

✓
Received by OSTI

APR 0 6 1989

LBL-PUB--3076

DE89 009221

IBM/VAX Gateway User's Guide

Edward Rosenthal

Administration Division
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

February 1989

MASTER


DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

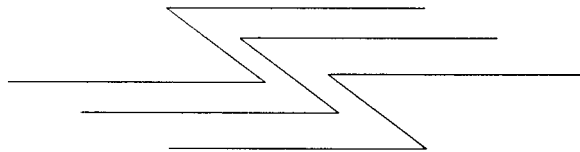
Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. Neither the United States Government nor any agency thereof, nor The Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or The Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or The Regents of the University of California and shall not be used for advertising or product endorsement purposes.

Lawrence Berkeley Laboratory is an equal opportunity employer. PUB-3076/2-89

IBM 4341



VAX/VMS

IBM/VAX GATEWAY USERS' GUIDE

TABLE OF CONTENTS

I. Introduction	1
A. Hardware Configuration	1
1. IBM Hardware	1
2. VAX Hardware	1
B. Software Configuration	2
1. IBM Software	2
2. VAX Software	2
II. Computer Operations	3
A. IBM Operations	3
1. Viewing File Transfer	3
B. VAX Operations	4
1. Viewing the Workstation	4
2. Job Submission	5
III. Applications Programming	8
A. IBM Job Control	8
B. VAX Applications Programming	9
1. RJEPROD	10
2. RJEUTIL	11
a) A Microfiche Utility	12
b) A Deblock Utility	13
IV. Restrictons and Limitations	15
A. Line Availabilty and Speed	15
B. Translate Tables	15
C. Record Length and Data Organization	15
D. Design Features and Implementation Assumptions	15
V. The Ibmrije Login	17
A. Logfiles:	18
VI. References	19
A. Memoranda	19
B. Source Codes	20
C. Configuration Drawing	

IBM/VAX GATEWAY USER'S GUIDE

I. Introduction

The "**Gateway User Guide**" describes the facility available for transferring files from the Data Processing Services IBM mainframe to the Computer Services VAX/VMS cluster.

This guide will benefit both Data Processing Operations Staff and programmers who would like to know more about how the "Gateway" operates and how to take advantage of its capabilities.

The main focus of discussion will be on "how to" after a brief overview of the hardware and software components. A special section concerning the IBMRJE login shows how to use the VAX/VMS system to advantage. Source codes for all the procedures and DCL programs is provided at the end, as well as references made in this document to other sources.

This first section of document will present a hardware and software overview of the current configuration.

A. Hardware Configuration

1. IBM Hardware

The main component of the IBM hardware is a 3705 which is channel connected to the IBM mainframe on one side, and a modem connected to the VAX Microserver on the other.

2. VAX Hardware

The main component of the Vax hardware consists of a Microserver, a small box that has a microvax chip, with a synchronous connection to the IBM Host, and an Ethernet connection to the Vax. The Server is booted from the VAX cluster, and is a general purpose low-level connection capable of serving four synchronous lines, with limitations on the number of high and low speed lines. There is currently one high speed (56000 Bits/sec) line and one low speed (9600 Bits/sec) line serving to connect the two mainframes.

B. Software Configuration

1. IBM Software

Teleprocessing is under control of the MVS/VTAM/JES systems. Communicating to the VAX thru the gateway is the same as talking to an RJE station (BARRHASP). That is, the commands which are used thru the remote lines to the VAX have much the same functionality and limitations as any of the RJE commands.

2. VAX Software

The hardware supplied by DEC is capable of running various network software protocols, including SNA, X25, and DECNET applications. These high level application packages can be loaded from the Ethernet connection to the server, such as IBMRJE or Data Transfer File application packages offered by DEC. The application package running is IBMRJE. The combination of hardware and software give the appearance of a complete RJE station from the point of view of the IBM.

