Design of the Emerging REXX Standard

Brian Marks
ANSI

This talk is in four sections.

The "Dull but informative" section is about the history of the committee, mandate, membership and progress so far.

The "Rexx users won't care" section is about technical aspects of writing the definition.

"What is new about errors" will explain the changes and extensions there.

"What is new about Command I/O" will explain our attempt to provide uniformity in the way data is exchanged across the interface between Rexx and system commands.

==

At the 1990 Rexx symposium in Stanford, a panel of Rexx experts was asked "Would a standard for Rexx be a good thing, and would you contribute to creating one?". Based on the strong support expressed there, a proposal was made and the first meeting of the X3J18 committee was held in January 1991.

The proposal that X3 voted on when setting up the committee had this key paragraph:

"The scope of the standard will be the second edition of the Cowlishaw book, plus consideration of implementation experience. The scope may be altered as necessary to promote portability, reliability, maintainability and efficient execution of REXX programs on a variety of computing systems."

Note that this mandate doesn't allow the committee to add things just because users would like them. While we all know of extensions that have been frequently requested, like date conversions and new ways of using stemmed variables, the committee doesn't consider them for the first standard. (Although we do have separate discussions with a view to a subsequent standard.)

We have just had our ninth meeting. The attendance has varied, but typically is about ten.

Membership is a big investment. The membership fee is only 300 dollars a year but the cost lies in travel and in the member's time. The four meetings a year are spread over the USA (to even out costs) and each lasts several days. In theory, there is even more cost in the time member's spend on X3J18 between meetings but in practice they are all professionals with other jobs, making it difficult for them to provide that extra contribution.

The balance of the committee has more participants that are primarily implementers than participants that are solely users. Of course, the implementers also represent users, but we would like more members from the user community.

The work has progressed to the point where we have most of a draft of the standard, although there is a lot of detail work to do. There are two extremes of looking at that: "How can so little be done in two years?" (if you count elapsed time) or "A great result for a month's work" (if you only count the meetings). The truth, no doubt, is somewhere in between.

==

The audience for a standard comprises implementers and people who want to validate implementations. Such people understand Rexx so the standard doesn't have to be a tutorial; it does need to be rigorous and complete. The actual users of Rexx are not so interested in how the draft is written, only in its content, which will be reflected in the manuals for users.

There are some languages, like VDL and Z, which were specifically designed for writing formal definitions. One of these could have been used to write the standard, but we chose to write much of the standard in Rexx.

Superficially this is circular, using a language to define itself, but in practice it is a bootstrapping exercise. The parts not written in Rexx provide the foundation for parts which are written in Rexx.

The syntax of programs can be specified using grammars in the familiar BNF notation. We use one for the tokens and one for the higher level constructs. There is an interaction between these grammars because of the detection of keywords and implied concatenations. Detecting keywords is a good example of something that a standardizing committee has to work hard on but the user doesn't care what the answer is.

For example, the rule about 'DO' keywords is different in the "Red Book" from the "Blue Book":

"The sub-keywords WHILE and UNTIL are reserved within a DO instruction, in that they cannot be used as symbols in any of the expressions. Similarly, TO, BY, and FOR cannot be used in expri, exprt, exprb, or exprf. FOREVER is also reserved, but only if it immediately follows the keyword DO."

"The sub-keywords TO, BY, FOR, WHILE, and UNTIL are reserved within a DO instruction, in that they cannot name variables in the expression(s) but they may be used as the name of the control variable. FOREVER is similarly reserved, but only if it immediately follows the keyword DO."

It is doubtful if any Rexx programmer cares at all, but it has to be defined. To avoid special cases for individual keywords, the committee has come up with the rule "If it could be a keyword it is". That means that the BNF and the rules for detecting labels and assignments are all applied in a left-to-right way; if then a potential keyword occurs and there has been nothing to contradict the possibility of it being a keyword then it is a keyword. We don't think this rule changes the behaviour of any existing error-free programs and it guarantees the definition hasn't missed any case.

==

The "Red Book" goes further than many language definitions by specifying the exact wording of all the error messages. However, it doesn't always say when a particular message is produced. There are some cases where the book says "It is an error..." and leaves it to the implementer to choose the message from the given set of messages. Not all implementers have made the same choice. This is another topic where the actual Rexx user probably doesn't care much what is done; but I think the committee is right to standardize the choice.

We do this for syntax errors (that is programs that contradict the BNF) by annotating the BNF to show what message should be produced for failures at any point. We do it for execution errors by writing tests in the definition for particular numbered errors.

The number of distinct error messages defined in the "Red Book" is far fewer than the number of places where the standard will detect an error. So the simplest approach would lead to the same message being given for several places where it was detected. For example, many different things wrong with a call might be detected, but they would all lead to syntax error 40.

The committee has decided to extend Rexx with subcodes - this is within our mandate because of the portability and maintainability considerations in error detection. So there will be, for instance,

Error 40.16 Argument <number> to routine <name> must be non-null

Existing programs might be dependent on actually testing the major error number, the 40 in this case, so that part of the language isn't changed. The subcode only comes into play if the program chooses to ask for it or on termination messages.

==

The ability to issue commands is a central pillar of Rexx strengths. The way of issuing commands is well-defined; a clause which is an expression. However, the way in which the commands access Rexx data and the way in which Rexx accesses the results of commands are not well-defined. There are mechanisms that can be used, such as streams, the stack, and the variable pool but how the commands actually do their I/O has always been left up to the implementation.

The committee has added an extension which will be available on all systems that conform to the standard, and hence provide a more portable way of using commands.

The ADDRESS instruction now has extra options:

```
ADDRESS ... WITH INPUT STREAM MyOne OUTPUT STREAM MyTwo

ADDRESS ... WITH INPUT STEM RxOut. OUTPUT STEM RxIn.
```

This way of using stemmed variables has been a popular convention in conjunction with implementers' "extras" to Rexx so the committee is not forcing some completely untested invention on users.

==

That is the end of the presentation except to remind you that this is merely an account of a snapshot in the development process - the content of the standard when eventually approved could be entirely different.

Dr B L Marks
Room G.0.023, MP 212
IBM UK Labs Ltd, Hursley Park, Winchester, Hampshire, England
Tel 44-962-844433 Ext 6643 Internet:marks@winvmd.vnet.ibm.com