IBM COMPILER AND LIBRARY FOR REXX/370

WALTER PACHL
IBM

IBM Compiler and Library for REXX/370

Walter Pachl

IBM Vienna Software Development Lab
c/o IBM Austria, Dept 00/705
Obere Donaustrasse 95
A-1020 Austria

May 1, 1992

3rd Annual REXX Symposium for Developers and Users

An Overview

- What's new?
- User Interfaces, CMS and TSO
- Performance Comparisons
- Building a Standalone Program
- Building REXX External Functions
- Packaging an Application Using REXX

In August 1991 two new IBM products were made generally available: the IBM Compiler and Library for REXX/370. The major news coming with these products will be covered in the first part of this presentation.

As the author has a close affiliation with user interfaces for invoking the compiler, a little description of these interfaces will be given — the nearly unchanged CMS invocation dialog and the new MVS foreground compilation panel.

Apart from other benefits, performance is a major aspect in compiling Rexx. The performance expectations and the results of a running a few benchmark programs will be shown.

Finally the new products offer new possibilities for packaging applications. The presentation will close by demonstrating how these possibilities can be used and what advantages can be expected.

Product Improvements

- Support MVS/ESA
- Smaller, Faster, Less Expensive

  Smaller compiler and compiled programs
  Faster compilation and program execution
  Lower price, in particular for smaller processors

- CONDENSE option to get significantly smaller compiled code
- DLINK option to allow for new packaging
- Support different parameter passing conventions on MVS
- Tolerate Interpret

One Product

- Cross Compiler
  Programs compiled on one system can be run on either system

- Gives enterprise the ability to purchase a single compiler

- Library for REXX/370 is system dependent

- Programs compiled with the predecessor product (CMS/Rexx Compiler) can be run without recompilation.

Compiler produces on either system

- A **compiled EXEC** that can be used instead of the source program. (Moving from one system to another or from one library to another on MVS may require conversion from one record format to a different one — a utility is provided for performing this task.)
- An **object module** that can be turned into an executable load module or that can be link-edited with other programs.

The compiler's system interfaces, the user interface, and the run-time support are, of course, system dependent. Ordered by product number with feature for CMS or MVS. Packaging in predecessor product was Compiler and Runtime Library or (on customer request) Runtime Library alone. The new products are Compiler alone and Runtime Library. User must order both products for compiling and executing programs.

Upward compatibility is maintained: Programs compiled with predecessor product can be run with the new Library. However, no downward compatibility — we move forward. Predecessor product is now withdrawn from marketing.

Customer Complaint:

"Compiled programs are (much) larger than source code"

A new compiler option is offered that allows to condense the object code.

- Compiled program uses less disk space
- Literal strings (and source lines) become illegible
- Unpacking the program for execution takes a little time.

Consider	should use CONDENSE		should not use CONDENSE
Machine bottleneck	I/O	CPU	Storage
Program location	Disk	Storage (single use)	Storage (shared use)
Program size	large	medium	small
Program execution	long-running		short-running
Program invocation	seldomly		frequently
Other	source/constant protection		DLINK required

Another Performance Boost

Significant (search) time is spent when external programs are invoked. CMS/Rexx Compiler allowed to create TEXT and MODULE for a Rexx program. A new compiler option, DLINK, generates weak external references for external functions and subroutines in compiled object modules.

These can be resolved by combining caller and callees, using the linkage editor.

Under MVS, modules must be pre-linked using an appropriate **stub** to accommodate the different parameter passing conventions.

Use of DLINK can make an application self-contained: No name clashes with user's environment.

- MVS has many different parameter passing conventions
- REXX programs understand arguments
- These arguments are passed in a table
- Compiler for REXX/370 supports four types of parameters

  MVS type, used in PARM= on JCL

  CALL type, used in the TSO/E CALL command

  CPPL type, used in TSO/E commands

  EFPL type, used in REXX external functions and function packages

- Source of "stubs" is provided as examples
- Can be modified for other parameter passing conventions

MVS Parameter Passing Conventions - NotesREXX

The MVS type and CALL type are very similar. The CALL type is limited to a single parameter, and it will have an address less than 16 Meg. The PARM= on JCL is a single parameter, but other programs that use this convention may have more than one parameter.

Register 1 points at a list of addresses, the last of which has the high order bit on. Addresses point at the individual argument strings each of which consists of a length field followed by the actual data.

