TECH_285
Using AI coding assist with Rexx
Practical techniques, real-world patterns, and debugging power moves

Roland J. Kinsman – roland_kinsman@bmc.com
Solution Engineer at BMC Software

February 26, 2026

TECH_285 Using AI coding assist with Rexx

Abstract

Supercharge Your Rexx Coding with AI: Tips, Tricks, and Real-World Wins

If you’re working with Rexx, chances are you’re deep in the world of systems programming—and that’s exactly where the fun begins. I use Rexx regularly and have discovered how AI tools can supercharge productivity, streamline debugging, and even inspire clever coding shortcuts. In this session, I’ll share practical Rexx tips, surprising AI-assisted techniques, and real-world examples that can make your Rexx experience faster, smarter, and more enjoyable. Whether you're a Rexx veteran or just curious about its potential in the age of AI, come join the conversation!

Who This Session Is For

- Rexx practitioners, sysprogs, and automation engineers
- Anyone curious how modern AI assistants can amplify Rexx workflows
- Attendees comfortable with JCL/ISPF basics (helpful but not required)

What You'll Learn

- Prompt patterns that produce reliable, runnable Rexx
- Defensive debugging: catch uninitialized variables
- Quick access to jobname/jobid for logging and diagnostics
- Data matching with stems: fast, SQL-like joins
- Inline data and self-contained ISPF dialogs
- Practical patterns for blanks and wildcards

TECH_285 Using AI coding assist with Rexx

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

Session Agenda

AI-assisted workflows

- Prompting Rexx with context
- Live demo (VS Code + Copilot)

Rexx language power moves

- Debugging with SIGNAL NOVALUE
- Jobname/Jobid from storage
- Matching data with stems
- Inline data via labels
- Embedding ISPF panels/messages
- Handling blanks & ABBREV wildcards

Resource

Github repository: https://github.com/RJKinsman-BMC/Rexx-tricks

Using AI to enhance Rexx development

- Use VS Code with GitHub Copilot or a chat assistant to draft Rexx quickly
- Always ask for comments, explainers, and ISPF context in the prompt
- Keep a small library of tested snippets you can paste into prompts (see GitHub repo)

Using AI to enhance Rexx development

<u>My experience</u>

Before AI, I was on my own. I did okay.

ChatGPT was good.

CoPilot is better.

Using AI to enhance Rexx development

Live Demo

VS Code + Copilot: generate, refine, and explain Rexx in minutes.

Tip: Ask for “why this line is needed” to surface hidden assumptions.

SIGNAL NOVALUE – catch uninitialized variables

By default, the value of a variable is the name in upper case, so for example, the following code:

```
say What is your name?
```

Displays the following:

WHAT IS YOUR NAME?

Uninitialized variables are easy to miss and difficult to debug.

Enabling NOVALUE/SYNTAX/FAILURE/HALT surfaces errors immediately.

SIGNAL NOVALUE – catch uninitialized variables

```
SIGNAL ON NOVALUE                         /* Enable the conditions and  */
SIGNAL ON SYNTAX                          /* point at reasonable        */
SIGNAL ON FAILURE                         /* condition handling         */
SIGNAL ON HALT
/* . . . Program body . . . */
EXIT 0
```

At the end of the code, we add the following:

SIGNAL NOVALUE – catch uninitialized variables

```
/*
   Condition handling

   Taken from "What's wrong with Rexx?" by Walter Pachl, IBM Retiree
   Downloaded from https://www.rexxla.org/presentations/2004/walterp.pdf

*/
NOVALUE:
SAY 'Novalue raised in line' SIGL   /* line in error number       */
SAY SOURCELINE(SIGL)                /* and text                   */
SAY 'Variable' CONDITION('D')       /* the bad variable reference */
SIGNAL Lookaround                   /* common interactive code    */

SYNTAX:
syntax_rc = rc                      /* Save the rc                */
SAY 'Syntax raised in line' SIGL    /* line in error number       */
SAY SOURCELINE(SIGL)                /* and text                   */
                                    /* the error code and message */
SAY 'rc='syntax_rc '('errortext(syntax_rc)')'

HALT:
Lookaround:                         /* common interactive code    */
IF user_interface = 'FORE' THEN DO  /* when running in foreground */
   SAY 'You can look around now.'   /* tell user what he can do   */
   TRACE ?R                         /* start interactive trace    */
   NOP                              /* and cause the first prompt */
   END
xrc=16 /* REXX return code */
SIGNAL QUIT
```

