you need to support 3 central versions, then you will have:

```
cv_table.1 = 'info for central version 1'
cv_table.2 = 'info for central version 2'
cv_table.3 = 'info for central version 3'
cv_table.0=3
```

6. Modify the member CMPIDMS and remove ddname IKANTEST from the PREIDMS step.

7. Run the Type JCL members

   - Member CMPNOIDM to compile the non-IDMS COBOL programs
   - Member CMPIDMS to compile the IDMS COBOL programs. This JCL member requires you to specify the Cv number against which you want to precompile. You also have to specify the name of the dictionary that contains the IDD records descriptions. The programs to be compiled in this member all use Subschema Idmsnwka and Schema Idmsntwk.

CHAPTER 4

Implementation of zOS Support

For support of mainframe components, including IDD components, the following files are delivered which will need user-specific modifications:

Note: The Ant scripts that are running during the Build and Deploy processes require the existence of property files and a number of parameters.

4.1. Property Files

The **Common.property file** contains properties that are valid for the whole IKAN ALM project. The following table gives an overview of the required properties and their meaning.

Note: All property names are in lowercase. However, Word will set the first character to uppercase.

Property names and default values	Description
cobol.ext=cbl	cobol.ext identifies the extension of COBOL programs
asm.ext=asm	asm.ext identifies the extension of Assembler programs.
macro.ext=mac	macro.ext identifies the extension of Assembler Macros.
cobolcopy.ext=cpy	cobolcopy.ext identifies the extension of COBOL Copybooks.
idmsdddl.ext=prc,rec,fil,mod,tab	idmsdddl.ext is a comma-separated list that identifies all the extensions of Types that need to be processed by IDMSDDDL. RECORD=rec, MODULE=mod,PROCESS=prc TABLE=tab, FILE=fil.
map.ext=map	map.ext identifies the extension of Type MAP
subschema.ext=sub	subschema.ext identifies the extension of Type SUBSCHEMA
schema.ext=sch	schema.ext identifies the extension of Type SCHEMA
dialog.ext=dia	# dialog.ext identifies the extension of Type DIALOG
adsa.ext=adsa	# adsa.ext identifies the extension of Type ADSA
ikan.internal.idmsdb=//*	For IKAN's internal use only.
ikan.include=<jcl include library>	ikan.include identifies the include Library from which JCL members will be included

Property names and default values	Description
ikan.test=//*	For IKAN's internal use only.

The **<ikanalm-level>.property file** contains properties that are valid for a specific IKAN ALM Level. Depending on the Level on which a build and/or deploy process is run, the process will load the property file that is identified by the name of the Level followed by '.property'. For example: if the Level is called 'Dev', the build/deploy script expects a File called 'Dev.property' (case-sensitive).

The 'this.'- properties are valid for the current environment. The 'next.'-properties are valid for the next environment (in regard of the Lifecycle).

For example: IKAN ALM has a Life-cycle defined as Dev, Syst, Qa and Prod. If the build runs in the Dev environment, the this.-properties will define the properties valid for the Dev environment and the next.-properties will identify the values that are valid for the Syst environment.

If a property is suffixed by a number, this identifies concatenations in compile and link edit JCL jobs. For example:

```
this.env2=SYST
this.env1=DEV
this.env3=QA
this.env4=PROD
```

