innovative. individual. independent.

improv IT
Software Innovations GmbH

eXtended Compiler for REXX (XCR)

Installation and User's Guide

Version 3.4.0

June 2016

improvIT Software Innovations GmbH
Große Elbstraße 141 a
22767 Hamburg
Telephone: +49 (0)40 540 90 29 - 7
Fax: +49 (0)40 540 90 29 - 9
Email: Contact@improvIT-Software-Innovations.de
Web: www.improvIT-Software-Innovations.de

This page intentionally left blank

I. Content

II. Listings

III.Tables

1. Introduction

REXX has evolved into the preferred programming language on IBM z/OS systems. It is the standard programming language for system programmers alongside assembler. It is also ideal for application programmers who require a powerful tool to reduce the time required for carrying out repetitive tasks in a TSO/ISPF development environment.

REXX does, however, have some disadvantages. This, for example, includes being a purely interpreted language. REXX can easily be copied and changed resulting in a lack of investment protection and security.

1.1. REXX Compilers

There are several products on the market that allow REXX procedures to be compiled under IBM z/OS systems. In most cases a real compile (i.e. a full conversion to machine code) is not performed. Instead the REXX procedure is “pre-processed” and encapsulated into a load module.

The compiled REXX procedures can then be directly called from all environments (e.g. TSO, JCL, other programs).

The eXtended Compiler for REXX (XCR) offers an effective and easy way to convert your REXX procedures into load modules.

1.2. eXtended Compiler for REXX

XCR was developed to generate load modules from REXX procedures. The compiler offers various technical features which are not available in the interpreter and in other similar products:

- The source code is encrypted (investment protection)
- Code can be copied from other libraries using the INCLUDE statement (simplified development)
- A Copyright message can be added to the load modules
- The load modules do not require a runtime library
- Unnecessary blanks and comments are removed from the source module during the compile (improved runtime performance)
- Does not require any runtime related information during the compile (no stubs)
- The modules can be called directly without using batch TSO
- The API may be used to call compiled REXX modules from other programming languages (e.g. Assembler, Cobol, C). All results are returned to the caller.
- System-Variables containing compile date/time information can be used.

These features can reduce maintenance costs and simplify development. System availability can also benefit.

1.2.1. Full ISPF Support

XCR also offers a unique feature which is not available using other compilers. IBM Interactive System Productivity Facility (ISPF) edit macros may be also compiled and executed.

Internally ISPF can only call edit macros written in REXX or as CLISTS and program macros written in a high level language (e.g. Assembler, Cobol, PL/1, C).

XCR allows compiled REXX edit macros to be directly called when using the ISPF editor. All advantages of the XCR compiler can therefore be applied to ISPF edit macros.

2. Compiler Installation

2.1. Software Prerequisites

The following IBM software is required:

– z/OS

All releases are supported.

The **improvIT Software Innovations** software “eXtended Common Services (XCS)” is only needed if you want to use the "TSO Dynamic STEPLIB" feature (see chapter 7 for details).

2.2. Runtime Datasets

It is recommended, that the XCR load libray is defined in the Linklist concatenation. Alternatively the load libraries may be added to a Steplib concatenation.

No system modifications are required.

2.3. Datasets

XCR consists of one loadlib (SXCRLOAD) and one samplib (SXCRSAMP). The files are delivered in TSO transmit format.

To install XCR, just transfer the xmit files in binary mode to your host system (e.g. using FTP). The files must be extracted by using the TSO receive command. The target dataset names can be specified at this point.

The following JCL shows a sample extract Job:

```
//JOBCARD  JOB (ACCT),'PROG',
//             CLASS=?,MSGCLASS=?,
//             NOTIFY=&SYSUID
//*********************************************/
//* EXTRACT XCR FROM XMIT FILES                */
//*********************************************/
//S1       EXEC PGM=IKJEFT01,DYNAMNBR=30
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
      PROFILE NOPREFIX
      RECEIVE INDSN('???.XCR340.XMIT(SXCRLOAD)')
      DSNAME('???.XCR340.SXCRLOAD')

      RECEIVE INDSN('???.XCR340.XMIT(SXCRSAMP)')
      DSNAME('???.XCR340.SXCRSAMP')
/*
```

Listing 1: JCL sample extract job

2.4. Licence Key

To use the XCR compiler you will need a licence key. This key is supplied by **improvIT Software Innovations**.