II. Computer Operations

A. IBM Operations

To find out if files are transferring or if there are any files waiting to transfer is done by bringing up one of the ISPF 8.O, 8.H, or 8.I menus. Here is an abbreviated sample output of the menu, showing relevant information only:

1. Viewing File Transfer

JOBNAME	JNUM	C	FORM	DEST	TOT-REC	DEVICE
DPEIRVAX	100	B	V21 R15	2,114	R15.PU1	
DPEIRVAX	100	B	V22 R15	2,112		

The sample output above displays a jobname DPEIRVAX that is using R15. That is, one job (but two files) are using the high speed line to the Vax. Alternatively, if R14 was the DEST, the job would be using the slow speed line. The filename on the VAX depends upon the JOBNAME on the IBM mainframe.

In the "C" column is class B, or Punch Class. Alternatively, if class R was displayed, that would mean the Print Class. Depending upon the class the file is assigned to, the file will end up on the VAX in the PRINT or PUNCH directories.

In the FORM column is V21 and V22, a unique form for each of the files. The FORM-ID is created by using the FORM-ID parameter in the SYSUT2 command. The default form is "STD", which would appear if the programmer left out the parameter. The file created on the VAX/VMS depends upon the FORM-ID for its extension.

In the the DEVICE column is R15.PU1, meaning the file is to be "punched" using the high speed line. This verifies information given in the "C" column since both indicate the "punch" class. Alternatively, if the column displayed R15.PR1, it would be going to the "print queue", and the class should also indicate "R" or "print" class.

To insure the GATEWAY is functioning properly, the IBM console operator can check on the RMT15 and RMT14 lines to see that they are up and active and not drained. Depending on which console or terminal the user or operator is on, to display the current status of the remote print and punch:

\$du,r15

From the information the operator should see whether or not the line is drained or not. If the line is drained or is not active then the command to start the line is used:

\$s,r15.pu1 or \$s,r15.pr1

If files are waiting to transfer but the line is up and active and not drained, then probably the VAX workstation is down. Under these circumstances, then logging in to the VAX is called for. The workstation on the VAX should be checked (see following section) and switched back "ON" if "OFF".

B. VAX Operations

1. Viewing the Workstation

On CSA3 privileged users (IBMRJE) can display and set the status of the workstation.

LOGIN TO CSA3 AS "IBMRJE".

From the CSA3> prompt type "XEQ SNARJE" (press Return)

The following will be displayed:

```
@sy systools:SNARJE
%SNARJE-I-USING, using workstation LBLRJE1
SNARJE>
```

The operator will have "SNARJE>" for a new prompt. It might be noted the operator could use the "help" command to see what is possible under the workstation commands. But for now the operator wants to know the status of the workstation. To do that issue the "show status" command:

SNARJE> SHOW STAT (press Return)

The following (abbreviated output) will be displayed:

Status of workstation LBLRJE1 on 29-NOV-1988

```
Gateway node:      LBLSNA      State:      OFF
Access name: A2          Application: A14JES21
Circuit:          SDLC-0      Logon mode: DECRJE
SNARJE>
```

The operator will notice whether or not the workstation is "ON" or "OFF" and issue the following command to turn it on:

SNARJE> SET WORK/STAT=ON (press Return)

To check the results of the previous command issue the status command once again:

SNARJE> SHOW STAT (press Return)

The workstation should now display "State ON".

To terminate the workstation session:

SNARJE> EXIT (press Return)

To terminate the VAX/VMS session:

CSA3>LO (press Return)

2. Job Submission

The RJE Hardware and Software allows data to be transferred in either direction. You can submit jobs from the VAX to the IBM mainframe. The security policy for this has not yet been fully formed, but assuming that only IBMRJE can submit jobs this would be the way to do it.

To submit a job, issue the SUBMIT command with the following options. The queue must be specified (currently it is called SNARJE\$FAST), and the SNA sub-option must be coded. Here is a sample submit:

```
CSA3> SET DEFAULT [IBMRJE.READER]
CSA3> SUBMIT/QUEUE=SNARJE$FAST/SNA test.jcl
```

Here is a listing of the test.jcl file:

```
//DPEIRVAX JOB (302501),'VAX DPEIRVAX',
//  MSGCLASS=W,
//  CLASS=A
/*ROUTE PRINT ECHO
//VAX2IBM EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT2 DD DSN=DPEIR.VAX.READER,
//  DISP=(OLD,KEEP,KEEP)
//SYSUT1 DD *
THIS IS THE START OF SOME TEST DATA
THIS IS SOME TEST DATA WRITTEN IN LINE FROM THE VAX
THIS IS SOME TEST DATA WRITTEN IN LINE FROM THE VAX
END OF TEST DATA
/*
//
```

The above example submits the file "test.jcl" to the VAX Reader Queue. If the physical line is open and the file is accepted by the IBM mainframe, IEBGENER copies the inline data to the dataset DD name 'DPEIR.VAX.READER'.