Property names	Description
this.prefix=IKANALM.IKANIDD	this.prefix identifies the prefix to be used for dataset allocation in all environments
this.cobol.srclib= ${this.prefix}.${this.env1}.COBOL	this.cobol.srclib identifies the COBOL source library where the COBOL sources will reside
this.jcl.srclib= ${this.prefix}.${this.env1}.CNTL	this.jcl.srclib identifies the JCL library that will be used for submittng JCL
this.cobol.csyslib1= ${this.prefix}.${this.env1}.COPYBOOK this.cobol.csyslib2= ${this.prefix}.${this.env2}.COPYBOOK this.cobol.csyslib3=$ {this.prefix}.${this.env3}.COPYBOOK this.cobol.csyslib4= ${this.prefix}.${this.env4}.COPYBOOK	this.cobol.csyslibX identifies the Copybook concatenation needed to compile COBOL programs
this.asm.csyslib1= ${this.prefix}.${this.env1}.MACRO this.asm.csyslib2= ${this.prefix}.${this.env2}.MACRO this.asm.csyslib3= ${this.prefix}.${this.env3}.MACRO this.asm.csyslib4= ${this.prefix}.${this.env4}.MACRO	this.asm.csyslibX identifies the Macro concatenation needed to assemble Assembler programs
this.batch.loadlib= ${this.prefix}.${this.env1}.LOADLIB	this.batch.loadlib identifies the target batch loadlibrary for this environment where linked modules will be stored
this.cics.loadlib= ${this.prefix}.${this.env1}.CICS.LOADLIB	this.cics.loadlib identifies the target Cics loadlibrary for this environment where linkedit modules will be stored

Property names	Description
this.idms.loadlib= ${this.prefix}.${this.env1}.IDMS.LOADLIB	this.idms.loadlib identifies the target loadlibrary for this environment where IDMS linked modules will be stored
this.objlib1= ${this.prefix}.${this.env1}.OBJLIB this.objlib2= ${this.prefix}.${this.env1}.OBJLIB this.objlib3= ${this.prefix}.${this.env1}.OBJLIB this.objlib4= ${this.prefix}.${this.env1}.OBJLIB	this.objlib1 identifies the concatenation of Object libraries for this environment when linking modules
this.dbrmlib= ${this.prefix}.${this.env1}.DBRMLIB	this.dbrmlib identifies the Dbrm library for this environment where the output from the DB2 precompile will be stored
this.cv=	identifies the IDMS system against which a JCL job should run for this environment
this.dictname=	identifies the IDMS IDD dictionary against which a JCL job should run for this environment
this.dbname=	identifies the IDMS IDD dbname against which a JCL job should run for this environment

For the 'next.'- properties, the same description applies as for the corresponding 'this.'-properties but then for the next environment as defined in the IKAN ALM Life-cycle.

next.cv=	
next.dictname=	
next.dbname=	

4.2. Parameters

Parameter	Description	Where defined
Script.location	The folder where the scripts are stored. This folder must contain the subfolders ´Models´, ´Properties´ and 'Utilities'	Build Environment
Zos.password	The password for the FTP server	Build Environment as secured parameter
Zos.url	The IP-address of the Mainframe FTP server	Build Environment

4.3. Ant scripts

BuildZos.xml

This Ant script is run by IKAN ALM during the build process. The default functions allow the compilation of COBOL programs and the support of IDD entities. To run correctly, parameters need to be defined for the Build Environment (Parameters (page 18)).

- The script will use imported Ant scripts for the IDD support and FTP handling.
- Target PutFtp is called to copy a source to a zOS Dataset (ZosFtp.xml (page 19)).
- Target SubmitJcl is called to submit a JCL job (ZosFtp.xml (page 19))
- If A COBOL compile fails, the process continues until all COBOL sources have been processed.
- At the end of the Ant script, a check for errors is done. In case errors are encountered, the build fails. By default, the target BuildZosIdms is called, which is located in the BuildZosIdms.xml script for supporting IDD migrations (BuildZosIdms.xml (page 20)). Error messages will show in the IKAN ALM log and the IKAN ALM archive dataset in the Listing folder.

DeployZos.xml

This script calls the target DeployZosIdms which is located in DeployZosIdms.xml (BuildZosIdms.xml (page 20)).

DeployZosIdms.xml

This script is used to deploy IDD components to a target IDD. It runs targets ProcessInit (Inits the Process), UploadSources (Put IDD sources from the IKANALM archive to Mainframe datasets), SubmitMigrate (Submits JCL to add sources into the target IDD) and ClearFiles (deletes the files allocated in ProcessInit).