The CPPL is a four word construct mapped by the macro IKJCPPL.
The DSECT for this macro is:

```
*********************************************************************
*    THE COMMAND PROCESSOR PARAMETER LIST (CPPL) IS A LIST OF
*    ADDRESSES PASSED FROM THE TMP TO THE CP VIA REGISTER 1
*********************************************************************
CPPL     DSECT
CPPLCBUF DS    A        PTR TO COMMAND BUFFER
CPPLUPT  DS    A        PTR TO UPT
CPPLPSCB DS    A        PTR TO PSCB
CPPLECT  DS    A        PTR TO ECT
```

The EFPL is a six word construct mapped by the macro IRXEFPL.
The DSECT for this macro is:

```
EFPL     DSECT
EFPLCOM  DS A             * RESERVED
EFPLBARG DS A             * RESERVED
EFPLEARG DS A             * RESERVED
EFPLFB   DS A             * RESERVED
EFPLARG  DS A             * POINTER TO ARGUMENTS TABLE
EFPLEVAL DS A             * POINTER TO ADDRESS OF EVALBLOCK
```

Stubs transform what comes in to what is expected. Under CMS, the compiled Rexx program is "self-adjusting."

Closer to Thee

- Interpret was flagged as SEVERE error by CMS/Rexx Compiler
  No compiled code was generated.

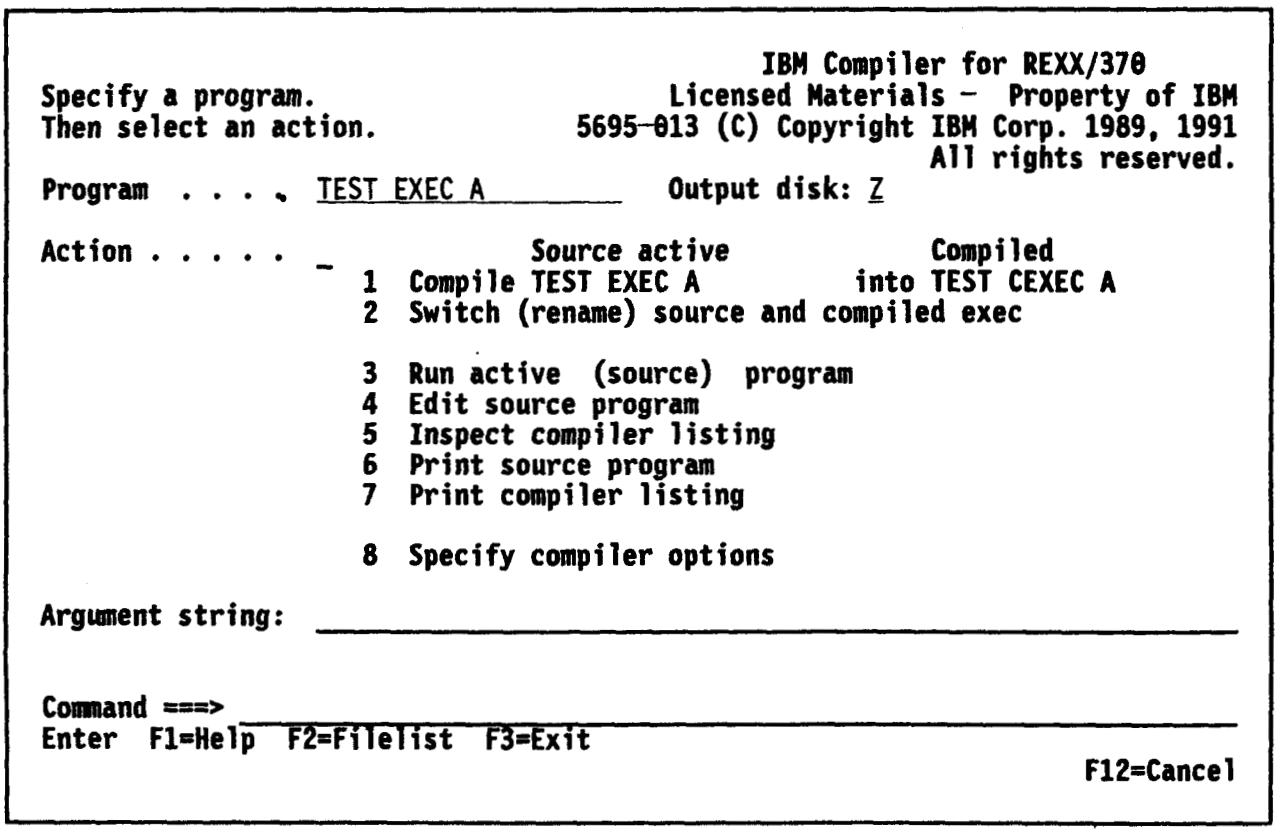
- Interpret is now flagged as ERROR
  Code is generated and causes a run-time error when the Interpret instruction is actually encountered.

