IKAN ALM Common Utilities

Release 1.0

February 2019

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

Table of Contents

CHAPTER 1
Introduction

This manual describes the installation procedure and functionalities of the *IKAN ALM Common Utilities*.

The software is delivered as a .zip file that needs to be extracted to a user folder.

IKAN Development assumes that the reader of this document is familiar with IBM mainframe concepts and terminology.

CHAPTER 2
Overview .zip File

The .zip file contains several files needed for the installation and use of the Common Utilities.

All files needed for the Mainframe are distributed in IEUPDTE format. The IBM IEBUPDTE utility adds members from a sequential file (using the IEBUPDTE command) into a PDS library.

Asm.seq	This file contains the Assembler programs in source format for the Mainframe.
Cobol.seq	This file contains the COBOL programs in source format for the Mainframe.
Cntl.seq	This file contains JCL to run several installation jobs.
Jclinclude.seq	This file contains files to be used as JCL includes or as parameters cards to be used in procedures.
Macro.seq	This file contains Assembler Macro sources to be used in the assembly of the Assembler programs in the file Asm.seq.
Rexx.seq	This file contains REXX programs in ASCII format.
Rexx.bin	This file contains REXX programs in Binary format.
Jchck000.exe	This file is an MS-Windows executable that is used to enable IKAN ALM to run JCL jobs on the Mainframe.
Jchck010.exe	This file is an MS-Windows executable that is used to enable IKAN ALM to run JCL jobs on the Mainframe. The Jchck000 and Jchck010 programs are working together.
Install_Common_Utilities.pdf	This document.

CHAPTER 3
Installation

3.1. Extracting the .zip File

Extract the .zip file to a folder from where you want to install the *IKAN ALM Common Utilities*.

The extract operation will create the following files in the target folder:

Asm.seq

Cobol.seq

Cntl.seq

Jclinclude.seq

Macro.seq

Rexx.seq

Rexx.bin

Jchck000.exe

Jchck010.exe

Install_Common_Utilities.pdf

3.2. Uploading the Mainframe Files

Upload the files to the Mainframe with the exception of the *Rexx.seq* file and the **.exe* files. The file *Rexx.bin* should be uploaded with the binary option.

The installation jobs expect the files to be uploaded to files that use the prefix IKANALM.IKAN.COMMON and the suffix ASM.SEQ, COBOL.SEQ, CNTL.SEQ, JCLINCL.SEQ, MACRO.SEQ and REXX.BIN for the respective types.

If you change the prefix and/or the suffix, you have to change the JCL member INSTALL accordingly after having run the next step.

3.3. Running IEBUPDTE

Run *IEBUPDTE* to create a JCL library into which the IKANALM.IKAN.COMMON.CNTL.SEQ file will be expanded. Make sure to change the Jobcard to your site's standards (see the example below).

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

3.4. Modifying Member Install

Modify the Member Install in the JCL library (by default IKANALM.IKAN.COMMON.CNTL) which was created in the previous step, in order to meet your dataset prefix choice.

By default, the Member Install will allocate new libraries for Assembler, COBOL, Macros, REXX, JCL includes, Load library, Object library and a Listing library for COBOL and Assembler. The datasets will be allocated as `<prefix>.<type>` with `<prefix>` being the general prefix to be used for the datasets and `<type>` being MACRO for Assembler Macros, ASM for Assembler programs, COBOL for COBOL programs, REXX for REXX programs, JCLINCL for JCL include members, LOADLIB for Load Library, OBJLIB for Object library, ASM.LISTLIB for the Assembler listings and COBOL.LISTLIB for the COBOL compile listings.

If you want to change the <type>, modify the SET statements accordingly.

