# REXX IN THE CICS ENVIRONMENT

DAVID I SHRIVER
IBM

# REXX in the CICS Environment

May 5, 1992

David I. Shriver

IBM
Mailstop 01-03-50
5 West Kirkwood Blvd.
Roanoke, TX 76299-0001
(817) 962-4142

**Third REXX Symposium, Anapolis, Maryland** 

ABSTRACT:

CICS/REXX is an IBM internal implementation of REXX, the IBM SAA Procedures Language, under CICS/MVS and CICS/ESA. Specifically, it provides REXX environment support under CICS for both the TSO/E Version 2 REXX interpreter and the REXX/370 compiler. This environment support includes interface routines for storage management, I/O handling and other miscellaneous REXX facilities. It also includes providing a command-level interface to CICS from REXX, and also provides interfaces to other CICS based products, such as IBM's OfficeVision/MVS.

# Contents

# CICS/REXX Overview

## Copyright



## Trademarks

The following terms used in this paper, are trademarks or service marks of IBM Corporation in the United States or other countries:

AIX, CICS/ESA, CICS/MVS, DB2, GDDM, IBM, QMF, MVS/ESA, OfficeVision, OS/2, PROFS, REXX

## Disclaimer

This discussion of REXX under CICS does not imply that IBM either does, or does not, have plans to incorporate all, or part of, this function into a product.

## Purpose of this paper

The purpose of this paper is to share information on an internal IBM implementation of REXX under CICS so as to promote technical discussion and generate customer feedback.

## Function/Feature Highlights

As follows are some of the highlight features of CICS/REXX:

- Full REXX language support under MVS CICS
- Support for both compiled and interpreted EXECs
- CICS based text editor for REXX EXECs and data
- VSAM based file system for REXX EXECs and data
- Support for popular EXEC CICS commands (not complete yet)
- Support for Subcommands written in REXX
- Support for application macros, written in REXX
- High-level client/server architecture support
- Command definition of REXX Subcommands
- Flat/Universal default REXX Subcommand space
- Transparent CICS Psuedo-conversational terminal support
- Support for system and user profile EXECs
- Shared EXECs in virtual storage

- Nested INCLUDE support in EXEC Loader
- EXEC Suspend/Resume support
- REXX interface to OfficeVision/MVS
- Compatibility support for several popular VM/CMS commands

## Full REXX language support under MVS CICS

CICS/REXX is currently at REXX language level 3.46 and supports all REXX language statements and built-in functions, as described for MVS in the *SAA Common Programming Interface Procedures Language Reference, SC26-4358.*

## Support for both compiled and interpreted EXECs

CICS/REXX includes support for both interpreted and compiled EXECs. Compiled and interpreted EXECs can be freely intermixed. Such a combination is powerful because the use of the interpreter provides a very productive development environment (quick development cycle, source level interactive debug, CICS based development) whereas the compiler allows the developed REXX code to be later optimized for the performance requirements of critical production systems. Since compiled and interpreted REXX EXECs can be intermixed transparently, compilation can be done selectively on those modules that need it most, and the replacement of interpreted REXX EXECs can be done gradually, without affecting system function.

## CICS based text editor for REXX EXECs and data

KEDIT, a full function text editor, similar to the VM/CMS XEDIT and TSO ISPF/PDF editors is provided as part of CICS/REXX, so EXECs can be written and modified directly under CICS, and from CICS based application platforms, such as OfficeVision/MVS.

## VSAM based file system for REXX EXECs and data

CICS/REXX includes a REXX file system that is hierarchically structured (similar to OS/2, AIX and the VM Shared File System), and automatically provides each REXX user with a file system in which to store EXECs and data. There is a FILELIST utility to facilitate working with this file system, the KEDIT editor will support editing members of this file system, and EXECs to be run are loaded from this file system. This library (file) system is VSAM RRDS based for performance, security and portability reasons.

## Support for popular EXEC CICS commands (not complete yet)

Support for several EXEC CICS commands is already included in CICS/REXX, and support for all popular CICS Command Level commands is planned.

## Support for Subcommands written in REXX

CICS/REXX supports the ability for users to write new REXX subcommands in REXX. These subcommands do not function as nested REXX EXECs, and unlike nested REXX EXECs will have the ability to get and set the values of REXX variables in the user EXEC that invoked them. Thus subcommands written in REXX can have similar capabilities as subcommands written in Assembler or other languages. Therefore subcommands can be quickly written in REXX to speed systems development (in a building block structure), and then can selectively be rewritten in Assembler, for example, at a later date, as performance requirements dictate. Or they may simply be compiled with the REXX compiler.

## Support for application macros, written in REXX

One of the strongest uses for REXX is to support the extension of existing applications via Application Macros. This provides a natural mechanism for the extension of product or application capability, and does so in a natural building block fashion. Since REXX Application Macros are separate from application code, this means they can be effectively created by application users, with little chance of causing application failure.

## High-level client/server architecture support

CICS/REXX includes built-in client/server architecture support to facilitate the use of this important new technology in systems development and to help enable a higher level of host involvement in Enterprise-wide computing solutions.

## Command definition of REXX Subcommands

CICS/REXX includes as one of its basic facilities, the ability for systems administrators and users to easily and dynamically define new REXX subcommands, either on a system-wide or user-by-user basis. One of the greatest strengths of REXX is its ability to be interfaced cleanly with other products, applications and system services. The goal for providing a command definition facility for new or existing subcommands is to facilitate the rapid and consistent high-level integration of various products and services together through the use of REXX. REXX subcommand definition is accomplished though the CICS/REXX DEFCMD and DEFSCMD subcommands.

## Flat/Universal default REXX Subcommand space

The CICS/REXX subcommand definition facility also optionally supports the use of a flat (or universal) REXX subcommand space. This would be consistent with the REXX goal of maintaining simplicity and naturalness. With this support, all REXX subcommands (which might span interfaces for multiple applications) would be mapped into one default subcommand environment. This would allow one global and consistent subcommand set to be provided and documented, and would free programmers from having to understand which subcommand environment a subcommand exists in, and it would remove the need to be constantly switching subcommand environments (switching environments is accomplished with the ADDRESS statement).

## Transparent CICS Pseudo-conversational terminal support

CICS/REXX supports both conversation and pseudo-conversational terminal I/O in REXX based transactions. Transparent, underlying pseudo-conversational support is provided if the PSEUDO ON subcommand is specified in an EXEC. This means that a program written in REXX can be switched between conversational and pseudo-conversational without changing the program structure.

## Support for system and user profile EXECs

To facilitate CICS/REXX system and user environment tailoring, CICS/REXX will attempt to execute a SYSPROF EXEC and user PROFILE EXECs if they exist. The SYSPROF EXEC must exist in the system base directory and is invoked before the first user EXEC runs after a CICS system restart. A user's PROFILE EXEC (if it exists in that user's base directory) will be invoked before the first EXEC is invoked for this user (after a CICS system restart).

## Shared EXECs in virtual storage

CICS/REXX supports both shared and unshared copies of REXX EXECs residing in virtual storage. Preloaded shared EXECs improve interactive response time of REXX applications, and sharing reduces the total virtual storage requirement.

## Nested INCLUDE support in EXEC Loader

Often in real world REXX programming, a programmer is torn between making a function or subroutine written in REXX, internal or external to a REXX application. There are significant performance and variable sharing advantages to making a subroutine internal. But there is a major drawback if this subroutine is to be shared by several REXX EXECs. Duplicate copies must be placed in all programs that use the subroutine and it is a nightmare trying to update all of these copies and to keep them the same, whenever a change is made to a subroutine. CICS/REXX nested INCLUDE support improves this situation by allowing one or more INCLUDE statements to be placed in REXX source files so that subroutines can be maintained as separate external files but be included as internal routines at EXEC load time. An additional opportunity is that only one copy of the source for a particular subroutine needs to be loaded into virtual storage, no matter how many EXECs are using it as an internal routine.