  This can be avoided by:

```
Parse Version v
If left(v,5)<>'REXXC' Then          /* running compiled program     */
  Interpret instruction             /* we can interpret             */
```

- Support of Interpret is now an "Accepted Requirement"

CMS Compiler Invocation



REXXD - Compiler Invocation Dialog - Notes REXX

The compiler invocation dialog is intended to support all tasks involved in compiling for programmers as well as for casual users.

To use the compiler-invocation dialog under CMS enter the command:

rexxd test exec a

The panel appears as shown in the previous foil.

You may now select *Actions*:

1. Select Action 1 to compile the source program.
2. Select Action 2 to rename the source program and the compiled program.
3. Select Action 3 to run the currently active program.

If you need more information, refer to the online help by pressing the F1 key.

The name of the program to be compiled is carried over from the REXXD invocation or from the last invocation of this dialog. The name can, however, be changed on this panel. The panel is identical to that of the predecessor product, with one addition: the possibility to specify an output disk.

The panel indicates whether the source program or the compiled EXEC is currently active. The effect of switching between the two is reflected by appropriate highlighting.

Compiler options in effect can be displayed, changed, saved, and reset by selecting Action 8.

The Compiler Options Specification Panel REXX

```
                    REXX Compiler Options Specifications

Specify which output files you want, and their File-IDs

                                File identifiers
     Program name               ROULETTE  EXEC      G1
  Y  Compiler listing (Y/N/P)   =         LISTING   =
  Y  Compiled EXEC    (Y/N)     =         C*        =
  Y  TEXT file        (Y/N)     =         TEXT      =

Specify compiler messages to be issued
  I  FLAG   Minimum severity of messages to be shown (I/W/E/S/T/N)
  N  TERM   Display messages at the terminal (Y/N)
  N  SAA    SAA-compliance checking (Y/N)

Specify contents of compiler listing
  Y  SOURCE Include source listing (Y/N)
  Y  XREF   Include cross-reference listing (Y/S/N)
  N  LC     Number of lines per page (10-99 or, for no page headings, 0 or N)

Additional options
  N  SL     Support SOURCELINE built-in function (Y/N)
  Y  TH     Support HI immediate command (Y/N)
  S  NOC    Error level to suppress compilation (*/W/E/S)
  N  COND   Condense compiled program (Y/N)
  Y  DLINK  Include ESD and RLD in TEXT output (Y/N)
Special compiler diagnostics
  N    DUMP Produce diagnostic output (0-2047, Y, or N)

Command ===>
Enter  F1=Help  F2=Filelist  F3=Exit  F4=Save  F5=Refresh  F6=Reset
                                                          F12=Cancel
```

The options in effect are shown. Using entry fields and PF keys, the user can

- Change each compiler option individually (user input is checked and errors are diagnosed top down, field by field)
- Save the options in effect (in LASTING GLOBALV)
- Refresh the options (from LASTING GLOBALV)
- Reset the options to the installation defaults (taken from REXXC)

Help panels explain the available options and their meaning.

MVS Compiler Invocation

Under MVS, the usual methods of compiler invocation are supported:

- Foreground Compilation
- Background Compilation
- Cataloged Procedures

```
                        FOREGROUND REXX COMPILE
COMMAND ===>

ISPF LIBRARY:
   PROJECT ===> TEST
   GROUP   ===> LIB1      ===> LIB2      ===> LIB3      ===>
   TYPE    ===> REXX
   MEMBER  ===>                  (Blank or pattern for member selection list)

OTHER PARTITIONED OR SEQUENTIAL DATA SET:
   DATASET NAME  ===>

LIST ID ===>

COMPILER OPTIONS:             (extended REXXC options can be used)
        ===>
        ===>
```

Invoking the Compiler with ISPF Panels (MVS/ESA)

Under ISPF, you can invoke the Compiler from the Foreground REXX Compile panel and the Batch REXX Compile panel. The panels are similar to those for other high-level language compilers.