SIGNAL NOVALUE – catch uninitialized variables

If we use an uninitialized variable, our code ends with the following message:

```
Novalue raised in line 99
SAY 'parmvalue1' parmvalue1
Variable PARMVALUE1
CPREXX Return Code=16
```

SIGNAL NOVALUE – catch uninitialized variables

This, however, can lead to a problem. What if you can't know if a variable was initialized? Use the following code:

```
/* Test a variable to see if it was initialized */
IF 'VAR' = SYMBOL(PARMVALUE1') THEN sw.PARMVALUE1 = 1
```

SIGNAL NOVALUE – catch uninitialized variables

Questions?

Getting jobname/jobid into a variable

This one is fairly simple. In some situations (logging, diagnostics, etc.) you may want to know the jobname/jobid in which you are executing.

These details are in storage, and you can access them with the following code:

```
ascbd=C2D(STORAGE(224,4))
assbd=C2D(STORAGE(D2X(ascbd+X2D(150)),4))
jsabd=C2D(STORAGE(D2X(assbd+X2D('a8')),4))

jobid=STORAGE(D2X(jsabd+X2D(14)),8)

jobName=STRIP(STORAGE(D2X(jsabd+X2D(1C)),8))
```

Getting jobname/jobid into a variable

Questions?

Matching data with stem variables

The concept:

Use stems as associative arrays keyed by a composite key.

Populate a presence flag (0/1). Join by testing the flag for each key in the other set.

Matching data with stem variables

First, build the lookup stem

```
ndvinv. = 0 /* initialize the stem for ndvinv */

/* populate the stem */
ndvinv_key = ndvinv_listElm.i ndvinv_listType.i ndvinv_listSys.i,
    ndvinv_listSubs.i ndvinv_listStg.i ndvinv_listVVLL.i
ndvinv_key.i = ndvinv_key
ndvinv.ndvinv_key.ndvinv_stem = 1
```

Matching data with stem variables

Lookup during iteration:

```
DO i = 1 TO cpinv.0
   cpinv_key = cpinv_key.i
   IF \ndvinv.cpinv_key.ndvinv_stem THEN DO
```

We’re looping through a stem of “cpinv” and for each one, if we don’t get a match on ndvinv, we do something.

Note that ndvinv_stem is initialized to zero unless there’s a value. If there is, it gets a 1, so it can be used as a switch.

Matching data with stem variables

Questions?

Inline data using labels

Using inline data, you can bundle a small set of data within your Rexx script.

Here is an example:

Inline data using labels

```
SIGNAL A074_XTN_DTDo;A074_XTN_DTDo:
xtn_line=SIGL+2;SIGNAL A074_XTN_DTDoo
''
'<!--'
'<!DOCTYPE Transactions ['
'<!ELEMENT Transactions (Transaction+)>'
'<!ELEMENT Transaction (Date,Rule_TN?,Description,Categories*,'
'Cat_Rule_ID?,Cat_rule_split?,Tran_Type,'
'Tran_Num,Description_Original,Month)>'
'<!ELEMENT Date (#PCDATA)>'
'<!ELEMENT Rule_TN (#PCDATA)>'
'<!ELEMENT Month (#PCDATA)>'
']>'
''
'-->'
''
A074_XTN_DTDoo:
DO xtn_sub=xtn_line WHILE SOURCELINE(xtn_sub) \= 'A074_XTN_DTDoo:'
   INTERPRET 'xtno_rec = ' SOURCELINE(xtn_sub);CALL G110_xtn_out
END
```

Inline data using labels

It skips over the inline block and processes it in a loop

It uses several features of Rexx:

SIGNAL command

The SIGL special variable

The SOURCELINE function

Inline data using labels

Questions?