## EXEC Suspend/Resume support

When CICS/REXX is used as a Procedures Language under CICS, there are times that EXECs are used to contain command lists of CICS commands (applications) to be STARTed. Since these CICS transactions often require a terminal to be available before they can run, a way is need to cause the transaction the EXEC is running under to end to free up the terminal, causing the EXEC to be temporarily suspended so it can be resumed later at the point after it was suspended. The CICS/REXX SUSPEND subcommand provides this capability.

## REXX interface to OfficeVision/MVS and ASF Version 2

OfficeVision/MVS and ASF Version 2 provide CICS based Application Integration platforms. Applications may be integrated with each other or with Office functions, for added value. CICS/REXX has special support to facilitate REXX EXECs being invoked from OfficeVision/MVS (or from ASF Version 2) and/or OfficeVision/MVS services being invoked from REXX EXECs in a CICS environment.

## Compatibility support for several popular VM/CMS commands

Compatibility support for several important VM/CMS commands has been provided in CICS/REXX to make it easier to port or migrate VM based EXECs to a CICS environment. This helps preserve customer investments in VM/CMS EXECs when such a migration is necessary, it helps facilitate the porting of a considerable amount of VM/CMS REXX based software to the CICS environment, and helps preserve investments in VM/CMS training and allows VM/CMS users to come up to speed more quickly in the CICS/REXX environment.

# CICS/REXX Benefits

## Business Solutions

CICS/REXX is an ideal system to use to deliver superior, valuable, and appropriate business solutions, in a much more timely and cost effective manner.

CICS/REXX is an excellent platform for the delivery of CICS based business solutions for the following reasons:

- ***CICS/REXX is a simpler, uniform, self contained development environment***

  To use CICS/REXX, a new programmer no longer has to learn TSO, ISPF, JCL, COBOL and much of the technical detail of CICS (such as the proper use the translator).

  For both new and experienced programmers, there is no longer the need to constantly switch back and forth between TSO and CICS, all the while flipping between several manuals for needed system and development information.

  CICS/REXX is a uniform, self contained system that supports development directly under CICS and provides everything the average CICS developer needs in one manageable package.

- ***CICS/REXX allows solutions to be delivered sooner***

  There is a combination of benefits that CICS/REXX delivers to cause major gains in application productivity and reduced delivery time. The REXX language alone has proven to be a major boost to application productivity because of its high level, simplicity, strong parsing and naturalness. On top of that, the synergy of an interpreter/compiler combination is a strong addition. The interpreter provides a very quick development cycle and provides excellent source-level interactive debugging capability. Experience has proven a ten-fold improvement in productivity, when using REXX over conventional languages and techniques, to be a conservative figure. The ability to deliver business solutions more quickly is an important advantage in today's competitive marketplace.

- ***CICS/REXX makes practical highly incremental development***

  One of the biggest advantages of the fact that CICS/REXX includes support for a REXX interpreter as well as a compiler, is that the interpreter, with its quick, natural development cycle and excellent source-based interactive debugging make it feasible to switch to an Incremental Development Methodology. This is also sometimes called a Prototyping Development Methodology.

  REXX is of a sufficiently high level to be a powerful language for quick and expressive prototyping, and because of the compiler and the robustness of the language, is also suitable for serious application development. This provides an ideal situation where prototypes can be quickly developed to test system feasibility, to gather requirements, to get customer involvement, and can then be "grown" into useful production systems.

  This approach bypasses the nasty surprises of finding late in the development cycle that the project isn't technically feasible, of delivering a system that isn't what the customers want (or even what they thought they were going to get), or of major schedule overruns without any deliverables. And a final nice benefit of incremental development is that it has the tendency to test the code much more thoroughly during development, usually resulting in much higher quality code.

- ***CICS/REXX applications are easier to maintain and support***

  REXX based applications, being high-level in nature, are usually smaller than comparable applications in other languages (in lines of code) and are easier to read. And the interactive source level debug capability of the REXX interpreter makes it easier to locate and fix problems, and to deliver enhancements. This equates to a cheaper, more effective support of REXX based applications.

- ***CICS/REXX is useable by business people***

  Quite often business people who best understand the business and their needed solutions have ideas as to ways to modify, customize, or enhance applications that they use. But when they discover the difficulty involved and the investment in education required, they often give up in frustration. But those who have persevered have often delivered some of the most timely and on-target solutions. One of the greatest strengths of REXX is its simplicity and naturalness on one hand, and its powerful capability, on the other hand. CICS/REXX will make it possible for CICS application users to more extensively customize and even extend their applications, without requiring a programmer. This will provide more timely, on-target solutions, and will free real programmers up for involvement in more strategic projects.

  This is in line with what many industry analysts believe is a fundamental shift happening in the model for application development within Fortune 1000 companies. Business is organizing into more autonomous units, competitive pressures have increased (demanding quicker solutions), and new technology such as workstations and Client/Server computing, have made it feasible for much application development to be moved from central MIS to line-of-business organizations.

- ***CICS/REXX makes complex systems manageable***

  One of the design goals of REXX has always been to bend over backwards to make programming simple and natural for the REXX programmer, even if this makes things complicated for the REXX implementer. The simplistic power of REXX makes it a good candidate for today's complex business systems, because it simplifies them and thus makes them more manageable.

  CICS/REXX organizes (breaks down) complex systems in several related ways to make them more manageable. One is that it promotes a natural building block approach made up of EXECs, application macros, and subcommands transparently implemented in a variety of languages. In close relationship to these, is built-in Client/Server computing support that encourages greater host involvement in the Enterprise-wide Client/Server Distributed Computing model, with all of the many benefits this entails. Another strength of CICS/REXX in this arena, is the facilities it has for integrating multiple applications, products, and system facilities together into one seamless package, from a user perspective, which greatly simplifies systems development efforts.

**The KEDIT story:** The KEDIT text editor was written so as to be externally similar to the IBM XEDIT and ISPF/PDF editors, so as to minimize user retraining needs.

KEDIT is an excellent example of the sophistication that is possible with REXX based applications under CICS/REXX. And it it is a good example of the development productivity improvements that are possible.

KEDIT was written completely in REXX (except for some general purpose primitives it uses that are written in Assembler, as will be the case with most REXX applications) by Kevin Wriston, who was new to REXX. Kevin wrote a useable editor (which he used for his own REXX development) in three weeks, and has spent a total of about three person months, developing KEDIT. And the finished product is only about 1000 lines of REXX code, a mere fraction of the XEDIT Assembler code.

The other nice thing is the quickness with which Kevin can respond to requests for changes or enhancements to KEDIT (often quicker than the average programmer can go get a cup of coffee).

Kevin recently added REXX macro support to KEDIT, a demonstration that under CICS/REXX, applications written in REXX, can also support application macros, written in REXX, an important new capability.

## Investment Protection

The IBM MVS CICS computing environment has one of, if not the, largest concentration of customer production applications and data, **in the world.** There has been tremendous customer investment in CICS based mainframe systems, CICS based application development, data collection for CICS based systems, and employee education relating to the use and support of CICS based systems. CICS/REXX helps to preserve and enhance the usefulness of this investment.

Not only does CICS/REXX enhance the delivery of traditional CICS based production applications, it makes the CICS environment suitable for a broader range of information processing activities. With CICS/REXX, it is now practical to also perform end-user computing, prototyping, and application development, directly within the CICS environment.