To use the Foreground REXX Compile panel:

1. Select FOREGROUND on the ISPF/PDF Primary Option Menu.
2. Select REXX Compiler.
3. Enter the appropriate data set names and (extended) compiler options. Extended compiler options allow to specify data set names where compiler output is to be stored.

From data entered on the panel, a command is built that allocates data sets as appropriate and that invokes the compiler with the appropriate compiler options. This command is, of course, implemented as a Rexx EXEC.

Rather unconventionally, background compilation **does not** employ file tailoring but uses also this REXXC command — albeit in batch.

Run-Time Performance Improvements

Programs with a lot of ...	TIMES faster than Interpreter	Performance Category
Arithmetic operations with default precision	6 - 10+	VERY HIGH
Arithmetic operations with other precision	4 - 25	
Assignments	6 - 10	
Changes to variables' values	4 - 6	HIGH
Constants and simple variables	4 - 6	
Reuse of compound variables	2 - 4	MEDIUM
Host commands	1 -	LOW

Performance - Notes

The performance improvements that you can expect when you run compiled REXX programs depend on the type of program. A program that performs large numbers of arithmetic operations of default precision shows the greatest improvement. A program that mainly issues commands to the host shows limited improvement because REXX cannot decrease the time taken by the host to process the commands.

Up to 30% CPU-load reduction have been reported on a heavily REXX-loaded machine.
" ... *better than last CPU upgrade.* On average 10-15% reduction are reported.

BENCHMAR EXEC

SPI-XA VM/XA System Product Interpreter Rel 5.6
REXX-370 IBM Compiler and Library for REXX/370 Rel 1.0

	TOTAL CPU TIME		RATIO
	SPI-XA	REXX-370	SPI/370
magnifier	0.76538	0.03749	20.42
forloop	7.97983	0.42967	18.57
whileloop	13.58915	0.57711	23.55
repeatloop	13.09567	0.54219	24.15
literalassign	10.34849	0.53132	19.48
memoryaccess	10.70410	0.56278	19.02
realarithmetic	3.29728	1.02355	3.22
realalgebra	2.69790	0.39420	6.84
vector	18.70649	1.89689	9.86
equalif	16.77867	0.91997	18.24
unequalif	16.74953	0.91375	18.33
noparameters	13.53182	1.29613	10.44
values	11.98723	2.49335	4.81
reference	19.90034	2.51308	7.92
wordscan400	21.16018	0.70951	29.82
command	0.68720	0.61554	1.12

An Example

- Compile the program using the OBJECT compiler option
- Turn it into a load module
  - Under MVS, by link-editing with the MVS-stub specified
  - Under CMS, by LOAD/GENMOD
- Place the load module
  - into an accessible library
  - onto an accessed minidisk
- Invoke it
  - from REXX (under MVS using Address LINKMVS)
  - from other languages, e.g., PL/I:

```
DCL  REXPGM ENTRY EXTERNAL OPTIONS(ASSEMBLER,INTER);
...
FETCH REXPGM;                          /* Bring it into storage   */
CALL REXPGM(VARSTRING);                /* Call the REXX program   */
RELEASE REXPGM;                        /* Release it from storage */
```

Performance opportunities

- External functions in load libraries generally found quicker (MVS)
- Function packages are first in the search order (Rexx search order)

Easy transition to load module

- Proceed as for standalone program
- but use EFPL stub instead of MVS

Building function packages

- Essentially a collection of external functions
- Each external Rexx function needs EFPL STUB (under MVS)
- When building package, naming convention consideration is important
- Description of packages in Rexx Reference manuals (system dependent).