// SET MACLIB=MACRO	will set the <type> to MACRO for Assembler Macros
// SET MACLIB=COBOL	will set the <type> to COBOL for COBOL programs
// SET ASSEM=ASM	will set the <type> to ASM for Assembler Programs
// 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 MACSEQ=MACRO.SEQ	UPLOADED MACROS
// SET ASSSEQ=ASM.SEQ	UPLOADED ASSEMBLER PROGRAMS
// SET COBSEQ=COBOL.SEQ	UPLOADED COBOL PROGRAMS
// SET REXXSEQ=REXX.BIN	UPLOADED REXX PROGRAMS
// SET INCLSEQ=JCLINCL.SEQ	UPLOADED JCL INCLUDE

When you are done with Member Install, run it. This will allocate and populate the mentioned datasets as described in the previous step. The job should end with a return code of maximum 4.

3.5. Running Member ASSEMBLE

Run Member *ASSEMBLE* from the JCL library to assemble the Assembler Programs. If you changed the Library name settings, make sure to change the Member *ASSEMBLE* accordingly.

3.6. Running Member COMPILE

Run Member *COMPILE* from the JCL library to compile the COBOL Programs. If you changed the Library name settings, make sure to change the member *COMPILE* accordingly.

3.7. Storing the exe Files

Copy the **.exe* files to a directory that you will be using in the IKAN ALM build and deploy scripts.

CHAPTER 4
Common Programs

This section describes the functionality of the Common Programs that reside in a z/OS load library.

4.1. IKANHEX

The program *IKANHEX* is a subroutine that is called by other programs. The program converts a string into hexadecimal characters and vice versa. The *Parameter4* option allows converting from character to hexadecimal or from hexadecimal to character.

Example:

The character string 'ABC' will be converted to the string 'C1C2C3' with the option 'HEXCHAR'. This can be used if the content of a storage field needs to be displayed in hexadecimal. In the example above, the output field that receives the conversion string needs to be defined twice as long as the input string.

The string 'F1F2' will be converted to X'F1F2' (12) with the option 'CHARHEX'.

The program needs 4 parameters:

Parameter1	The area that contains the string to be converted
Parameter2	The area that will receive the value to be converted
Parameter3	The number of bytes to convert. This should be a full word which contains the number of bytes to convert in binary format (in COBOL this is a PIC 9(8) COMP field)
Parameter4	The string 'HEXCHAR' or 'CHARHEX' to indicate the conversion method.

COBOL example:

```
CALL 'IKANHEX' USING FIELD1 FIELD2 NUMBER-OF-BYTES METHOD
```

With:

```
FIELD1 PIC X(2)  VALUE 'MY'.
FIELD2 PIC X(4).
NUMBER-OF-BYTES PIC 9(8) COMP.
METHOD PIC X(8) VALUE 'HEXCHAR'
```

The example above will place the string 'D4E8' in FIELD2. The string 'MY' is in hexadecimal 'D4E8'.

The user is responsible for providing the correct input. The default conversion method is 'CHARHEX'.

4.2. IKANPDSL

The program *IKANPDSL* lists all the members of a specified PDS and can run as a stand-alone program in a JCL job. Refer to the example below which creates a member list of the PDS IKANALM.IKAN.COMMON.ASM.

```
//ADCDMSTA JOB (5145,00000,2233,T),'IKAN',
//         MSGLEVEL=(1,1),MSGCLASS=X,
//         CLASS=A,REGION=0M,
//         NOTIFY=&SYSUID
//*
//         SET STEPLIB=IKANALM.IKAN.COMMON.LOADLIB
//*
//IKANPDSL EXEC PGM=IKANPDSL
//STEPLIB  DD DISP=SHR,DSN=&STEPLIB1
//IKANRPT  DD SYSOUT=*,LRECL=121,BLKSIZE=12100,RECFM=FB
//PDSLIST  DD SYSOUT=*,LRECL=121,BLKSIZE=12100,RECFM=FB
//SYSIN    DD DUMMY
//PDSIN    DD DISP=SHR,DSN=IKANALM.IKAN.COMMON.ASM
```

DDname PDSIN	describes the PDS from which a member listing should be produced.
DDname PDSLIST	receives the list of members, including the TTR information
DDname IKANRPT	is for future use
DDname SYSIN	must be allocated, but is not used.