The key needs to be installed using the samplib member "ACTLIKEY". The sample AMASPZAP job updates the licence information in the XCR loadlib.

There are 3 different licence types:

1. Base: A maximum of 100 executable REXX statements are allowed
2. Developer: A maximum of 250 executable REXX statements are allowed
3. Full licence: No limitations

Please contact improvIT Software Innovations for further details.

3. Getting started

3.1. Including Code

The INCLUDE statement is a feature that enables you to copy REXX source statements into your procedure from other libraries. Define the libraries to be searched by adding one or more datasets to the SYSLIB DDName in the compile step. The SYSLIB DDName is optional or may be allocated as Dummy. If used the libraries must be partitioned with a logical record length of 80.

In the REXX procedure the INCLUDE statement must be coded exactly as follows: “/*%INCLUDE member*/”. Only the member name may be changed. No spaces may be added.

Note: Nested %INCLUDE statements are not resolved.

3.2. Copyright

The COPYRIGHT statement enables you to place a legible text notice into your load modules.

In the REXX procedure the COPYRIGHT statement must be coded exactly as follows: “/*%COPYRIGHT user text*/”.

The end of the coded statement should not exceed byte 70 of the record. Otherwise XCR will treat the statement as a normal comment and it will not be legible in the load module.

Note: Only the first three %COPYRIGHT statement are processed. All others are treated as normal Rexx comments.

3.3. XCR System Variables

Three XCR system variables may be used anywhere within the Rexx code (also in %Copyright statements and %Include code). These are resolved by XCR during the compile. The variables with their returned values are as follows:

%XCRDATE - dd.mm.yy

%XCRTIME - hh:mm:ss

%XCRYEAR - yyyy

Values are always left justified and padded with spaces to a length of 8 bytes.

Example: "Say Compile Date is %XCRDATE".

3.4. Compile Job

REXX procedures are compiled through a simple batch job. A sample job “REXXCOMP” can be found in the samplib dataset. It is recommended, that the required JCL is made available to all users in form of a JCL procedure.

The compile job consists of 2 steps:

- S1COMP - Converts the REXX procedure into a source deck
- S2LINK - Links the source deck and generates the load module

3.4.1. Step 1 - Compile

The following JCL shows the compile step:

```
//*******************************************************************
//*  COMPILE THE REXX                                               *
//*******************************************************************
//S1COMP  EXEC PGM=XCRLRLMG [,PARM='<REXXNAME>']
//STEPLIB  DD DISP=SHR,DSN= <SXCRLOAD>
//* REXX SRC:
//SYSIN    DD DISP=SHR,DSN= <YOUR.REXX.LIB>(<REXXNAME>)
//* OPTIONAL FOR INCLUDE STATEMENTS;
//SYSLIB   DD DISP=SHR,DSN= <YOUR.INCLUDE.LIB>
//*
//XCRDSRC  DD SYSOUT=*
//XCRDRUN  DD SYSOUT=*
//XCRDOBJ  DD DSN=&&REXXOBJ,
//            DISP=(,PASS),UNIT=VIO,SPACE=(CYL,(1,1)),
//            BLKSIZE=3200
//XCRDLNK  DD DSN=&&REXXLNK,
//         DISP=(,PASS),UNIT=VIO,SPACE=(CYL,(1,1)),
//         BLKSIZE=3200
//XCRDWRK  DD UNIT=SYSDA,SPACE=(CYL,(5,5)),BLKSIZE=3200
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
```

Listing 2: JCL Compile

The optional PARM value *<rexxname>* determines the name of the REXX CSECT in the load module. Generally it should not be specified. But it is required if SYSIN is not a member of a PO dataset. The STEPLIB is only required if the runtime libraries are not in the Linklist.