Another way to submit a job is to concatenate the JCL to the inline data. Assuming the file data.dat contains the inline data for IEBGENER:

```
CSA3> SUBMIT/QUEUE=SNARJE$FAST/SNA begin.jcl +
      data.dat + end.jcl
```

This approach allows the flexibility to create the data from some other process and to include it inline in the production jcl.

When the VAX job is submitted the job entry number is displayed and the job goes into the queue. When the job begins executing the current **login.com** file (in the home directory) is executed to recreate the defaults for the session. The default directory becomes the home directory. Because of this the full pathname to the files might have to be given if the command is executed from any location other than the home directory i.e.;

```
CSA3> SUBMIT/QUEUE=SNARJE$FAST/SNA -
CSA3>      [ibmrje.reader]begin.jcl -
CSA3> _    + [ibmrje.reader]data.dat -
CSA3> _    + [ibmrje.reader]end.jcl
```

The continuation "-" symbol is actually typed at the command line to allow further typing on the succeeding lines, and the VAX responds with the underscore symbol.

To save keystrokes in submitting a job a command procedure solves the problem. Here is the listing of the procedure **"subreader.com"** located in the **[ibmrje.reader]** directory.

```
Csa3> type subreader.com (press Return)
$if pl .eqs. "" then goto nofilemsg
$file=pl
$! set protection for reader
$set prot=w:r 'file'
$submit/queue=rje$fast/sna -
      [ibmrje.reader]'file'
$ goto exit
$nofilemsg:
$write sys$output " no input file(s)"
$exit:
$exit
$!end of procedure
Csa3¢
```

To submit a file to the queue using the above procedure issue the command:

```
CSA3> @SUBREADER MYFILE
```

Improvements to the **"subreader.com"** procedure could include:

- expansion to three files (concatenating) in place of one.
- checking for existence of the files
- setting protections on all the files
- copying files from other directories

Because of the security precautions associated with sending jobs to the IBM mainframe, there are some corresponding protective measures built into the directories and subdirectories associated with the IBMRJE login. Files can only be submitted when the protection for the file is set properly so that the VAX reader can pick up the file and submit it thru the gateway. To set the protection for the file:

CSA3> SET PROT= W:R FILENAME

After the file is submitted a logfile is created in the home directory which can be looked at to see if the job was transferred successfully to the IBM mainframe. The number of records transferred will be part of the logfile. Any error messages recorded in the logfile should be passed along to the Gateway Coordinator or the VAX/VMS administrators of the DEC software (Helpdesk, or Eric Beals).

III. Applications Programming

This next section describes how to implement the Gateway using the necessary JCL and DCL (Digital Command Language). Many examples are taken from current production jobs and procedures.

A. IBM Job Control

To build an application the programmer should keep in mind the following:

The filenames as it will appear to the VAX:

The FILENAME controlled by the JOBNAME.
The EXTENSION controlled by the FORM-ID.

The destination directory for the file(s) controlled by the sysout class:

SYSOUT=R (PRINT-CLASS)
 files go to: [ibmrje.print]
SYSOUT=B (PUNCH-CLASS).
 files go to: [ibmrje.punch]

Here is a sample job transferring data via the print queue:

```
//PRODUCT JOB(), 'SAMPLE JOB', MSGCLASS=W, CLASS=P,
//      MSGLEVEL=(1,1)
//STEP1      EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN      DD DUMMY
//SYSUT1     DD DSN=MYDATA.FOR.TRANSFER, DISP=OLD
//SYSUT2     DD SYSOUT=R, DEST=RMT15
```

In the above example, a file will arrive on the VAX with the filename **"PRODUCT.STD"**, and be placed in the **"PRINT"** subdirectory, [ibmrje.print].