For each member, a record will be written to DDname IKANRPT with the following layout:

Position 1-8:	Member name
Position9:	Space
Position10-17:	'NOALIAS' if the member is not an alias or '*ALIAS' if it is
Position18:	Space
Position19-21:	TTR (Track Address)
Position22:	Space
Position23-84:	User Data of directory entry

4.3. IKANSTWA

The program *IKANSTWA* will update the members in a PDS with user supplied User Data (maximum 62 bytes)

DDname MBRIN	contains a list of members to be updated with the User Data which in DDname USERDATA. The records in DDname MBRIN must have the same lay-out as the one produced by the program IKANPDSL. The TTR address listed by IKANPDSL is used to 'Stow' the member on the same track address.
DDname PDSLIB	contains the PDS in which the members reside that need User Data modification.
DDname IKANRPT	is for future use.

In the example below, a member list is created in step IKANPDSL for the PDS ADCDMST.TEST.CNTL.IKANSTWU. In the second step, IKANSTWA, the list of members which are specified in dataset &&PDSLIST is updated with the User Data 'V01-IKAN BUILD 11-05-2011 11:02' in the PDS ADCDMST.TEST.CNTL.IKANSTWU.

```
//*
//*-------------------------------------------------------
//*--- LIST THE PDS CONTENT
//*-------------------------------------------------------
//IKANPDSL EXEC PGM=IKANPDSL
//IKANRPT  DD SYSOUT=*,
//         LRECL=121,RECFM=FB,BLKSIZE=0,DSORG=PS
//PDSLIST  DD DISP=(,PASS),DSN=&&PDSLIST,
//         UNIT=SYSDA,SPACE=(CYL,(2,2,)),
//         LRECL=80,RECFM=FB,BLKSIZE=0,DSORG=PS
//SYSIN    DD DUMMY
//PDSIN    DD DISP=SHR,DSN=ADCDMST.TEST.CNTL.IKANSTWU
//*
//*-------------------------------------------------------
//*--- SET NEW USERDATA
//*-------------------------------------------------------
//IKANSTWA EXEC PGM=IKANSTWA,
//         COND=(4,LT)
//IKANRPT  DD SYSOUT=*,
//         LRECL=121,RECFM=FB,BLKSIZE=0,DSORG=PS
//MBRIN    DD DISP=(OLD,PASS),DSN=&&PDSLIST
//PDSLIB   DD DISP=SHR,DSN=ADCDMST.TEST.CNTL.IKANSTWU
//USERDATA DD *
V01-IKAN BUILD 11-05-2011 11:02
```

4.4. IKANSTWU

The program *IKANSTWU* will update one member in a PDS with user-supplied User Data (maximum 62 bytes)

Basically, the operation is the same as for the program IKANSTWA with this difference that IKANSTWU operates only on 1 member. The name of the member to be updated should be passed to the program through a parameter that describes the length of the member name passed in the first two bytes of the parameter, followed by the member name itself. Because the first two bytes describe the length of the member name, it is also possible to run this program as a batch program in JCL (see the example later on).

DDname PDSLIB	contains the PDS in which the member resides that needs a User Data modification.
DDname USERDATA	contains the User Data that should be added to the directory entry.