The following dataset changes need to be made in Step 1:

DDName	Format	Description
SYSIN	LRECL=80	Library containing input REXX source
SYSLIB	LRECL=80	Library containing code as required by INCLUDE statements
XCRDSRC	LRECL=90	Listing of the input REXX after resolving any INCLUDE statements
XCRDRUN	LRECL=90	Listing of the compressed input REXX
SYSPRINT	Sysout	Contains compile statistics

Table 1: Datasets for compile

The following values are returned by the compiler:

Return Code	Description
0	Compile completed successfully
4	Licence expires in less than 30 days or Code was modified by Compiler
8	Licence expired less than 14 days ago
12	Invalid parameter, processing error or CSECT name could not be determined (source was not in PO dataset or optional call PARM not used) Maximum lines of code for licence type (BASE = 100 or DEVELOPER = 250) exceeded.
16	No real statements in the source REXX (only comments)
36	Licence missing or invalid

Table 2: Return codes from compile

3.4.2. Step 2 - Link

The following JCL shows the link step:

```
//*********************************************************************
//*  LINK THE OBJ FILE                                                *
//*********************************************************************
//S2LINK  EXEC PGM=IEWL,
//             PARM='LIST,XREF,MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DSN=&SYSUT1,SPACE=(1024,(120,120),,,ROUND),UNIT=SYSDA,
//             DCB=BUFNO=1
//SYSLIB   DD  DISP=SHR, DSN= <SXCRLOAD>
//SYSLIN   DD  DSN=&&REXXOBJ,DISP=(OLD,DELETE)
//         DD  DSN=&&REXXLNK,DISP=(OLD,DELETE)
//SYSLMOD  DD  DISP=SHR,DSN= <YOUR.REXX.LOADLIB>
```

Listing 3: JCL Link

The following dataset changes need to be made in Step 2:

DDName	Description
SYSLIB	Must contain the name of the XCR load library
SYSLMOD	The name of the target load library

Table 3: Dataset changes for link step

3.5. Considerations

The following points must be observed when using the XCR compiler:

- If an EXIT or RETURN statement is coded without a return value, then the compiler will automatically append a zero to the corresponding statement. The compile will terminate with a return code of 4 if changes were made to EXIT or RETURN statements
- REXX functions may not return data greater than 32K in length (using the Return or Exit statements). otherwise a REXX error 40 will occur.

4. Compiled ISPF Edit macros

4.1. Compiling ISPF Edit Macros

All IBM ISPF Edit Macros can be compiled into load modules using the eXtended Compiler for REXX. XCR automatically recognizes when a macro is being compiled and performs the required changes.

Due to ISPF system restrictions, XCR internally removes the initial "ISREDIT MACRO" instruction in the edit macro and replaces it with a corresponding "Parse ARG" statement in the load module.

XCR requires that at least the following code is located somewhere in the edit macro in order for recognition to take place:

"ISREDIT MACRO" or 'ISREDIT MACRO' or ISREDIT MACRO

Optional parameters as well as the PROCESS and NOPROCESS options are handled automatically.

Note: If a compiled edit macro is called directly from environment other than the IBM ISPF Editor (e.g. TSO), then the macro will terminate prior to execution!

4.2. Using ISPF Edit Macros

It is not possible to directly call compiled ISPF edit macros from the editor. This is due to ISPF restrictions. It is necessary to first define all compiled edit macros within the ISPF edit session. This needs to be performed every time the ISPF editor is used. The easiest method to execute the definitions every time the editor is started is to use the ISPF system wide editor initialisation macro (as defined in the ISPF system options). The options can be customised using the IBM ISPF configuration tool ISPCCONF.

The sample "ISPFIMAC" is located in the XRS samplib. Copy the necessary code to your default ISPF system wide editor initialisation macro.

```
/* REXX ***************************************************************/
/*                                                                    */
/* This sample edit macro demonstrates how compiled edit macros may   */
/* called from the ISPF editor. This code should generally be added   */
/* to the system wide ISPF initial edit macro.                        */
/*                                                                    */
/**********************************************************************/

/* Setup environment */
Address ISREDIT
"MACRO NOPROCESS"

/* Define the XCR edit macro to ISPF */
"DEFINE <MACRO-NAME> MACRO PROGRAM"

Return 0
```

Listing 4: Sample "ISPFIMAC"

These ISPF Define statements are required in order that the IBM ISPF editor can process compiled edit macros (also known as ‘program macros’). One DEFINE statement must be added per program macro.