Here is another example:

```
//XYZIGGY JOB(), 'SAMPLE TWO', MSGCLASS=W, CLASS=P,
//      MSGLEVEL=(1,1)
//STEP1      EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN      DD DUMMY
//SYSUT1     DD DSN=MYDATA.FOR.TRANSFER, DISP=OLD
//SYSUT2     DD SYSOUT=(B,,001),
//      DEST=RMT15
```

In the above example, a file will arrive on the VAX with the filename

"XYZIGGY.001" in the punch subdirectory, [ibmrje.punch].

Thus, in any one job, any number of files with unique names can be sent across the gateway. As noted in the next section, under VAX programming applications, by coordinating the names of incoming files, custom applications can be built for any job.

B. VAX Applications Programming

Once files are transferred to the VAX there remains the task of performing some action upon them. These files may become the input to a Focus FOCEXEC program, or may be printed or copied elsewhere, or microfiched, or sent to a label maker, etc. To provide an automatic interface to any of these several possibilities requires writing DCL or DIGITAL Command Language programs.

Currently there are two main DCL programs which serve as "hooks" for any other applications which may be designed. These jobs are running in "background" batch jobs and become active when their "timers" go off. These programs search for file types or specific files, and when found start up other programs which may be general purpose programs which pass further information along to more specific programs, or may be special production programs customized for the job. These two background jobs are RJEPROD and RJEUTIL.

To display the names of any jobs running:

CSA3> show queue *normal*
The output will look similar to this:

Generic batch queue BATCH\$NORMAL

Batch queue CSA1_NORMAL, on CSA1::

Jobname	Username	Entry	Status
RJEUTIL	IBMRJE	210	Holding until 16-DEC-1988 3:08
RJEPROD	IBMRJE	209	Holding until 16-DEC-1988 3:12

This would be interpreted as showing that the two batch jobs **RJEUTIL** and **RJEPROD** are currently in the batch queue, and they will become active at the time and date shown. If either of these two jobs are not displayed then they must be restarted. To restart the job:

CSA3> SET DEFAULT [IBMRJE.COMFILES]
CSA3> SUBMIT RJEUTIL (OR SUBMIT RJEPROD)

A batch job entry number will be displayed showing the VAX has accepted the job for execution.

1. RJEPROD

The **RJEPROD** command procedure serves as a front end program for any production job coming across the gateway. If, for example, the production job **"EFF0950"** runs on the IBM mainframe, then a file with the filename **"EFF0950"** will eventually arrive in one of the two subdirectories, **[ibmrje.print]** or **[ibmrje.punch]**.

Knowing how the file was sent and with what form-id, the **RJEPROD** program knows which queue to search and what the exact file name will be. While **RJEPROD** is **"active"**, one of the files searched for is **"EFF0950.WEF"**, and when found it calls another program, the **EFF0950** Command Procedure.

Here is an excerpt from the **RJEPROD** command procedure:

```
$eff0950_label:
$FILE = F$SEARCH("[IBMRJE.PUNCH]EFF0950.wef")
$IF FILE .EQS. "" THEN GOTO eff0975_label
$!
$submit/name=effp0950 -
    /queue=csa4_normal -
    [ibmrje.comfiles]eff0950
$!
$eff0975_label:
...
```

The above code begins with a label for the section dealing with **EFF0950**. The file **"eff0950.wef"** is searched for in the punch directory, and if it doesn't exist the process proceeds to the next label. If the file exists then another job is submitted with the name **effp0950**, using the queue and name option. Since **EFF0950** is a **FOCUS** job, it must be forced to run on the **CSA4** machine, since that is where it is licensed.

In the case of **EFF0950**, the job requires the file to be copied to another subdirectory on another disk. After the file is transferred, or copied, a **Focus** batch job must be run, and users must be informed of the changes to the **Focus** Database. Some other programs, also called from **RJEPROD**, may not require as much programming or even testing.

While **RJEPROD** runs, **EFF0950** becomes an active program, runs, finishes, and returns to **RJEPROD**. **RJEPROD** continues to search for other files with their specific predetermined names, and perhaps call other specialized programs acting upon these files. The complete source code for **EFF0950.COM** and **RJEPROD.COM** is included at the end of this guide.