Also, CICS/REXX, which currently runs under MVS CICS, was designed so it can be later ported to provide REXX support for CICS running under OS/2, AIX, VSE and OS/400. One goal is to provide consistent REXX support across these environments, so as to preserve customer investments. Another is to facilitate the use of cooperative processing, between these environments.

## User Productivity

CICS/REXX can enhance CICS user productivity in several ways:

- Allows simpler, but more flexible application customization by typical users. This allows them to more effectively tailor these applications to their individual business needs.
- Advanced users will be able to make application enhancements that normally would have been reserved for professional application developers. This has the effect of providing solutions needed to improve productivity and satisfy business needs more quickly. It also reduces the demand on application developers for application changes and frees them to work on more significant long range efforts.
- Facilitates the use of a prototyping methodology. This means that the users of an application in development participate very closely in the application development process (if they do not own the process outright). The end result is that the users, who have the best understanding of the business and their needs can better ensure that the application solution delivered matches their needs. This close involvement will also have the added benefit that the human factor needs (useability) of the user audience will also tend to be addressed in the application, enhancing their productivity.

## Growth Enablement

Because CICS/REXX reduces the complexity of application development and maintenance, it makes it feasible to develop and support larger and more complex systems. This is true because:

- REXX is a high level language whose major emphasis has been to be natural to use and to free its user (the programmer) from any unnecessary detail. Thus REXX programs tend to be shorter and easier to follow.
- REXX encourages the use of a more manageable building block approach to systems development. The integrated Client/Server and dynamic subcommand definition capabilities of CICS/REXX even further enhance this.
- Major productivity improvements achieved by using the powerful interactive source level debugging capability and the quick development cycle of the REXX interpreter will make larger, more sophisticated development efforts feasible.

## Systems Management

One of the major strengths of REXX is its usefulness as a Procedures Language. When used in this way, it can automate sequences of CICS system and application Systems Management activities, providing greater productivity and reliability.

Also, since CICS/REXX supports application development (and testing) directly under CICS, systems management can be greatly simplified. For example, the need for many CICS developers to have a TSO userid, could be removed, in many situations. Reducing the volume of TSO userids that need to be administered and managed would equate to an overall reduction in systems management activities.

# CICS/REXX General Architecture/Implementation

## General Design Goals

Some specific design goals/objectives for this project were:

- Provide the CICS or OfficeVision/MVS user or application developer/integrator with a simple but powerful self contained REXX based environment with the necessary interfaces to productively accomplish application development, application integration and customization.
- Provide a high-level, easy to use, REXX interface to the existing CICS command level facilities so as to improve the productivity of existing, experienced CICS developers.
- Provide a high-level, easy to use alternative programming environment that removes the need for casual programmers (or users) to learn the CICS environment.
- Bring product interfaces together, in one, self contained place for both ease of use and added synergy.
- Provide a flexible CICS REXX implementation that can be easily customized, tailored or extended by customers for their own unique needs.
- Capitalize on new REXX/370 compiler, C/370 Version 2 and other products
- Provide an environment conducive to the building block approach to code development. One of important needs in this area is to allow administrators and users to replace one type of building block or primitive with one written in a different language or with a different name without having to change the programs that reference it. Support interfaces to multiple programming languages.
- Provide an architecture capable of supporting large complex systems
- Perform acceptably for use in large production CICS environments
- Provide security sufficient for CICS production environments
- Exploit CICS/ESA and MVS/ESA when available

## Basic structure of REXX running under CICS

CICS/REXX support provides a program called REXX which is used to load and invoke REXX EXECs within a CICS region. This program uses the Clearly Differentiated Programming Interfaces (CDPI) of TSO/E Version 2 REXX to define a new CICS specific REXX language processor environment for the user EXEC, and then invokes the EXEC. The REXX program also contains several REXX replaceable routines to handle all REXX storage requests, line-mode I/O and various other functions. On the very first invocation of the REXX program within a CICS region, a REXX system server is automatically started, under its own REXX environment control block. Thereafter, the REXX system server receives notification before the invocation and after the termination of each user EXEC invoked by the REXX program. The REXX system server is a shared server that all REXX user execs can route requests to, by ADDRESSing the subcommand environment SYSTEM. The GLOBALV global variable command support that is provided is an example of using the system server to add additional subcommands to REXX.

## REXX EXEC invocation

- EXECs invoked from a terminal

  REXX EXECs are invoked by a CICS/REXX program named REXX. A CICS transaction id must be defined for this program. If the tran id is REXX then the name of the EXECs and its arguments follow on the command line. For example: REXX MYEXEC ABC will invoke the REXX EXEC MYEXEC and pass it the string ABC as an argument. If a transaction id other than REXX is associated with the REXX program, the name of the EXEC that is invoked is the same as the transaction id.

- EXECs invoked by a START command

  The REXX transaction associated with the REXX program may be invoked with the EXEC CICS START command. If start data is provided, that is passed to the EXEC as an argument. The name of the EXEC to invoke is normally expected to be provided in the start data.

- EXECs invoked by a LINK or XCTL

  The REXX program, when invoked by a LINK or XCTL, will attempt to find the name of the REXX EXEC to invoke in the COMMAREA, if one is available. The entire COMMAREA will also be passed to the EXEC as an argument.

## Where EXECs execute

CICS/REXX EXECs are executed as part of the CICS task that invokes them, within the CICS region. The REXX interpreter is fully reentrant and runs above the 16 MB line (AMODE=31,RMODE=ANY).

## How EXECs are located and loaded

The directories of specified REXX libraries are searched, in concatenation sequence in an attempt to locate an EXEC. If it is located, it is read into storage and control is given to the REXX interpreter to invoke it. Before REXX libraries are searched, there is first a check to see if the EXEC is already loaded in storage, and if so, since REXX EXECs are re-entrant, control is given immediately to the REXX interpreter.

## How EXECs are edited

CICS/REXX includes a CICS-based text editor, wich is similar to the IBM XEDIT and ISPF/PDF editors, to edit EXECs and data files, directly under CICS.

## Control of EXEC execution search order

A PATH subcommand is provided to control the search order of REXX File System directories. The directories specified in the PATH command are searched after the current directory (specified by the CD command).

## REXX EXEC File System structure

Execs are currently stored as members a VSAM-based REXX file system. Some features of the REXX File System are: are:

- Hierarchical Directory structure (like OS/2, AIX, VM SFS)
- No need to register new users
- No need to register individual EXECs
- Basic support without an External Security Manager
- Import/Export to MVS Partitioned Datasets
- Management functions for members (COPY, DELETE, RENAME)
- FILELIST file directory interface utility
- An EXECIO-like I/O utility (FSIO)
- Supports insertion of records in middle of files
- Maximum records per member is approx. 2**32 minus 2
- Maximum record length is 2**32 minus 2
- Maximum VSAM datasets per a RFS filepool is 511
- Number of filepools is limited by system storage
- Execute-only support by library and by member
- Support for authorized REXX libraries (for authorized primitives)

## Support of standard REXX features

### SAY and TRACE statements

The REXX SAY and TRACE terminal I/O output statements use CICS Terminal Control Support to provide simulated line-mode output.

### PULL and PARSE EXTERNAL statements

The REXX PULL and PARSE EXTERNAL terminal I/O input statements use CICS Terminal Control Support to provide simulated line-mode input.

### REXX stack support

Same as TSO/E Version 2 REXX

### REXX function support

CICS/REXX supports the same built-in function set as TSO/E Version 2 REXX with the following exceptions. The USERID function will return a 1 to 8 character CICS userid if the user is signed on, otherwise it will return blanks. The STORAGE function, which allows a REXX user to freely display and/or modify the virtual storage of the CICS region will be disabled or restricted.