In the example below, the member MYMEMBER is updated with the User Data supplied in DDname USERDATA ('V01-IKAN BUILD 11-05-2011 11:02').

```
//*
//*-------------------------------------------------------
//*--- SET NEW USERDATA
//*-------------------------------------------------------
//IKANSTWU EXEC PGM=IKANSTWU,
//          PARM='MYMEMBER ',
//          COND=(4,LT)
//PDSLIB    DD DISP=SHR,DSN=ADCDMST.TEST.CNTL.IKANSTWU
//USERDATA DD *
V01-IKAN BUILD 11-05-2011 11:02
```

The program IKANSTWU can also be called with the following call (COBOL example):

```
CALL 'IKANSTWU' USING PARM-DATA.
```

With:

```
01 PARM-DATA.
 03  PARM-LENGTH  PIC 9(4) COMP VALUE 8.
 03  PARM-MEMBER PIC X(8) VALUE 'MYMEMBER'.
```

4.5. IKANTIOT

The program *IKANTIOT* is a subroutine that can be called by other programs to check whether a specific DDname has been allocated. For that, it checks the z/OS Task Input Output Table (TIOT).

The program accepts 2 parameters.

Parameter1:	The 8-byte filed that contains the DDname to be checked
Parameter2:	An optional 44-byte field that receives the dataset name to which the DDname has been allocated.

If parameter2 is not supplied, the dataset name will not be returned.

If the specified DDname has been allocated, the program will set the return code to 0. If not, return code 8 will be set.

COBOL Example 1:

```
CALL 'IKANTIOT' USING DD-NAME.
```

with:

```
01  DD-NAME PIC X(8) VALUE 'PDSLIB'
```

will set the return code to 0 if DDname 'PDSLIB' has been allocated. Otherwise, it will set the return code to 8.

COBOL Example 2:

```
CALL 'IKANTIOT' USING DD-NAME DS-NAME.
```

with:

```
01  DD-NAME PIC X(8) VALUE 'PDSLIB'.
01  DS-NAME PIC X(44) VALUE SPACES.
```

If DDname 'PDSLIB' has been allocated, the return code will be set to 0 and it will write the dataset name that is allocated by DDname PDSLIB into the storage area DS-NAME. Otherwise, it will set the return code to 8.

4.6. IKANUPPR

The program *IKANUPPR* is a subroutine that will set an 80-byte storage area to upper case.

COBOL Example:

```
CALL 'IKANUPPR' USING MY-STORAGE.
```

with:

```
01  MY-STORAGE  PIC X(80) VALUE 'abcdefgh'.
```

Result after the Call:

Storage area MY-STORAGE will contain 'ABCDEFGH'.

4.7. IKANPRGM

Th program *IKANPRGM* will write the parmcard that is supplied in the JCL to DDname PARMOUT. DDname PARMOUT must have been allocated with a logical record length of 80 bytes.

Example:

```
//ADCDMST,A JOB (5145,00000,2233,T),'IKAN',
//          MSGLEVEL=(1,1),MSGCLASS=X,
//          CLASS=A,REGION=0M,
//          NOTIFY=&SYSUID
//*
//         SET STEPLIB1=IKANALM.IKAN.COMMON.LOADLIB
//IKANPRMG EXEC PGM=IKANPRMG,
//          PARM='EXAMPLE DATA'
//STEPLIB   DD DISP=SHR,DSN=&STEPLIB1
//PARMOUT   DD SYSOUT=*,LRECL=80,BLKSIZE=8000,RECFM=FB
```

In the example above, the program will write the string 'EXAMPLE DATA' to DDname PARMOUT.

If you want to use the program IKANPRGM as a subroutine, the caller needs to indicate the length of the data field (see the COBOL example below).

```
01 PARM-DATA.
03  PARM-LENGTH    PIC 9(04) COMP.
03  PARM-DATA      PIC X(01) OCCURS 0 TO 121 TIMES DEPENDING ON PARM-LENGTH
```

CHAPTER 5
REXX Programs

This section describes the functionality of the Common REXX Programs.

5.1. IKANCONC

The REXX program *IKANCONC* concatenates one or more datasets into one output dataset. It is possible to concatenate a file twice (or more).

DDname READINDD accepts control statements that specify the DDnames to read from. The sequence of the control statements also determines the sequence of the DDnames to be read.

The DDname to which the output is written is determined by the JCL PARM= option 'OUTDD=<name of DDname to receive the output>'

The parameters for this REXX program should be supplied through the PARM= statement in the JCL.