Here is an excerpt from **EFF0950.COM**:

```
$! CHECK TO SEE IF IT ACROSS THE GATEWAY
```

```

$! 1) IF THE FILE IS ABLE TO BE OPENED FOR READ
$! 2) then it can be safely copied
$wef=f$search("eff0950.wef")
$!
$CHECK_LOCK:
$ON WARNING THEN GOTO POSSIBLE_LOCK
$OPEN/READ SCRATCH 'wef'
$CLOSE SCRATCH

```

One of the first things that must be checked is that the file is completely across the gateway before any action is taken on the file. If all is well, the code continues to operate, the file remains untouched, and when RJEPROD is reactivated this code runs again also. By then the file will have finished its transfer, and the job will continue. Later in the code eff0950.wef is deleted from the punch directory, so the process is not duplicated.

To summarize, the requirements for coding production job:

- Write the IBM JCL to create the VAX files.
- Modify the current RJEPROD to:
 - Search for the new files
 - Calls the external job if found
- Create an external job called from RJEPROD
 - The job must insure the file has crossed the gateway
- Fullfill the requirements of the job.

2. RJEUTIL

A more general program is RJEUTIL. It can do general tasks or be made to do such things that could be considered general, for example, copying, printing, and microfiching. It acts as a front end to another series of programs, including the ACTION procedure.

After a discussion of what the program expects in the way of an information file, an extended look at how this is applied to the microfiche problem is considered.

Via JCL, "*information files*", which are files with an extension "INF", must be sent to the punch queue i.e.; [ibmrje.punch]. The information file must include "*data*" and "*action*" statements. This allows the program to search for the data files, and to determine whether the files have come across the gateway. It also does a syntax check to see whether all the statements needed are there, and exits with error messages if it finds anything amiss. Once it does that however, it passes its information along to a second program, called the ACTION Procedure.

The ACTION procedure determines from the information given to it what type of

action is to be performed on the given file. If the action is to fiche the file, it starts up the microfiche program, or if the file is to be printed, it starts up the print program. By this method, any general type of action can be implemented automatically.

a) A Microfiche Utility

The JCL to produce microfiche would have to have at least two steps, one would send the data itself across the gateway, perhaps to the punch queue with the form-id of mfr or std, it doesn't really matter. The other step would have to send an information file telling the vax to produce the fiche. Here is the step and what that file would have to look like:

```
//STEP3      EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//* SEND THIS INFORMATION FILE TO THE PUNCH QUEUE
//* OVERRIDING THE STD EXTENSION.
//SYSUT2     DD SYSOUT=(B,,INF),DEST=RMT15
//SYSUT1     DD      DSN=DPEIR.JCL.FOCUS(INFOFILE),
//           DISP=SHR
//SYSIN      DD DUMMY
/*
//
```

And the information file would have to look similar to this:

```
DATA (F=DPEIR134.STD,SYSOUT=R,JOB=DPEIRJOB),
FICHE(F=DPEIR134.STD,#=1,T=PMA3150A,R=ROSE BLDG 65A)
```

The data and fiche statements should appear each on one line, but they can be in any order, and any number of data files can be indicated, along with their corresponding fiche statements.

The RJEUTIL procedure would check this file to see if data and action statements are present. It also checks the syntax of the data statement.

The DATA statements themselves have three parts;

COLS 1-4 "DATA" signals this is a DATA statement.

Then within parentheses, and separated by commas the following items:

F=FILENAME.EXT Naming the file as it will appear on the VAX.

A SYSOUT=R or SYSOUT=B statement.

A JOB=JOBNAME statement indicating the name of the job.

If the jobname statement is omitted, "JOB" is the default name of the job.

When the RJEUTIL program "awakens", and sees the inf file has arrived, it will read it, check to see if the files have come across the gateway, and then call the program RJEFICHE. The fiche program will check the syntax of the fiche statement, and if all goes well, will send the file to the DICOMED along with the proper commands to satisfy the job.

The form of the fiche statements is:

COLS 1-5 "FICHE"

Then within parentheses the following items separated by commas:

F=FILENAME.EXT the filename to fiche.