Packaging Concept

- Can write an entire application in REXX
- External routines are directly LINKed
- Enabled through the use of DLINK compiler option

DLINK advantages

- Tremendous performance improvements from interpreted

  Mostly by eliminating search time

  Also due to inherent better performance in compiled REXX

- Functional isolation

  Each function can be in an external routine

  No name clashes with other system execs or commands

  No maintenance problems due to inadvertent modification of the exec

Packaging Considerations for MVS

- Naming convention unique to seven characters
- Use the DLINK option with all OBJECTs created by the Compiler for the application package
- All external functions use EFPL stub
- Main program may have different type of STUB
- All programs need to have a STUB created using a catalogued procedure
- Link edit all created programs together to create package

Example (for MVS)

- Begin with the following three execs

  DLT to drive the process

  CPUTIME to get the CPU time

  INCR simply returns the passed argument

DLT

```
/* REXX * DLT ************************************************************
* Performance Test for DLINK option:
* Invoke external routine INCR 50 times and tell how long it took
************************************************************************/
  n='DLT'
  Parse Version v .          /* Use Parse Version to see if compiled  */
  If left(v,5)='REXXC' Then what=n 'compiled'
                       Else what=n 'interpreted'
  Say what
  num=50

  t0=cputime()
  Call time 'R'
  Say num 'invocations of INCR will be measured'
  Do i=1 To num
    Call incr i
    End
  Say 'This took me' (cputime()-t0) 'CPU-seconds.',
                                          '(elapsed:' time('E')')'
```

One of the aspects of REXX that makes it an easy to use language is the ease with which it can concatenate strings. This is observed in the if statement, where the name of the exec in the variable n is concatenated with the string indicating whether the exec is compiled or interpreted.

Also note that the PARSE VERSION gives the programmer the ability to determine if the exec is running compiled or interpreted. If needed, different logic paths can be followed, depending on whether the exec is being interpreted or run as compiled program.

Similarly Parse Source lets you determine how the exec was invoked and on which system.

CPUTIME

```
/* REXX * CPUTIME ************************************************************
* Return the cpu-time used up so far
*******************************************************************************/
  Parse Version v
  Parse Source  s

  Parse Var s sys .

  Select                            /* Figure out which system we are on */
    When sys='CMS' Then Do
      qt="DIAG"(8,'Q TIME')
      Parse Var qt . 'VIRTCPU=' mm . ':' +1 ss +6
      cpu=mm*60+ss
      End
    When sys='TSO' Then Do
      cpu=sysvar('SYSCPU')
      End
    When wordpos(sys,'PCODS OS/2')>0 Then Do
      t=Time()
      Parse Var t hh ':' mm ':' ss
      cpu=(hh*60+mm)*60+ss
      End
    Otherwise Do
      Say 'System' sys 'is unknown to CPUTIME'
      cpu=0
      End
    End
  If word(s,2)='COMMAND' Then
    Say 'CPU time used so far:' cpu
  Else                              /* When an external routine     */
    Return cpu                      /* Return the CPU time          */
```

INCR

```
/* REXX * INCR ***************************************************************
* Performance Test for DLINK option:
* Return the argument
*******************************************************************************/
  Return arg(1)
```

Building this package

- Interpreted Case

  Normally all three execs reside in SYSEXEC

  Invoked by entering DLT from TSO command line

  CPUTIME and INCR are external routines

- Hence, DLT will need CPPL STUB

- CPUTIME and INCR need EFPL STUB

- All OBJECTs are created with DLINK option

- Catalogued procedure used for each one

Creating the final module

- Link edit all load modules together

  After each has its STUB added

  Using INCLUDE and NAME control cards

- In this example, BJVLIB is the DDNAME of the library containing the programs

- Control cards would be

```
  INCLUDE BJVLIB(DLT)
  INCLUDE BJVLIB(INCR)
  INCLUDE BJVLIB(CPUTIME)
  ENTRY   DLT
NAME DLT(R)
```

Under CMS, simply

```
LOAD DLT INCR CPUTIME
GENMOD DLT
```

Output Comparison

- When Interpreted

```
DLT interpreted
50 invocations of INCR will be measured
This took me 1.30 CPU-seconds. (elapsed: 11.14)
```

- When Compiled using OBJECT and DLINK

```
DLT compiled
50 invocations of INCR will be measured
This took me 0.74 CPU-seconds. (elapsed:  0.89)
```

Under CMS

```
This took me 0.23 CPU-seconds. (elapsed: 1.891623)
vs.
This took me 0.06 CPU-seconds. (elapsed: 0.142037)
```

Significant Performance Improvement

- Interpreted uses 75 % more CPU
- Interpreted is 12.5 times slower in elapsed time

The IBM Compiler and Library for REXX/370

- Open new programming possibilities
- Support both function and application packaging
- Give you more time on your own CPU!
- And we did not even touch
  - Program Documentation
  - Plug-compatibility
  - 31-Bit Capability (VM/XA)
  - New language on old systems