Parameters supported:

TRACE=YES	is an optional parameter that will activate the REXX trace facility.
TRACE=NO	is an optional parameter that will deactivate the REXX trace facility
OUTDD=DDname	is a mandatory parameter that specifies the output DDname to which the output is written.
REMOVEASA	is an optional parameter that will remove the first byte from the input before it is written to the output

DDname READINDD accepts (one or more) control statements which need to be specified as:

INDD=input-DDname

Example:

```
//*
//COMBOALL EXEC PGM=IKJEFT01,
//         PARM='%IKANCONC OUTDD=FTPALL',
//         COND=(4,LT)
//SYSEXEC  DD DISP=SHR,DSN=IKANALM.IKAN.COMMON.REXX
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DUMMY
//READINDD DD *
INDD=FTPIDMS
INDD=FTPPDS
//SYSTSIN  DD DUMMY
```

```
//FTPIDMS  DD DISP=(OLD,PASS),DSN=&&FTPALLI
//FTPPDS   DD DISP=(OLD,PASS),DSN=&&FTPALLP
//FTPALL   DD DISP=(OLD,PASS),DSN=&&FTPALL
```

In the example above, the DDname READINDD specifies that DDname FTPIDMS will be read first and written to output DDname FTPALL. Secondly, DDname FTPPDS will be read and written (added) to output DDname FTPALL.

5.2. IKANCTL

The REXX program *IKANCTL* is a program that calls other programs or REXX programs. If so, it will allocate resources which are needed to communicate with an IDMS Central Version.

```
//*
//IKANIDD EXEC PGM=IKJEFT01,
// PARM='%IKANCTL PGM=IKANIDD CV=20 DICT=APPLDICT',
//         COND=(4,LT)
//*
//SYSEXEC  DD DISP=SHR,DSN=IKANALM.IKAN.COMMON.REXX
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DUMMY
//SYSOUT   DD SYSOUT=*
//SYSIDMS  DD DISP=(OLD,PASS),DSN=&&SYSIDMS
```

In the example above, the program allocates the necessary SYSCTL file to communicate with IDMS and itwrites information to DDname SYSIDMS to set the Database environment where the data resides. After that, it will call the program IKANIDD.

Parameters supported:

CV=cvnumber	is a mandatory parameter identifying the IDMS Central Version to which the SYSCTL file will be allocated.
PGM=program	is a parameter that specifies the program name to call. This parameter is mutually exclusive with the REXX= parameter.
REXX=rexx-program	specifies the REXX program to call. This parameter is mutually exclusive with the PGM= parameter.
PARM=parameter	is an optional parameter that specifies the parameters that need to be passed to the program (PGM= or REXX=). The parameter to be passed must be enclosed by open and close brackets.
MODE=mode	is an optional parameter that specifies to run against IDMS in Central or Local mode. Default is CENTRAL.
DICTNAME=dictname	is a mandatory parameter that specifies the Dictionary to be set. If no dictionary is required, the value NODICT must be specified. The DICTNAME=dictname statement is written to DDname SYSIDMS unless the value equals NODICT.
DICT=dictname	(see DICTNAME above)
DBNAME=dbname	is an optional parameter that will force the program to write a 'DBNAME=dbname' to DDname SYSIDMS.

RC=returncode	is a parameter that specifies the maximum acceptable return code from the program that is called (PGM= or REXX=). If the return code is greater that the value specified in RC=value then the REXX program will set the return code to 100. If the return code is less or equal the parameter value, the REXX program will set the return code to 4. If the return code is zero, the return code will also be set to zero. This parameter is mutually exclusive with MAXRC=
MAXRC=returncode	(see RC above)
TRACE=YES/NO	YES will activate the REXX trace facility. NO will not activate the REXX trace facility