ZosFtp.xml

- Target PutFtp puts sources from the PC to a zOS dataset. It needs the following properties: 'put.target.srcfile' (the name of the zOS dataset which receives a source) and 'local.putfile' (the name of the PC file to be put). If the put fails, the process fails.
- Target SubmitJcl submits JCL jobs and determines the success of the JCL job. It needs the following properties: 'local.ftpsubmitbody' (the name of the model that contains the FTp commands to run a zOS JCL job), 'local.ftpsubmit.cmd' (the name of the model that contains the commands to convert a JCL model to real JCL, run the FTP commands to submit a job and the commands to determine the success of a submission) 'jcl.model' (the name of the JCL model to be submitted) and 'zos.member' (the name of the zOS member in the JCL library that will contain the JCL to be run). If the submission fails, the property submit.ok is set to false, else it is set to true. The caller determines whether the process should be stopped.
- Target GetFtp copies a source from a zOS dataset to a PC File. It needs the following properties: 'get. srcfile' (the name of zOS Dataset where the source resides), 'local.getfile' (the name of the PC file that receives the zOS dataset) and 'local.ftpgetbody' (the name of the model that contains the FTP commands). If the get fails, the process fails.

BuildZosIdms.xml

This script needs to collect all the IDD sources that need to be migrated to a target IDD during the deploy process. Before a proposed migration is considered to be consistent, several checks need to be executed. Several targets are to be executed:

- 'ProcessInit' allocates datasets that are needed for the process. All the files allocated are using the IKAN ALM Build number to make sure the datasets are unique. The files allocated are deleted in the target Clearfiles. To this end, the script is using the models Jclinit (Allocation) and Jclclear (Deallocation)
- 'ProcessDddl' processes all sources that need to be processed by the IDMS utility Idmsdddl. This is identified by the property idmsdddl.ext which is a comma-separated list of extensions. For every file that is included in the selection, a JCL job is submitted that adds the relevant data of the IDD entity to the 'Entfile'. See Collect Information on page 4.
- 'ProcessMap' processes all sources of the type Map and adds the relevant data of the IDD entity to the 'Entfile'.
- 'ProcessDia' processes all sources of the type Dialog and adds the relevant data of the IDD entity to the 'Entfile'.
- 'ProcessAdsa' processes all sources of the type Adsa Application and adds the relevant data of the IDD entity to the 'Entfile'.
- 'ProcessSchem' processes all sources of the type Schema and adds the relevant data of the IDD entity to the 'Entfile'.
- 'ProcessSub' processes all sources of the type Subschema and adds the relevant data of the IDD entity to the 'Entfile'.
- 'ProcessCollectCheck' processes all the data of the IDD entities in the 'Entfile'. Consistency checks are executed by submitting the model CollectCheck.model. This JCL will check the child parent information regarding the sequence of modification. (A parent must be modified after the child has been modified). Also migration stoppers are checked. If one of the checks fails, the process fails.
- 'CollectIdmsData' processes all the files that contain data (Sources and Loads) that is needed for the deployment of IDD entities.

4.4. Models

Assemble.model	This model contains JCL that will Assemble Assembler Programs.
COBOLCompile.model	This model contains JCL that will Compile COBOL Programs.
Collect.model	This model contains JCL that collects information regarding IDD components.
Collectcheck.model	This model contains JCL that collects information concerning the consistency of the IDD immigration.
FtpsubmitJcl.cmd	This command file creates a JCL, submits the JCL, receives the Job output and will determine whether the JCL job was successful (jchck010.exe) or not. For more information, refer to the chapter *Windows Programs* in the *IKAN Common Utilities* documentation.
Get.ftpbody	This model contains FTP commands to get sources from the mainframe to the PC.