### REXX Function Packages

The function packages provided with TSO/E REXX that are not TSO specific, are provided and system administrators will have the ability to define/add additional function packages using standard documented interfaces.

## REXX Subcommand Environment Support

REXX subcommand environments that are currently available (to use with the REXX ADDRESS command) are CICS, COMMAND, MVS and SYSTEM.

### Invoking another EXEC as a subcommand

EXECs may be invoked as subcommands using the new client/server support (described later in this document).

### Invoking CICS load modules as user provided subcommands

Support is provided for site provided subcommands, in the form of CICS LOAD modules (loaded using an EXEC CICS LINK) to be defined using the DEFCMD and DEFSCMD commands.

### Adding REXX host subcommand environments

Support is provided to allow new CICS/REXX host subcommand environments to be added and supported in a variety of languages, including REXX. This is done using the DEFCMD and DEFSCMD subcommands, or by using the standard documented TSO/E REXX interfaces.

## Support of standard CICS features/facilities

### CICS mapped I/O support

Support is not yet available for CICS BMS I/O commands as REXX subcommands in the CICS subcommand environment.

### Dataset I/O Services

Verbs for standard CICS dataset I/O services commands are planned as REXX subcommands.

### Interfaces to CICS Facilities and Services

From within the ADDRESS CICS subcommand environment, support is planned for most popular CICS commands (as defined in the CICS Application Programmer's Reference Guide). Currently support is provided for the function provided by the following CICS Command Level commands:

- EXEC CICS SEND
- EXEC CICS SEND TEXT
- EXEC CICS RECEIVE
- EXEC CICS READQ TS

- EXEC CICS WRITEQ TS
- EXEC CICS DELETEQ TS
- EXEC CICS ASSIGN USERID
- EXEC CICS READ RRN
- EXEC CICS WRITE RRN
- EXEC CICS REWRITE RRN
- EXEC CICS DELETE RRN
- EXEC CICS UNLOCK
- EXEC CICS START
- EXEC CICS LINK
- EXEC CICS XCTL
- EXEC CICS SUSPEND

### Invoking user applications from EXECs

EXEC CICS START, LINK and XCTL commands are currently supported.

### REXX interfaces to CICS temporary & transient storage queues

Currently subcommand support exist for reading, writing and deleting CICS temporary storage queues from REXX.

### Pseudo-conversational transaction support

CICS pseudo-conversational support for REXX EXECs is provided. If this support is enabled, an EXEC CICS RETURN TRANSID could is automatically issued before each CICS RECEIVE, the execution state of the EXEC preserved and the REXX transaction ended. The the next terminal I/O event would cause the REXX transaction to be re-invoked and the EXEC to be resumed at the next statement after the RECEIVE.

## REXX EXEC Suspend/Resume support

CICS/REXX support includes a primitive (subcommand) to suspend the execution of the EXEC and causes the invoking transaction to end, allowing another transaction to run, attaching the terminal. The next time the REXX program is invoked, the suspended transaction will resume the suspended EXEC. Any start data passed is placed in the reserved REXX variable SDATA.

## Interfaces to other programming languages

The goal is to provide interfaces to COBOL, C/370, Assembler, and maybe PL/I.

## Security

Normal CICS interfaces to the MVS System Authorization Facility (SAF) will create the framework for CICS/REXX security. Advanced security needs for REXX subcommand and client/server security is expected to be provided under CICS/ESA using the EXEC CICS QUERY SECURITY command.

## Performance discussion

Because of the production nature of CICS, much emphasis is being placed on performance. There are many design choices that can affect security. These include how REXX environments are defined, how the REXX file system structure is implemented, how security interfaces are implemented, how much virtual storage is given to an EXEC at invocation.

REXX is an excellent performer, especially for an interpreter, because it internally uses many sophisticated techniques, such as look-aside tables, for good performance. REXX has proven itself to be a reasonable performer in the VM arena as much of PROFS code is written in REXX. Many PROFS systems today support thousands of users in production. Another point to note, is that although REXX EXECs are interpreted, most of the actual processing for the typical application is spent executing REXX subcommands which do most of the actual work. These primitives can be and usually are written in a compiled language, when performance is an important consideration. Usually, for the majority of small to medium scale CICS applications the productivity benefits of using REXX far outweigh the performance penalty of using REXX. A similar analogy is customers using DB2 vs IMS. DB2 often requires more resources, but the benefits more than outweigh the added processing cost. The net result is that DB2 users are happy because they are more productive.

The best news from a performance perspective, is that the IBM REXX/370 compiler will work with CICS/REXX, whenever performance critical applications need it.

## Miscellaneous features

A TERMID subcommand has been provided to return the four character terminal identifier of a CICS user.

A RETRIEVE PF key has been setup to retrieve the last input line enter using line-mode I/O.

## Supported Environments and prerequisites

CICS/REXX currently runs under CICS/MVS and CICS/ESA. CICS/REXX requires that TSO/E V2.0 or later be installed and, if the REXX/370 compiler is used, in addition to the interpreter, then TSO/E V2.3.1 or later must be installed. Certain advanced functions, such as the planned REXX interface utilizing the programming interface of CICS 3.2 for Resource Definition Online, will only supported under CICS/ESA.

## National language and DBCS support

The full range of DBCS functions and handling techniques that are included in TSO/E Version 2 REXX are available to the CICS/REXX user.

It is expected that the national languages supported for CICS/REXX will match those supported for TSO/E Version 2. Refer to announcement 288-694, dated December 6, 1988. The support for national languages will likely lag the initial American/English language support.

## Building block S/W development - Common Interface Routine

One of the foundation architecture pieces of the CICS/REXX support code is a routine called the Common Interface Routine (CIR).

The purpose of this routine is to allow transparency and flexibility as to the implementation method and language of programs that make up software systems under CICS/REXX. That is, systems implementers should be free to create systems comprised of a mixture of traditional and client/server interpreted REXX EXECs, compiled REXX EXECs, COBOL, C and Assembler language programs. And they should be later free to change the language or implementation method of a program without affecting the correct functioning of the system as a whole.

This is accomplished by having REXX and all other programs that wish to participate in this system, to call the Common Interface Routine whenever control (or a client/server request) needs to be passed to another program. The CIR then determines from a table or data dictionary, the type and language of the target program, so it can invoke it (or pass the request to it) properly.

All programs that use the Common Interface Routine must use a consistent format for the passing of parameters (or requests) to the target and for the returning of any resulting data.

The use of the Common Interface Routine does not require the use of client/server computing, but is a closely related technique.

# CICS/REXX Client/Server Architecture

## High-level Client/Server support

A major new thrust of data processing is in the area of client/server processing. Many realize that this method of computing holds much promise for accomplishing their computing needs in a more responsive and cost-effective manner, especially in today's ever more increasingly work station based computing environments. However, many realize the promise and recognize the opportunity, but lack the tools to effectively accomplish their goals. The goal here is to augment the general CICS/REXX environment with a high-level, easy to use, REXX-based client/server processing support that will make it feasible for customers to easily implement client/server processing applications that they could have never before considered, better utilizing mainframe and workstation resources.