5.3. IKANFLAT

The REXX program *IKANFLAT* is a program that reads PDS members and creates a sequential file in which every member is 'prefixed' by an IEUPDTE command. The sequential file can be processed by the IBM utility IEUPDTE to populate a PDS.

```
//*
//*-------------------------------------------------------
//*--- FLATTEN PDS, INCLUDING './ ADD NAME=MBR' STATEMENT
//*-------------------------------------------------------
//FLATREX  EXEC PGM=IKJEFT01,
// PARM='%IKANFLAT TRACE=YES PDS=IKANALM.IKAN.COMMON.REXX',
//       COND=(4,LT)
//SYSEXEC  DD DISP=SHR,DSN=IKANALM.IKAN.COMMON.REXX
//SYSTSIN  DD DUMMY
//SYSTSPRT DD SYSOUT=*
//IEBUPDTE DD DISP=SHR,DSN=SEQOUT
//MBRLIST  DD DISP=(OLD,PASS),DSN=&&MEMBERS
```

In the example above, the REXX program will read the member names from DDname MBRLIST. Every member will be read from the PDS IKANALM.IKAN.COMMON.REXX which is passed as a parameter. All members are written to DDname IEBUPDTE. Before a member is written to DDname IEBUPDTE, the program writes an IEUPDTE statement .i.e. './ ADD NAME=membername'.

Input:

DDname MBRLIST needs the member name in position 1-8.

Parameters:

TRACE=YES/NO. This option is optional. Option YES will activate the REXX trace facility. Option NO will deactivate the trace facility.

PDS=pdsname is a mandatory option and indicates the name of the PDS from which members should be read.

5.4. IKANGENR

The REXX program *IKANGENR* will copy a file from one DDname to another. By default, the input is read from DDname SYSUT1 and the output is written to DDname SYSUT2. The input and output DDnames can be chosen by the user. The option REMOVEASA will remove position 1 from the input file before the file is written to the output DDname.

Parameters supported:

TRACE=YES	is an optional parameter that will activate the REXX trace facility.
TRACE=NO	is an optional parameter that will deactivate the REXX trace facility
OUTDD=DDname	is an optional parameter that specifies the output DDname to which the output is written.
INDD=DDname	is an optional parameter that specifies the input DDname from which the input is read.
REMOVEASA	is an optional parameter that will remove the first byte from the input before it is written to the output

```
//*-------------------------------------------
//PRINT1   EXEC PGM=IKJEFT01,
//         PARM='%IKANGENR OUTDD=IKANOUT ',
//         COND=(4,LT)
//SYSTSPRT DD  SYSOUT=*
//SYSEXEC  DD  DISP=SHR,DSN=IKANALM.IKAN.COMMON.REXX
//SYSTSIN  DD  DUMMY
//IKANOUT  DD  SYSOUT=*
//SYSUT1   DD  *
  DATA TO BE COPIED TO DDNAME IKANOUT
```

In the example above, the input from (default) DDname SYSUT1 is copied as is to output DDname IKANOUT (as specified in the PARM= option).

5.5. IKANRC

The REXX program *IKANRC* reads 1 line from DDname RC and determines whether to set a high return code indicating an error, so that additional JCL steps can react on that high return code.

This is applicable when running a batch job under 'ISPF' (ISPSTART). In those circumstances return codes are not passed back to z/OS. In other words: this JCL step will always return a step condition code of zero (which indicates that everything is fine). Such a batch job REXX execution might write info to the RC DDname to indicate that an error has been encountered. The REXX program IKANRC is then able to set a real return code indicating the error.

Parameters:

STEP=stepname is a mandatory parameter and should indicate the step name in which the error has been encountered.