5. Running Compiled REXX modules

Generally no modifications are required when running compiled REXX modules.

However, different values might be returned, if you are using the "parse source" command to establish the current runtime environment.

The DDName "SYSTSPRT" must always be allocated if you are running compiled REXX modules. If functions such as interactive tracing are being used, then the DDName "SYSTSIN" must also be allocated. These DDNames are always available when running under online TSO.

If you are using ISPF dialog functions or accessing ISPF variables, then the initial REXX module must be started using the "LANG(CREX)" subparameter of the "SELECT" command.

5.1. Batch Execution

Compiled REXX modules can be run in batch using:

TSO IKJEFT1A/01 or

XCR service routine XCRLSJCL

If you wish to execute compiled REXX modules directly (i.e. with "EXEC PGM=") and not via TSO, then use the XCR service routine XCRLSJCL. This routine also allows REXX modules to return non-numeric data without resulting in "uncontrollable" return codes.

The name of the compiled REXX to be executed and any parameters are passed to XCRLSJCL. After processing, the first 120 bytes of data returned by the compiled REXX are written to the job log.

The DDName "SYSTSPRT" must be allocated. All output is written to this DDName.

5.1.1. Return Codes

Return code	Description
0	Processing successful
16	An invalid REXX module name was specified
20	No Parameters were supplied
24	More than 120 bytes of data were returned by the REXX module
36	The requested REXX module could not be found
??	Runtime return codes from REXX processing

Table 4: Return codes

6. Callable API

The XCR API offers a unique feature. By using the API, it is possible to call compiled REXX modules from any programming language adhering to the standard IBM Linkage Conventions. The API allows values to be passed to the REXX and returns the results after processing. Results can be any value and not just a numeric return code.

6.1. Calling Conventions

The structure of the API parameters is as follows:

Parameter	Format	Description
REXX Module	Character 8 bytes	Must contain the name of the compiled REXX which needs to be executed
Input Length	Fullword Binary (4 bytes)	The length of the parameter area which needs to be passed to the compiled REXX
Input Area	Character with a length matching the value of "Input Length"	This field contains the string to be passed to the compiled REXX
Output Length	Fullword Binary (4 bytes)	Input: The length of the parameter area in which the results of the compiled REXX are to be returned. Output: The length of the data returned by the compiled REXX module.
Output Area	Character with a length matching the value of "Output Length"	This field contains the result string returned by the compiled REXX

Table 5: XCR API Calling Conventions

The Module "XCRLSPGM" must be called using the above parameters. Register 1 must point to a list of addresses which in turn point to the data areas (IBM Linkage Conventions).

Three supplied samples demonstrate the use of the API:

Sample	Description
ASM2XCR	Using Assembler
COB2XCR	Using Cobol
C2XCR	Using C

Table 6: API Samples

The DDName "SYSTSPRT" must be allocated. All output is written to this DDName.

6.2. API Return Codes

The following return code values are possible:

Return code	Description
0	Processing successful
24	The supplied "output area" was not large enough to store all of the returned data. The required length can be found in field "output length" after processing
36	The requested REXX module could not be found
??	Runtime return codes from REXX processing

Table 7: API Return Codes

7. TSO Dynamic STEPLIB feature

To support testing of the compiled REXX programs the "TSO Dynamic STEPLIB" feature is included in this release.

It allows you to test a new version of a load module in the TSO STEPLIB without a TSO LOGOFF/LOGON.

You must install the **improvIT Software Innovations** base product XCS (eXtended Common Services version 2.8.0 or higher) if you want to use this feature.

7.1. Installing eXtended Common Services (XCS)

XCS consists of two loadlib (SXCSLOAD and SXCSAPF). The files are delivered in TSO transmit format.