### Client/Server Design goals

- Allow REXX servers to service multiple REXX clients, which may be located on a variety of remote systems (long-term)
- Provide an identity service to dynamically track and route requests and responses between servers and requestors on multiple systems by server name. It should support the concept of local and global resource management (long-term)
- Provide security interfaces to effectively and efficiently control authorization of access and communication between servers and requestors.
- Support both synchronous and asynchronous communication between servers and requestors
- Very high level, easy to use but flexible REXX interface to this server/requestor support
- Support parallel communication activities between a client and a server, at least separate command and data sockets/sessions (long term)
- Provide syncpoint and recovery capability
- Good performance through use of efficient techniques
- General enough in design to have a wide variety of uses

## Current Client/Server Implementation

- Provides client/server support within a CICS region
- High-level REXX based
- Provides a common shared REXX system server
- Supports requests from both REXX and assembler clients
- Supports automatic server initiation

Requests are sent from a REXX client to a server as follows:

ADDRESS serverid 'request'

The server waits on and receives requests from a client by issuing the 'WAITREQ' subcommand. The server is suspended until a client request arrives (which is placed in the reserved REXX variable REQUEST).

There are subcommands available to REXX servers to get or replace the contents of client REXX variables, by name.

The security characteristics/authority level of a client are automatically inherited by the server while it is processing the request from that client.

# CICS/REXX OfficeVision/MVS Environment Support

## REXX EXECS for Application Integration

Currently OfficeVision/MVS provides the capability for the user to add new menu items or commands along with their associated CICS applications to their OfficeVision/MVS desktop. This is done using the Application Services component online administration utility to define new applications (represented by Application Type Descriptor (ATD) definitions).

Since REXX EXECs are invoked as a normal CICS program or transaction, REXX EXECs can easily be invoked from the OfficeVision/MVS desktop.

REXX EXECs under CICS/REXX are enabled to use the OfficeVision/MVS System Interface Block (SIB). The REXX program (or transaction) can be STARTed or XCTLed with a SIB passed to indicate what EXEC to invoke and also where to transfer control to when the EXEC has finished its processing. REXX EXECs can also pass an outbound SIB to OfficeVision/MVS or another SIB enabled application. This should greatly facilitate OfficeVision/MVS based Application Integration.

For security reasons, CICS/REXX will not allow a user EXEC to pass a SIB to OfficeVision/MVS unless that user is already signed on.

## REXX EXECS as exits

It is planned to support the use of REXX EXECs as exit programs for OfficeVision/MVS and other CICS based applications. It is the exit implementer's responsibility to determine if a REXX exit would be suitable as an exit (for performance reasons, especially when an interpreted EXEC is used). However, it should be noted that REXX EXECs are successfully being used to code exits routines for production applications running under VM/CMS.

# CICS/REXX Interfaces to other products

One of the strengths of REXX is the ease with which high-level interfaces to other products can be provided. It seems a logical next step to add interfaces from CICS REXX to DB2, GDDM and other products, on an as needed basis.

## Description of interface to DB2

This interface would be similar to the REXX to SQL interface available under VM but would use the CICS dynamic SQL interface to DB2.

## Description of interface to GDDM

This interface would function essentially the same as the existing GDDM/REXX product under VM.

# CICS/REXX CMS Environment Compatibility/Emulation

To facilitate the migrating of systems and the porting of software from VM/CMS to MVS CICS, the following VM/CMS capabilities are provided:

- Global variable support compatible with the VM/CMS GLOBALV command has been provided.
- Full-screen terminal I/O support, compatible with the VM/CMS WAITREAD command has been provided.
- EXECIO command is supported for I/O to sequential datasets
- Xedit editor limited compatibility

# Summary

## Prototype development experience

My prototype development experience has led me to the conclusion that it is feasible to do a good implementation of REXX under CICS. However what will do more to guarantee a good implementation of REXX under CICS, more than anything else, is the feedback, input and participation of IBM customers in this effort.

## Much more than just another language for CICS

I hope that by now you have come to the conclusion that CICS/REXX is much more than just another CICS language. That it is rather the beginning of a new environment with the potential to dramatically improve the way that we work.

# Appendix - Sample CICS/REXX screens

## Sample FILELIST screen

```
USER=WRISTON  DIRECTORY=/
CMD   FILENAME FILETYPE ATTRIBUTES RECORDS BLOCKS DATE     TIME
 • • • Top Of File • • •
      TEST2    EXEC     FILE       5       1      03/27/92 10:31:04
      TEST1    EXEC     FILE       11      1      03/27/92 10:30:29
      GENID    EXEC     FILE       7       1      03/13/92 09:00:37
      SECURITY EXEC     FILE       21      1      03/13/92 08:59:31
      TEST     EXEC     FILE       14      1      03/11/92 15:06:53
      FSIO     LIB      FILE       493     3      03/11/92 08:42:04
      WINDOWS  EXEC     FILE       50      0      03/10/92 10:46:19
      KEDIT    EXEC     FILE       1278    5      03/10/92 08:49:14
      USERS              DIR       1       1      03/10/92 08:49:10
 • • • End Of File • • •









COMMAND ===>
```

## Sample KEDIT Screen

```
WRISTON  /USERS/WRISTON NONAME SIZE=0 LINE=0 CHANGED=NO
K E D I T  1.1.9 -  CICS Editor









00000 *•*•*•Top Of File *•*•*•
00001 *•*•*•End Of File *•*•*•









COMMAND ===>
```

## DEMO EXEC

### Source listing

```
EDIT ---- SHRIVER.REXX(DEMO) - 01.08 ------------------------ COLUMNS 001 072
COMMAND ===>                                                  SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000100 /* REXX */
000200 TRACE 'o'
000201 arg parms
000202 parse source . . . . . environm
000203
000204 say '***-----------------------------------------------------***'
000205 SAY '*** This is a test REXX program running under' environm '***'
000206 say '***-----------------------------------------------------***'
000207 say
000208 say 'The arguments passed were:' parms
000209 say
000210
000211 /* example of REXX standard line-mode input */
000212 say 'What is your name?'
000213 parse pull name
000214 say
000215 say 'Welcome to' environm 'REXX,' name
000216 say
000217
F13=HELP     F14=SPLIT    F15=END      F16=RETURN   F17=RFIND    F18=RCHANGE
F19=UP       F20=DOWN     F21=SWAP     F22=LEFT     F23=RIGHT    F24=RETRIEVE
```

```
EDIT ---- SHRIVER.REXX(DEMO) - 01.08 ------------------------ COLUMNS 001 072
COMMAND ===>                                                  SCROLL ===> PAGE
000218 /* example of nesting */
000219 address mvs
000220 'demo2 xxx'
000221
000222 /* example of CICS subcommands */
000223 address cics
000224 'TERMID'      /* get my CICS terminal id */
000225 outbuf = sba(22 12)||'This is fullscreen output to terminal' termid
000226
000227 /* perform CICS fullscreen output */
000228 'SEND' outbuf /* do a CICS EXEC CICS SEND */
000229 outbuf = sba(23 12) || 'Now try some fullscreen input'
000230 'SEND' outbuf
000231
000232 /* perform CICS fullscreen input */
000233 'WAITREAD'    /* do an EXEC CICS RECEIVE and parse into vars */
000234 say 'The AID key that was pressed =' waitread.1
000235 say 'The cursor was at (Row Col):' subword(Waitread.2,2,2)
000236 say 'The data that was entered (Row Col Data):' subword(waitread.3,2)
000237 say
F13=HELP     F14=SPLIT    F15=END      F16=RETURN   F17=RFIND    F18=RCHANGE
F19=UP       F20=DOWN     F21=SWAP     F22=LEFT     F23=RIGHT    F24=RETRIEVE
```

```
EDIT ---- SHRIVER.REXX(DEMO) - 01.08 ------------------------ COLUMNS 001 072
COMMAND ===>                                                  SCROLL ===> PAGE
000238
000239 /* example of using the system server */
000240 say 'send a GLOBALV SET and GET commands to the system server'
000241 address system
000242 'GLOBALV SELECT GROUP1 SET VAR1 test data'
000243 'GLOBALV SELECT GROUP1 GET VAR1'
000244 say 'The contents of VAR1 =' var1
000245 say
000246
000247 trace 'o'  /* don't want to trace large loop */
000248 /* example of stand REXX line-mode output with more than 1 screen */
000249 do i = 1 to 20
000250   do j = 1 to 1000
000251     a = 5
000252   end
000253   say i*1000 'assignment statements have been executed'
000254 end
000255
000256 say
000257 /* show that built-in REXX functions are available */
F13=HELP     F14=SPLIT    F15=END      F16=RETURN   F17=RFIND    F18=RCHANGE
F19=UP       F20=DOWN     F21=SWAP     F22=LEFT     F23=RIGHT    F24=RETRIEVE
```

```
EDIT ---- SHRIVER.REXX(DEMO) - 01.08 ------------------------ COLUMNS 001 072
COMMAND ===>                                                  SCROLL ===> PAGE
000258 say "Today's date is" date('w') date()
000260 say 'The time is' time()
000400 EXIT
****** ***************************** BOTTOM OF DATA ******************************
F13=HELP     F14=SPLIT    F15=END      F16=RETURN   F17=RFIND    F18=RCHANGE
F19=UP       F20=DOWN     F21=SWAP     F22=LEFT     F23=RIGHT    F24=RETRIEVE
```