Input:

DDname RC The first word in this DDname should be 'RC=returncode'. If the 'returncode' value does not equal the value zero, the REXX program will set the return code to 100.

```
//*-------------------------------------------
//SETRC   EXEC PGM=IKJEFT01,
//         PARM='%IKANRC STEP=ISPF '
//SYSTSPRT DD  SYSOUT=*
//SYSEXEC  DD  DISP=SHR,DSN=IKANALM.IKAN.COMMON.REXX
//SYSTSIN   DD  DUMMY
//RC              DD  *
RC=8
```

In the example above, the RC=8 return code indicates that step ISPF has encountered an error. The REXX program will generate the following message:

'Return code 8 encountered in step ISPF'

and it will set the return code to value 100.

```
//*-------------------------------------------
//SETRC   EXEC PGM=IKJEFT01,
//         PARM='%IKANRC STEP=ISPF '
//SYSTSPRT DD  SYSOUT=*
//SYSEXEC  DD  DISP=SHR,DSN=IKANALM.IKAN.COMMON.REXX
//SYSTSIN   DD  DUMMY
//RC              DD  *
RC=0
```

In the example above, the RC=0 return code indicates that the ISPF stepdid not encounter an error. The REXX program will not generate any message and it will set the return code to value 0 (zero).

5.6. IKANUPDT

The REXX program *IKANUPDT* is an alternative for the IBM utility IEUPDTE. This IBM utility is limited in its usage, i.e., it is only capable to process PDS libraries with a logical record length of 80 bytes. This REXX program can handle any record length. The REXX program must be run under Ispstart.

Parameters supported:

TRACE=YES	is an optional parameter that will activate the REXX trace facility.
TRACE=NO	is an optional parameter that will deactivate the REXX trace facility
DDNAME=DDname	is a mandatory parameter that specifies the DDname of the PDS to which the members should be written.

DDnames:

INPUT identifies the input sequential file that contains the member sources that must be prefixed by an IEUPDTE control statement (./ ADD NAME=membername)

RC is the output file to which the return code is written (IKANRC (page 15)).

```
//*  ADD  MEMBERS  INTO  PDS  USING  OWN  IEBUPDTE  (LRECL  NE  80)                //
*************************************************
//ADDADS   EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
ISPSTART CMD(%IKANUPDT DDNAME=DD01)
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSEXEC  DD DISP=SHR,DSN=IKANALM.IKAN.COMMON.REXX
//ISPPLIB  DD DISP=SHR,DSN=ISP.SISPPENU
//ISPMLIB  DD DISP=SHR,DSN=ISP.SISPMENU
//ISPSLIB  DD DISP=SHR,DSN=ISP.SISPSENU
//ISPTLIB  DD DISP=SHR,DSN=ISP.SISPTENU
//ISPPROF  DD DISP=(OLD,PASS),DSN=&&ISPPROF
//RC       DD DISP=(OLD,PASS),DSN=&&RC            CONTAINS REAL RC
```

```
//DD01     DD DISP=(OLD,PASS),DSN=&&USERPDS
//INPUT    DD *
./ ADD NAME=MEMBER1
DATA for member MEMBER1
./ ADD NAME=MEMBER2
DATA for member MEMBER2
```

In the example above, the members Member1 and Member2 are written to the PDS that is allocated with DDname DD01.

The result of this operation is written to DDname RC.

CHAPTER 6
Windows Programs

The *IKAN ALM Common Utilities* come with 2 Windows Programs (executables) which enable JCL jobs to run on z/OS and to decide whether the JCL has run successfully. Both programs should run in a Windows command file.

- The program *JCHCK000* reads a JCL model that contains information about each JCL step and the maximum return codes that are acceptable for each step. The maximum return code information is indicated in the JCL model but is no JCL. This step information is extracted and removed from the model to create runnable JCL which can be sent to z/OS to run on z/OS as a batch job.
- The program *JCHCK010* reads the output from the JCL job and determines whether JCL steps have failed based on the extracted information by JCHCK000. If there are no errors in the JCL job, the command file error-level will be set to zero, else the error-level will be set to value 100.

Combined solution:

The programs *JCHCK000* and *JCHCK010* together allow you to fine-tune your step control when submitting jobs to the z/OS mainframe environment. By adding the statement `MAXRC=max-return-code` to your JCL model, you are able to determine acceptable step execution.