GetBinary.ftpbody	This model contains FTP commands to get sources from the mainframe to the PC in binary format.
Jclclear.model	This model contains JCL to delete the catalogued datasets that were needed on a temporary basis.
Jclinit.model	This model contains JCL to allocate catalogued datasets that are needed on a temporary basis.
Loadidd.model	Not used
Migrate.model	This model contains JCL to migrate IDD components into a target IDD.
Put.ftpbody	This model contains FTP commands to put sources from the PC to the mainframe.
PutBinary.ftpbody	This model contains FTP commands to put sources from the PC to the mainframe in binary mode.
PutFtp.cmd	not used
Submit.ftpbody	

4.5. Properties

Common.property

Environment#1.property

Environment#2.property etcetera

4.6. Utilities

jchck000.exe is used to pre-process ´JCL´. For a description, refer to the document the 'IKAN ALM Common Utilities'.

jchck010.exe is used to determine whether a JCL job was successful. For a description, refer to the document "IKAN ALM Common Utilities".

Some additional components need modification at the customer's site:

- Modify the REXX program IKANCTL (see Installation (page 13)) to specify a table of IDMS central version numbers and the corresponding sysctl files and dmcl names (for more information, refer to the document '"KAN ALM Common Utilities").
- Modify the JCL models in folder zOS/Models which are used in the IKAN ALM build and deploy scripts.
- Concatenation character in REXX programs If you uploaded the REXX programs in binary format, you can skip this section and proceed with the next step 'Decide on migration strategy'. Some REXX programs use a concatenation character to combine two strings without a space in between. On the mainframe the character is always X'4F' (hexadecimal 4F, decimal 79). When uploading the REXX programs from the PC to the mainframe, this character is not always translated correctly during ASCII-EBCDIC translation. The character on the PC will be '||'. In the REXX program, you will see the following line /*. Make sure that '||' equals X'4F' */ . Probably the '||' characters will have changed to something else. The actual hex value of those characters can be revealed by using the 'HEX ON' command in ISPF mode. Let us assume that the characters '||' have been changed to '!!' during the upload. In that case, we need to execute the ISPF edit command "change ! X'4F' all" and save the REXX program. This issue relates to the REXX programs IKANADSA, IKANDDDL, IKANDIA and IKANSUBS.

- Decide on migration strategy.

  The IDD migration concerns the migration of Load Area Modules, IDD sources and Link Edit of the Load Area modules. By default, the IDD entities are migrated with both sources and loads. If you don't want to migrate sources but only loads, the JCl in the model migrate.model should be adapted to migrate only the loads. The same applies if you want to link edit the Load Area Modules to a Load library.

- Decide on check strategy.

  In the JCL in CollectionCheck.model, the customer should decide which checks should be active. By default, there is a comparison between child date and time and parent date and time. If the child date and time is more recent than the parent date and time, the parent should be regenerated. The default child-parent checks are:

PRCS-DIA='TRUE'	Process-Dia check
REC-DIA='TRUE'	Record-Dia check
REC-ADSA='TRUE'	Record-Adsa check
REC-MAP='FALSE'	NO Record-Map check
MAP-DIA='TRUE'	Map-Dia check
MAP-LOAD='TRUE'	Map-Load check
DIA-LOAD='TRUE'	Dia-Load check
ADSA-LOAD='TRUE'	Adsa-Load check
SUB-LOAD='TRUE'	Subschema-Load check

Note: By setting the value above to 'FALSE' the corresponding check will not be executed.

CHAPTER 5
Components Overview

5.1. REXX Programs

IKANPNCH	This REXX program will generate statements for 'New copies', 'Delete Schema', 'Mod Schema <name>. Validate.', 'Delete Subschema', ''Mod Subschema <name>. Validate.', 'Delete Map', 'Generate Map', 'Punch Load Module' and 'Adsotatu' statements.
IKANSTRT	This REXX program will call other REXX programs to extract name, version, type, date and time from the source of an IDD entity. Which program to call, depends on the parameter 'TYPE=<iddtype>'
IKANDDDL	This REXX program is called by IkanSTRT and will process the source of an IDD entity of type Process, Record, Module, Program, Element, Schema, File or Table to extract the type, name, version, date and time.
IKANSUBS	This REXX program is called by IkanSTRT and will process the source of an IDD entity of type Subschema to extract the type, name, version, date and time.
IKANMAP	This REXX program is called by IkanSTRT and will process the source of an IDD entity of type Map to extract the type, name and version.
IKANDIA	This REXX program is called by IkanSTRT and will process the 'source' of an IDD entity of type Dialog to extract the type, name, version, date and time. Source means an Adso Report from the dialog.
IKANADSA	This REXX program is called by IkanSTRT and will process the 'source' of an IDD entity of type Adsa Application to extract the type, name, version, date and time. Source is an Adso Report from the Adsa Application.
IKANSLST	This REXX program will check the output from a Idmsdddl execution. It screens certain errors. Some errors are not important, others are. If an important error is encountered, the fail flag is raised (high return code).
IKANCTL	For more information, refer to the document ”IKAN ALM Common Utilities”.
IKANCOMT	This REXX program will add 'Commit' statements to sources that need to be processed by an IDMS utility (like idmsdddl, idmschem etc.)