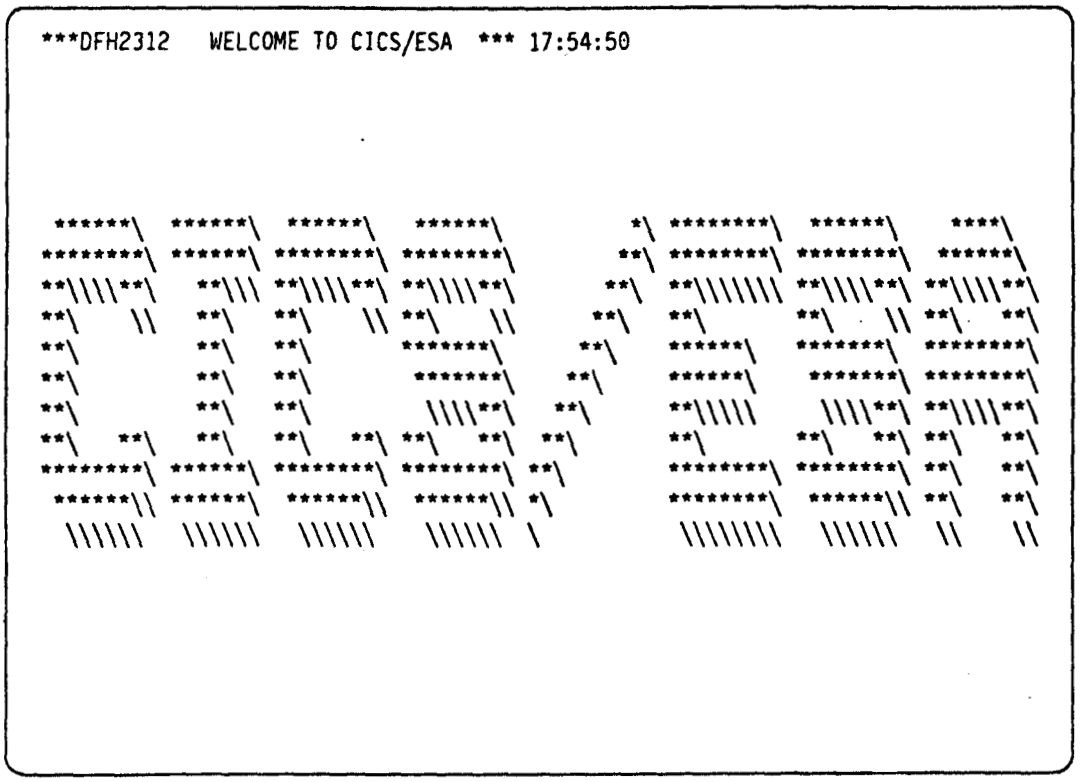
```
EDIT ---- SHRIVER.REXX(DEMO2) - 01.03 ---------------------- COLUMNS 001 072
COMMAND ===>                                                 SCROLL ===> PAGE
****** ***************************** TOP OF DATA ********************************
000001 /* nest level 2 */
000010 trace 'r'
000100 say 'you entered demo2 exec'
000110 address mvs
000200 'demo3 yyy'
000300 exit
****** **************************** BOTTOM OF DATA ******************************



F13=HELP     F14=SPLIT    F15=END      F16=RETURN   F17=RFIND    F18=RCHANGE
F19=UP       F20=DOWN     F21=SWAP     F22=LEFT     F23=RIGHT    F24=RETRIEVE
```

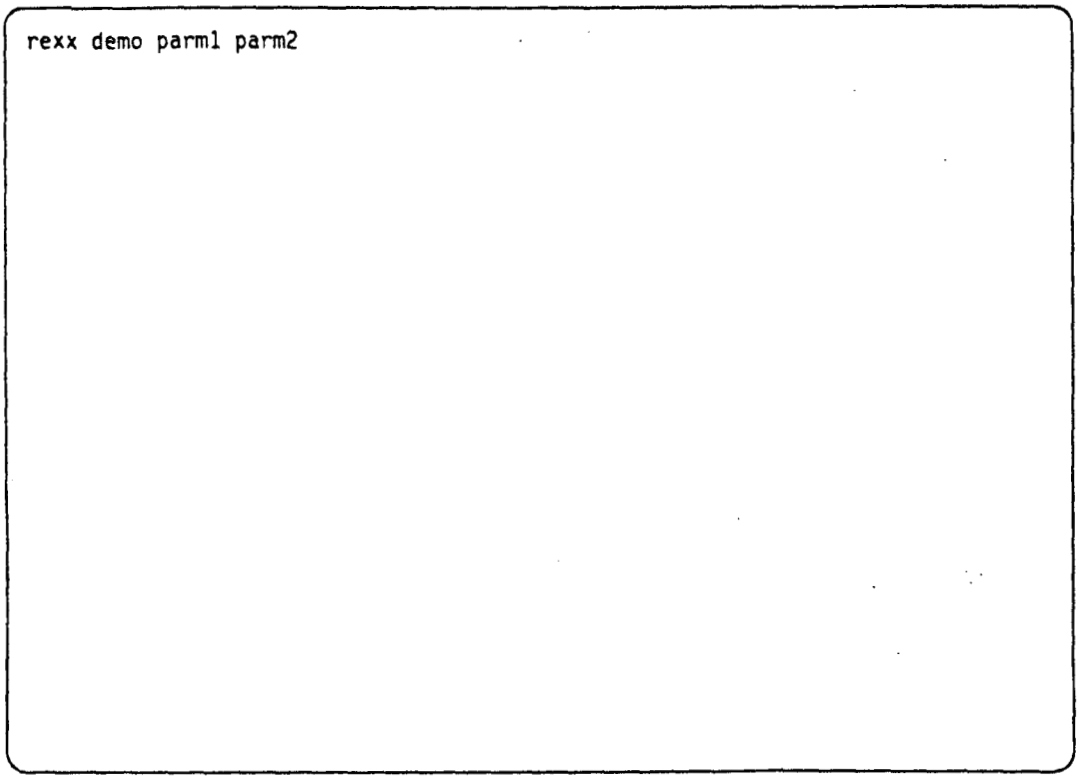
```
EDIT ---- SHRIVER.REXX(DEMO3) - 01.03 ---------------------- COLUMNS 001 072
COMMAND ===>                                                 SCROLL ===> PAGE
****** ***************************** TOP OF DATA ********************************
000010 /* next level 3 */
000020 address mvs
000100 say 'you entered demo3 exec'
000200 'demo4'
****** **************************** BOTTOM OF DATA ******************************



F13=HELP     F14=SPLIT    F15=END      F16=RETURN   F17=RFIND    F18=RCHANGE
F19=UP       F20=DOWN     F21=SWAP     F22=LEFT     F23=RIGHT    F24=RETRIEVE
```

```
EDIT ---- SHRIVER.REXX(DEMO4) - 01.03 ----------------------- COLUMNS 001 072
COMMAND ===>                                                  SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000010 /* nest level 4 */
000100 say 'you entered demo4 exec'
000110 address mvs 'demo5'
****** **************************** BOTTOM OF DATA ****************************

F13=HELP     F14=SPLIT    F15=END      F16=RETURN   F17=RFIND    F18=RCHANGE
F19=UP       F20=DOWN     F21=SWAP     F22=LEFT     F23=RIGHT    F24=RETRIEVE
```

```
EDIT ---- SHRIVER.REXX(DEMO5) - 01.00 ----------------------- COLUMNS 001 072
COMMAND ===>                                                  SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000200 /* rexx */
000300 say 'you entered demo5 exec'
****** **************************** BOTTOM OF DATA ****************************

F13=HELP     F14=SPLIT    F15=END      F16=RETURN   F17=RFIND    F18=RCHANGE
F19=UP       F20=DOWN     F21=SWAP     F22=LEFT     F23=RIGHT    F24=RETRIEVE
```