= X where x=number of copies 1-9

T=TITLING this title will appear on fiche.

R=ROUTING this information appears on the sticky label.

b) A Deblock Utility

Mentioned in the following section are some of the restrictions and limitations imposed upon the applications due to hardware or software technology, but sometimes there are "work arounds" which circumscribe these circumstances. If the data that is to be transferred across the gateway has a record length of larger than 255 but less than 2048, transfer is made possible by a combination of blocking on the ibm mainframe and deblocking (or unblocking) on the vax side. This section describes how to use these two programs.

The JCL required to block the file to 80 byte records would utilize an assembly language program (Russ Montello), and might go like this:

```
//DPEIRBLK JOB (XXXX), 'BLOCK IT', CLASS=A, MSGCLASS=W,
// NOTIFY=DPEIR
//ROUTE PUNCH RMT15
//*****
//STEP01 EXEC MVS2RJE, DESTIN=R15, HEADER=NOHEAD
//*****
//*
//*****
//STP1.FILEIN DD DSN=DPEIR.TEST.LONG2048, DISP=SHR
//*
//*****
//* STEP02 SEND INFORMATION FILE TO PUNCH QUEUE TO
//* UNBLOCK
//* THE STATEMENTS IN THE PARM FILE LOOK LIKE THIS
//* DATA(F=FILENAME-ON-VAX, SYSOUT=B, JOB=DPEIRBLK)
//* UNBLOCK(F=FILENAME-ON-VAX, RECL=RECORD-LENGTH)
//*****
//STEP02 EXEC PGM=IEBGENER
```

```
//SYSPRINT      DD SYSOUT=*
//SYSIN         DD DUMMY
//SYSUT2        DD SYSOUT=(B,,INF),DEST=RMT15
//SYSUT1        DD DSN=DPEIR.JCL.CNTL(UNBLOCK),DISP=SHR
//
```

Once the file is blocked and sent thru the gateway along with its accompanying information file, the VAX RJEUTIL Program would find the "inf" file and read it, find the verb "unblock", and awaken the unblock command procedure, passing along relevant data to it. The unblock procedure checks the syntax, checks the files, and runs a C Language program which does the actual unblocking. Statistics output on the IBM and on the VAX can be compared for lines blocked and unblocked, and for examining the times the programs ran.

The inf file looks like this:

```
DATA(F=DPEIRBLK.STD,SYSOUT=B,JOB=DPEIRBLK)
UNBLOCK(F=DPEIRBLK.STD,RECL=2048)
```

The data file to unblock must be sent to the punch directory, to insure no job headers would interfere with the process of unblocking the records, and "inf" files are always sent to the punch queue.

The dpeirblk.std file would be sitting in the punch directory when the unblocking process begins, and an output file dpeirblk.out is produced also in the punch directory. There currently is no option to name the output file, but there is no need, since the file will eventually move to a user directory, and could then be renamed.

IV. Restrictons and Limitations

The purpose of this section of the document is to describe some of the restrictions and limitations of the Gateway.

A. Line Availabilty and Speed

The first and formost limitation of the Gateway product is that it is a telecommunications link, subject to unavailability. Even though there are currently two lines, a low speed line and a high-speed line, both of these lines may be unavailable at the time when you need to transfer data.

If the Gateway is unavailable, contact the appropriate operations staff. It may be possible to start the line and/or activate a resource very quickly. As a backup procedure, you can create a tape and load it on the VAX.

B. Translate Tables

A second limitation arises from the nature of the computer systems, where the IBM is an EBCDIC machine and the DEC is an ASCII machine. In other words, in addition to data transmissions, the Gateway performs data conversion services. Due to the differences between the EBCDIC and ASCII character sets, there cannot be a one for one character conversion. If your application expects non-standard or special characters to be preserved across applications, special tests may need to be conducted to insure correct translation.

C. Record Length and Data Organization

Another limitation may arise from the size and organization of transfer files. Our version of the Gateway supports RJE stations, and we inherit the standard set of RJE restrictions. That is, for the most part, data is expected to be in the form of "card images", print lines, or console lines. Focus databases, VSAM clusters, variable length files, must be converted or unloaded before they are transferred. Also, the files must be of a certain length. Currently the Gateway supports punch and print files of up to 255 bytes. An in-house program is available to convert files with record lengths exceeding the 255 byte limit to 80 byte card images as part of the transfer process, up to 2048 characters.