For example: you have a two-step job with a compile step and a copy member step. The compile step normally will be successful when the return code does not exceed the value 4. However, the copy step (PGM=IEBCOPY) will not be successful when returning a return code of 4. In this case, the compile step requires MAXRC=4 and the copy step requires a MAXRC=0.

The `MAXRC=n` statement is not a valid JCL statement and therefore will generate a JCL error. This problem is solved by the program *JCHCK000*. This program will read the JCL model. If it encounters a `MAXRC= statement` it will turn the statement into a JCL comment. The result of this operation is correct JCL and will be written to the specified file. It will also write the MAXRC information to a file that can be used by the second program (*JCHCK010*).

It does not matter where you place your `MAXRC= statement`, as long it is 'JCL syntactically' correct. The program *JCHCK000* will find it and will create correct JCL.

6.1. JCHCK000

Requirements

- The program must run on a Windows platform in a command file
- It requires two parameters to be passed to the program

Parameters

-jcl:jclfiletobesubmitted	This file contains the JCL model to be processed. The MAXRC= statements are optional. This file will be rewritten by the program with JCL statements without MAXRC= statements, i.e. MAXRC= statements will appear as comment. This parameter is mandatory.
-step:filetoholdmaxrcinformation	This file will be populated by the program with the maximum allowed return code per step name.This parameter is mandatory.

Note: If the filename specified in either the –JCL parameter or the –step parameter contains spaces, the filename should be enclosed by quotes (single or double)

Functionality

- The program will remove `MAXRC=value` statements from the JCL file and will change them into comments.
- `MAXRC=value` information is written with the step name to the file mentioned in the '-step:' parameter.
- The JCL without the `MAXRC=value` statements are written back to the file mentioned in the '-jcl:' parameter.
- If some error is encountered, like unable to read files or incorrect parameters, the batch command error level is set to 100.

Examples:

Original JCL:

```
//jobname JOB etcetera
//STEP1 EXEC PGM=IEFBR14,REGION=0M,
//             MAXRC=0
//DD01 DD DISP=(OLD,PASS),DSN=&&TEMP
//STEP2 EXEC PGM=IEFBR14,MAXRC=0,COND=(4,LT),
//             REGION=2040K
```

New JCL:

```
//jobname JOB etcetera
//STEP1 EXEC PGM=IEFBR14,REGION=0M
//*            MAXRC=0
//DD01 DD DISP=(OLD,PASS),DSN=&&TEMP
//STEP2 EXEC PGM=IEFBR14,COND=(4,LT),
//*       MAXRC=0
//             REGION=2040K
```

In the example above, you see that the MAXRC statements have been removed from the JCL and have been set to comment.

6.2. JCHCK010

Requirements

- The program is running on a Windows platform in a command file
- It requires two parameters

Parameters

-log:mvsjoblogfile	This file contains the job log that is received from the z/OS platform after running. This parameter is mandatory.
-out:reportfile	This file contains the messages that are generated by the program. This parameter is mandatory.
-step:filethatholdsmaxrcinformation	This file contains, per step, the return code that is acceptable for this JCL run. This parameter is optional. The file specified should match the file that was passed to the program JCHCK000 with the parameter '-step:filetoholdmaxrcinformation'

Functionality

The program will read the job log and for every IEF142I message, the return code is reported and compared with the maximum allowed value of the step involved.

If you specify a file that holds step info ('-step: filetoholdmaxrcinformation' parameter), the step return code will be compared with the MAXRC value for that step in the step info file.

If you do not specify the '-step: filetoholdmaxrcinformation' parameter, a maximum value of 4 will be assumed.

Check on error messages IEF212I, IEF213I, IEF450I and IEF453I. If such a message is encountered, the error level is set to 100.

If some step return code exceeds the maximum allowed, the error level is set to 100.

Messages are written to the file mentioned in the '-out:' parameter.