## Execution with trace off


***DFH2312 WELCOME TO CICS/ESA *** 17:54:50


rexx demo parm1 parm2

```
***-----------------------------------------------------***
*** This is a test REXX program running under MVS CICS ***
***-----------------------------------------------------***

The arguments passed were: PARM1 PARM2

What is your name?









Dave                                                  READ
```

```
***-----------------------------------------------------***
*** This is a test REXX program running under MVS CICS ***
***-----------------------------------------------------***

The arguments passed were: PARM1 PARM2

What is your name?
Dave

Welcome to MVS CICS REXX, Dave

     3 *-* say 'you entered demo2 exec'
       >>>   "you entered demo2 exec"
you entered demo2 exec
     4 *-* address mvs
     5 *-* 'demo3 yyy'
       >>>   "demo3 yyy"
you entered demo3 exec
you entered demo4 exec
you entered demo5 exec
     6 *-* exit
          This is fullscreen output to terminal 04G1
          Now try some fullscreen input
```

```
***-----------------------------------------------------***
*** This is a test REXX program running under MVS CICS ***
***-----------------------------------------------------***

The arguments passed were: PARM1 PARM2

What is your name?
Dave

Welcome to MVS CICS REXX, Dave

     3 *-* say 'you entered demo2 exec'
       >>>   "you entered demo2 exec"
you entered demo2 exec
     4 *-* address mvs
     5 *-* 'demo3 yyy'
       >>>   "demo3 yyy"
you entered demo3 exec
you entered demo4 exec
you entered demo5 exec
     6 *-* exit
The AID key that was pressed = ENTER
         Now try some fullscreen input
test input                                                    MORE
```

```
The cursor was at (Row Col): 24 16
The data that was entered (Row Col Data): 24 2 test input

send a GLOBALV SET and GET commands to the system server
The contents of VAR1 = test data

1000 assignment statements have been executed
2000 assignment statements have been executed
3000 assignment statements have been executed
4000 assignment statements have been executed
5000 assignment statements have been executed
6000 assignment statements have been executed
7000 assignment statements have been executed
8000 assignment statements have been executed
9000 assignment statements have been executed
10000 assignment statements have been executed
11000 assignment statements have been executed
12000 assignment statements have been executed
13000 assignment statements have been executed
14000 assignment statements have been executed
15000 assignment statements have been executed
16000 assignment statements have been executed

                                                              MORE
```

```
17000 assignment statements have been executed
18000 assignment statements have been executed
19000 assignment statements have been executed
20000 assignment statements have been executed

Today's date is Tuesday 20 Aug 1991
The time is 17:59:31
Ready; (5.232298)
```

## Execution with trace on

```
rexx demo parm1 parm2
```

```
     3 *-* arg parms
       >>>   "PARM1 PARM2"
     4 *-* parse source . . . . . environm
       >.>   "TSO"
       >.>   "COMMAND"
       >.>   "DEMO"
       >.>   "SYSEXEC"
       >.>   "?"
       >.>   "DEMO"
       >.>   "CICS"
       >>>   "MVS CICS"
     6 *-* say '***-------------------------------------------------------***'
       >>>   "****-----------------------------------------------------****
***-----------------------------------------------------***
     7 *-* SAY '*** This is a test REXX program running under' environm '***'
       >>>   "**** This is a test REXX program running under MVS CICS ****
*** This is a test REXX program running under MVS CICS ***
     8 *-* say '***-------------------------------------------------------***'
       >>>   "****-----------------------------------------------------****
***-----------------------------------------------------***
     9 *-* say


                                                                      MORE
```

```
    10 *-* say 'The arguments passed were:' parms
       >>>   "The arguments passed were: PARM1 PARM2"
The arguments passed were: PARM1 PARM2
    11 *-* say

    13 *-* /* example of REXX standard line-mode input */
    14 *-* say 'What is your name?'
       >>>   "What is your name?"
What is your name?
    15 *-* parse pull name









                                                                      READ
```

```
    10 *-* say 'The arguments passed were:' parms
       >>>   "The arguments passed were: PARM1 PARM2"
The arguments passed were: PARM1 PARM2
    11 *-* say

    13 *-* /* example of REXX standard line-mode input */
    14 *-* say 'What is your name?'
       >>>   "What is your name?"
What is your name?
    15 *-* parse pull name
```

David Shriver READ

```
    10 *-* say 'The arguments passed were:' parms
       >>>   "The arguments passed were: PARM1 PARM2"
The arguments passed were: PARM1 PARM2
    11 *-* say

    13 *-* /* example of REXX standard line-mode input */
    14 *-* say 'What is your name?'
       >>>   "What is your name?"
What is your name?
    15 *-* parse pull name
David Shriver
       >>>   "David Shriver"
    16 *-* say

    17 *-* say 'Welcome to' environm 'REXX,' name
       >>>   "Welcome to MVS CICS REXX, David Shriver"
Welcome to MVS CICS REXX, David Shriver
    18 *-* say

    20 *-* /* example of nesting */
    21 *-* address mvs
    22 *-* 'demo2 xxx'
```