To install XCS, just transfer the xmit files in binary mode to your host system (e.g. using FTP). The files must be extracted by using the TSO receive command. The target dataset names can be specified at this point.

It is recommended to add both load libraries to the LNKLST.

The SXCSAPF dataset must be APF authorized. Furthermore you must add the entry point name XCSADSTP to the AUTHTSF section in your IKJTSOxx PARMLIB member.

7.2. Using the TSO Dynamic STEPLIB feature

This feature is supplied as a REXX function. It may be called from any REXX exec.

The syntax is as follows:

XRC = XCSXDSTP(function, stemname, datasetname)

The parameters are:

1. Function: One of the following functions must be specified

ADDF	Adds the dataset as first dataset in the current STEPLIB concatenation
ADDL	Adds the dataset as last dataset in the current STEPLIB concatenation
ALLOC	Allocates a new STEPLIB
FREE	Frees the current STEPLIB
LIST	Lists the current STEPLIB datasets
REMOVE	Removes the dataset from the current STEPLIB

2. Stemname: The name of a stem in which the STEPLIB dataset names are returned (must end with a dot and must not exceed 8 characters)
3. Datasetname: The name of a dataset to be added to / removed from the current STEPLIB (mandatory for functions ADDx, ALLOC, REMOVE)

The results are returned in stem variables:

<STEM>.0	Number of dataset entries (new STEPLIB)
<STEM>.1	New STEPLIB DDNAME (#TEP0001, #TEP0002,…)
<STEM>.2	First dataset of new STEPLIB
<STEM>.n	Last dataset of new STEPLIB

A new STEPLIB is always allocated with a new DDNAME: the next available name #TEPnnnn will be used.

Examples:

```
XRC = XCSXDSTP('LIST', 'DSTP.')        lists the current STEPLIB

XRC = XCSXDSTP('FREE', 'DSTP.')      frees the current STEPLIB

XRC = XCSXDSTP('ALLOC, 'DSTP.', 'USER.LOADLIB')  allocates a new STEPLIB
```

The return value XRC of this function is a one byte value containing '1' (function successful) or '0' (function unsuccessful).

Two further variables are also created:

XCSXDSTP.FC and XCSXDSTP.FRN contain the function return code and the function reason code.

The following table contains a list of possible situations and the corresponding results:

Error	XRC	.FC	.FRN
Processing successful	1	0000	0000
No arguments found	0	0012	0001
Zero length (argument 1)	0	0012	0002
Zero length (argument 2)	0	0012	0003
Zero length (argument 3)	0	0012	0004
Wrong length (argument 1)	0	0012	0006
Wrong length (argument 2)	0	0012	0007
Wrong length (argument 3)	0	0012	0008
Argument missing	0	0012	0010
Too much arguments (general)	0	0012	0011
Too much arguments (FREE/LIST)	0	0012	0012
Invalid argument 1	0	0012	0013
Invalid argument 2	0	0012	0014
The dataset does not exist	0	0012	0015
Licence module not found	0	0012	0020
Error in licence string	0	0012	0023
Licence has expired	0	0012	0025
Dynalloc (SVC 99) error	0	> 256	SVC 99 error code
Function not APF authorized	0	1044	0056

Table 8: Dynamic STEPLIB return and reason codes

Note: The XCR licence module XCR$$LIC must be accessible when calling the REXX function XCSXDSTP !

8. Release Information

8.1. Changes in 3.4.0

- "TSO Dynamic STEPLIB" support (see chapter 7 for details)
- The **improvIT Software Innovations** software “eXtended Common Services (XCS)” is only needed if you want to use the "TSO Dynamic STEPLIB" feature

9. Contact

For further information regarding the eXtended Compiler for Rexx (XCR) please contact:

improvIT Software Innovations GmbH
Große Elbstraße 141 a
D-22767 Hamburg
Germany

Telephone: +49 (0)40 540 90 29 - 7
Fax: +49 (0)40 540 90 29 - 9
Email: Contact@improvIT-Software-Innovations.de
Web: www.improvIT-Software-Innovations.de