Imbedding ISPF components into your Rexx (panels, messages, etc)

Imbedding ISPF components into your Rexx takes the “inline” concept a step further.

Benefit: rather than dealing with multiple members in multiple libraries, you can have an entire ISPF dialog in a single Rexx program.

No dependency on multiple members in multiple libraries.

Imbedding ISPF components into your Rexx (panels, messages, etc)

Start by allocating a temporary file, for example…

```
'alloc fi(otfispf)' ISPF_unit 'new reuse dir(5) space(1,1)',
   'tracks dsorg(po) recfm(f,b) lrecl(80) blksize(3280)'
```

Imbedding ISPF components into your Rexx (panels, messages, etc.)

Next, assign that library to different ISPF data:

```
ADDRESS ISPEXEC

/* Issue the LIBDEFs */
'LIBDEF ISPMLIB LIBRARY ID(OTFISPF) STACK'
'LIBDEF ISPPLIB LIBRARY ID(OTFISPF) STACK'
'LIBDEF ISPSLIB LIBRARY ID(OTFISPF) STACK'

'CONTROL REFLIST NOUPDATE'
'LMINIT DATAID(ISPFID) DDNAME(otfispf)'
'CONTROL REFLIST UPDATE'
'LMOPEN DATAID('ISPFID') OPTION(OUTPUT)'
```

Imbedding ISPF components into your Rexx (panels, messages, etc)

Finally, populate the library using the inline technique:

```
SIGNAL C104_Member;C104_Member:
mbr_line=SIGL+2;SIGNAL C104_Membero
MSG010 'ENTER THE PROGRAM NAME' .ALARM=YES
'THE PROGRAM NAME IS REQUIRED TO RETRIEVE THE LISTING'
MSG011 'FILE CHOICE IS BLANK' .ALARM=YES
'THE SELECTED FILE CHOICE, NUMBER &lfnum, IS BLANK'
MSG012 'DATASET NOT VALID' .ALARM=YES
'SELECTED DATASET &endsn IS NOT VALID'
MSG013 'DATASET NOT VALID' .ALARM=YES
'SELECTED DATASET &userddio IS NOT VALID'
MSG014 'Enter Required Field' .ALARM=YES
'Enter your Compuware Source Listing (DDIO) File'
MSG015 'Press Enter to continue' .ALARM=YES
'Press Enter to specify source listing file'
C104_Membero:
DO sub=mbr_line WHILE SOURCELINE(sub) ^= 'C104_Membero:'
   line = SOURCELINE(sub)
   'LMPUT DATAID('ISPFID') DATALOC(LINE) MODE(INVAR) DATALEN(80)'
END
'LMMADD DATAID('ISPFID') MEMBER(MSG01)'
```

Imbedding ISPF components into your Rexx (panels, messages, etc)

Questions?

Account for blank values – placeholder

```
dsn_line = applist.i,
    WORD(WLBMTYPE '*',1),
    WORD(WLBMCLAS '*',1),
    WORD(WLBLVL '*',1),
    WORD(WLBNAME '*',1)
```

If any variable contains a null value, the result will be an asterisk instead.

Account for blank values – placeholder

Questions?

Wildcard matching – using the ABBREV function

I use this one all the time:

```
IF ABBREV('CANCEL',zcmd,3) THEN SIGNAL QUIT
```

It’s self-explanatory. If the user types anything beginning with the string, “can” then we quit.

Wildcard matching – using the ABBREV function

Questions?

Your feedback is important!

Submit a session evaluation for each session you attend:

www.share.org/evaluation

THANK YOU VERY MUCH!
WHAT ARE YOUR FAVORITE REXX CODING TECHNIQUES?