MORE

```
      >>>   "demo2 xxx"
    3 *-* say 'you entered demo2 exec'
      >>>   "you entered demo2 exec"
you entered demo2 exec
    4 *-* address mvs
    5 *-* 'demo3 yyy'
      >>>   "demo3 yyy"
you entered demo3 exec
you entered demo4 exec
you entered demo5 exec
    6 *-* exit
   24 *-* /* example of CICS subcommands */
   25 *-* address cics
   26 *-* 'TERMID'      /* get my CICS terminal id */
      >>>   "TERMID"
   27 *-* outbuf = sba(22 12)||'This is fullscreen output to terminal' termid
      >>>   "?!$This is fullscreen output to terminal 04G1"
   29 *-* /* perform CICS fullscreen output */
   30 *-* 'SEND' outbuf /* do a CICS EXEC CICS SEND */
      >>>   "SEND ?!$This is fullscreen output to terminal 04G1"
   31 *-* outbuf = sba(23 12) || 'Now try some fullscreen input'
      >>>   "?$,Now try some fullscreen input"

                                                                      MORE
```

```
   32 *-* 'SEND' outbuf
      >>>   "SEND ?$,Now try some fullscreen input"
   34 *-* /* perform CICS fullscreen input */
   35 *-* 'WAITREAD'    /* do an EXEC CICS RECEIVE and parse into vars */
      >>>   "WAITREAD"









         Now try some fullscreen input
```

```
    32 *-* 'SEND' outbuf
       >>>   "SEND ?$,Now try some fullscreen input"
    34 *-* /* perform CICS fullscreen input */
    35 *-* 'WAITREAD'    /* do an EXEC CICS RECEIVE and parse into vars */
       >>>   "WAITREAD"
```

```
         Now try some fullscreen input
test input
```

```
    32 *-* 'SEND' outbuf
       >>>   "SEND ?$,Now try some fullscreen input"
    34 *-* /* perform CICS fullscreen input */
    35 *-* 'WAITREAD'    /* do an EXEC CICS RECEIVE and parse into vars */
       >>>   "WAITREAD"
    36 *-* say 'The AID key that was pressed =' waitread.1
       >>>   "The AID key that was pressed = ENTER   "
The AID key that was pressed = ENTER
    37 *-* say 'The cursor was at (Row Col):' subword(Waitread.2,2,2)
       >>>   "The cursor was at (Row Col): 24 12"
The cursor was at (Row Col): 24 12
    38 *-* say 'The data that was entered (Row Col Data):' subword(waitread.3,2
       >>>   "The data that was entered (Row Col Data): 24 2 test input"
The data that was entered (Row Col Data): 24 2 test input
    39 *-* say

    41 *-* /* example of using the system server */
    42 *-* say 'send a GLOBALV SET and GET commands to the system server'
       >>>   "send a GLOBALV SET and GET commands to the system server"
send a GLOBALV SET and GET commands to the system server
    43 *-* address system
    44 *-* 'GLOBALV SELECT GROUP1 SET VAR1 test data'
         Now try some fullscreen input
test input                                                          MORE
```

```
      >>>   "GLOBALV SELECT GROUP1 SET VAR1 test data"
   45 *-* 'GLOBALV SELECT GROUP1 GET VAR1'
      >>>   "GLOBALV SELECT GROUP1 GET VAR1"
   46 *-* say 'The contents of VAR1 =' var1
      >>>   "The contents of VAR1 = test data"
The contents of VAR1 = test data
   47 *-* say

   49 *-* trace 'o'  /* don't want to trace large loop */
1000 assignment statements have been executed
2000 assignment statements have been executed
3000 assignment statements have been executed
4000 assignment statements have been executed
5000 assignment statements have been executed
6000 assignment statements have been executed
7000 assignment statements have been executed
8000 assignment statements have been executed
9000 assignment statements have been executed
10000 assignment statements have been executed
11000 assignment statements have been executed
12000 assignment statements have been executed
13000 assignment statements have been executed

                                                                    MORE
```

```
14000 assignment statements have been executed
15000 assignment statements have been executed
16000 assignment statements have been executed
17000 assignment statements have been executed
18000 assignment statements have been executed
19000 assignment statements have been executed
20000 assignment statements have been executed

Today's date is Tuesday 20 Aug 1991
The time is 18:02:03
Ready; (9.924010)
```

## REX EXEC

## Source listing

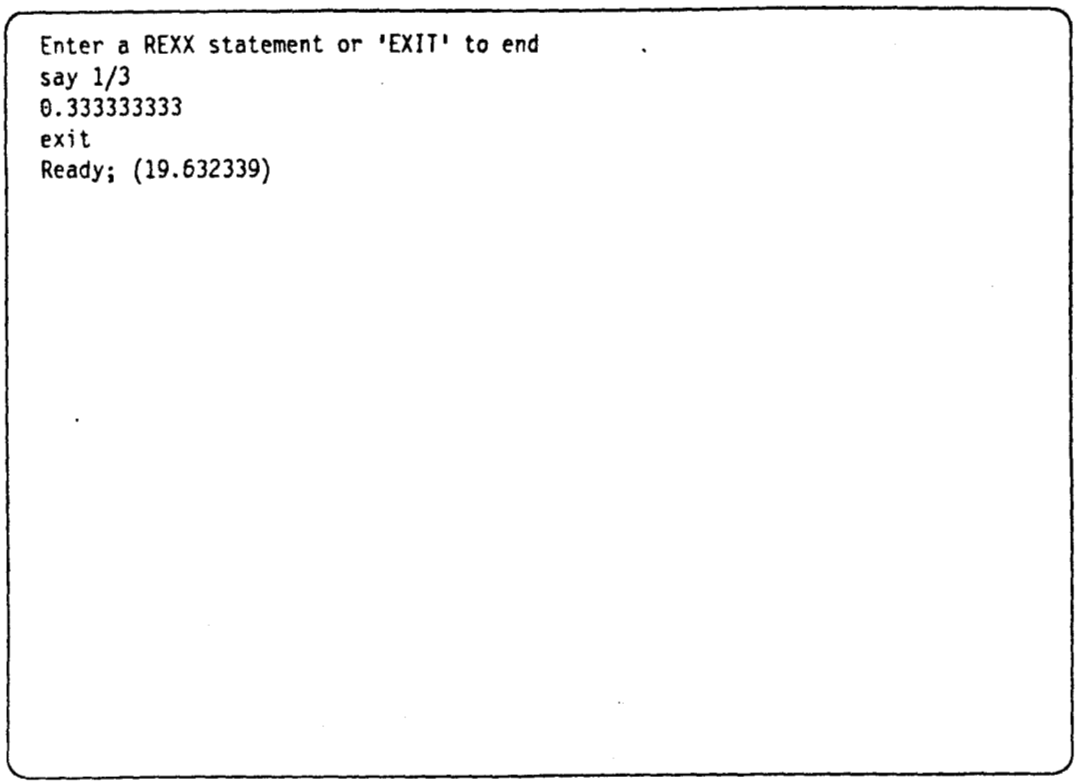
```
EDIT ---- SHRIVER.REXX(REX) - 01.08 ------------------------ MEMBER REX SAVED
COMMAND ===>                                                  SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000001 /* interpretive execution of REXX statements from the terminal */
000002 TRACE 'O'
000003 parse arg arg
000004 signal on error
000005 signal on syntax
000006 SAY "Enter a REXX statement or 'EXIT' to end"
000007 restart:
000008 DO FOREVER
000009   parse external input
000010   if input = '' then SAY "Enter a REXX statement or 'EXIT' to end"
000011   INTERPRET input
000012   if substr(input,1,1) = "'" then say 'RC =' rc';'
000013 END
000014 EXIT
000015 error:
000016       say 'RC =' rc
000017       signal on error
000018       signal restart
000019 syntax:
F13=HELP     F14=SPLIT    F15=END      F16=RETURN   F17=RFIND    F18=RCHANGE
F19=UP       F20=DOWN     F21=SWAP     F22=LEFT     F23=RIGHT    F24=RETRIEVE
```

```
EDIT ---- SHRIVER.REXX(REX) - 01.08 ------------------------ COLUMNS 001 072
COMMAND ===>                                                  SCROLL ===> PAGE
000020       Say 'Syntax error ----  re-enter'
000021       signal on syntax
000022       signal restart
****** ***************************** BOTTOM OF DATA ***************************
F13=HELP     F14=SPLIT    F15=END      F16=RETURN   F17=RFIND    F18=RCHANGE
F19=UP       F20=DOWN     F21=SWAP     F22=LEFT     F23=RIGHT    F24=RETRIEVE
```

## Execution

```
rex
```

```
Enter a REXX statement or 'EXIT' to end
say 1/3
0.333333333


exit                                                          READ
```


Enter a REXX statement or 'EXIT' to end
say 1/3
0.333333333
exit
Ready; (19.632339)