IKANLKED	This REXX program will read the output from the Idmsdddl utility which was created by the statements 'PUNCH LOAD MODULE NAME IS <loadname> WITH SYN. The output contains 'ADD LOAD …' statements and Object decks. The REXX will create statements for the IEBUPDTE program to add the object decks as members in an Object PDS. It will also create ' INCLUDE OBJLIB(member)' and "NAME member (R)" statements to be processed by the Linkage editor which will create loadlib members. This REXX program can be used if the customer likes to linkedit modules from the IDD Load Area
IKANCHCK	This REXX program will compare date/time information of a child and a parent. If the child date/time info is more recent than the parent date/time, it will set a high return code to stop the migration process with an appropriate message indicating that the parent should be regenerated.

5.2. COBOL Copybooks

IKANENT	This is a COBOL Copybook representation of the 'Entfile' that is used in the migration Process.
IKANCOMM	This is a COBOL Copybook representation of the communication block between IKAN programs to pass on values etc.
IKANPGMS	This is a COBOL Copybook representation of the names of IKAN programs that can be called dynamically.
IKANBITW	This is a COBOL Copybook defining working storage fields that are needed by copybook IKANBITP which will test bit values.
IKANBITP	This is a COBOL Copybook to be used in the Procedure Division. It will set the value of bit values to 'y'. For example:. if the first bit of a byte is '1' (X'80'), field x80 will be set to 'y'. If a byte contains X'81', fields x80 and x01 will be set to 'y'.

5.3. COBOL Programs

IKANIDD	This program will call all necessary programs that are needed in an IDD migration depending on the type to process.
IKANFILE	This program is called by the IKANIDD program and will collect Date information of an IDD File.
IKANPRCS	This program is called by the IKANIDD program and will collect Date and Time of an Adso Process and will collect parent information on all related Dialogs.
IKANMODL	This program is called by the IKANIDD program and will collect Date and Time of a Module (not being an Adso Process) as well as date and time information of the Module.
IKANSCHM	This program is called by the IKANIDD program and will collect Date and Time of a Schema as well as the date and time of the schema.

IKANSUBS	This program is called by the IKANIDD program and will collect Date and Time of a Subschema and as well as parent information on all related Schemas. It also will collect date/time information about the LoadAreaModule.
IKANMAPS	This program is called by the IKANIDD program and will collect Date and Time of a Map and the relations Map-Dialog. It will also collect date/time information about the LoadAreaModule.
IKANREC	This program is called by the IKANIDD program and will collect Date and Time of a Record and the relations Rec-Map, Rec-Dialog, Rec-Adsa, Rec-Table, Rec-Schema.
IKAN ALM_IDD _Cobol_IkanLOAD.txt	This program is called by the IKANIDD program and will collect Date and Time of the LoadAreaModule of Dialogs and Adsa Applications.
IKANTABL	This program is called by the IKANIDD program and will collect Date and Time of a Edit/Code Table and the relations Table-Map. It will also collect date/time information about the LoadAreaModule.
IKANPRMG	This program writes the content of a parmcard to ddname Parmout. The file allocated on ddname Parmout should have a logical record length of 80 bytes (Lrecl=80)