D. Design Features and Implementation Assumptions

There are several assumptions made in the implementation of the Gateway you should keep in mind. First, currently you can only submit a job from the CSA3. Thus, if CSA3 is unavailable, it is not possible to send a file to the IBM from the VAX or from the VAX to the IBM. Jobs are placed into the queue, but no transfer takes place until CSA3 is ready. Secondly, the DecNet software constructs a filename for the transfer data from the IBM/MVS jobname. The file extension is built from the FORMS type (usually STD). This can be an advantage if the transfer

data set is a print file designed for special forms.

V. The Ibmjrje Login

Since the IBMRJE login has many directories and subdirectories, and because there are many jobs that are associated with the production and utility programs, some shortcuts and special commands have been developed to provide easy access to the whole system. Some of these abbreviated commands are developed as time goes on, so this cannot be a complete list, but it should provide any privileged IBMRJE user who is unfamiliar with the system to be able to get around fairly easily. If after logging on one wanted to see an overview of the directories issue the following command:

CSA3> SHOWMAP and see something like this:

```

                        [ibmrje]
                        |
      _____|_____
     |   |   |   |   |   |
    BIN COMFILES LOGFILES PRINT PUNCH READER
     |   |   |   |   |   |
     |   |   |   |   |   |
    _____|_____
   |             |
  PRINTCOM      PUNCHCOM

"home"          sets default [ibmrje]
"gobin"         sets default [ibmrje.bin]
"gocom"         sets default [ibmrje.comfiles]
"golog"         sets default [ibmrje.logfiles]
"gowprint"      sets default [ibmrje.print]
"gowpunch"      sets default [ibmrje.punch]
"gowreader"     sets default [ibmrje.reader]
"gowcomprint"   sets default
                [ibmrje.comfiles.printcom]
"gowcommpunch"  sets default
                [ibmrje.comfiles.punchcom]
```

to see this list run showmap
to see other abbreviated symbols run showsym
Csa3>

The above output gives an overview of the structure of the directories and shows the shortcuts to set default to them. If one wanted to quickly goto the subdirectory where jobs submitted to the IBM mainframe reside, the [ibmrje.reader] then issueing the command "gowreader" would do that.

The other commands mentioned include showsym, and the output from that command would look similar to this (due to the length of the list this is somewhat abbreviated):

```

$SET TERM/DEVICE=VT102
$mydir      :-      'f$logical("sys$disk")'f$directory()'
$home       :==      set def 'mydir'
$gobin      :-      set def [ibmrje.bin]
$gotst      :-      set def [ibmrje.bin.test]
$xeq        :-      $sy_syexe:xeqsystool
$gocom      :-      set def [ibmrje.comfiles]
$golog      :-      set def [ibmrje.logfiles]
$unblock    :-      "@[ibmrje.bin]unblock.com"
$clear      :-      "@[ibmrje.bin]clear.com"
$gore*ader  :==      set def [ibmrje.reader]
$gocompr*int :-      set def [ibmrje.comfiles.printcom]
$gocompu*nch :-      set def [ibmrje.comfiles.punchcom]
$ibmsub     :-      "@[ibmrje.comfiles]ibmsub.com"
$subprod    :-      "@[ibmrje.comfiles]rjeprod.com"
$subutil    :-      "@[ibmrje.comfiles]rjeutil.com"
$rjeprint   :-      "@[ibmrje.comfiles]rjeprint.com"
$rjepunch   :-      "@[ibmrje.comfiles]rjepunch.com"
$micro*fiche :-      "@[ibmrje.bin]microfiche.com"
$showqn     ==      "show queue *normal*"
$showqe     ==      "show queue *economy*"
$showqb     ==      "show queue sys$batch"
$showmap    :-      "@[ibmrje.bin]showmap.com"
$showsym    :-      "@[ibmrje.bin]showsym.com"
$listjob    :-      "@[ibmrje.bin]listjob.com"
Csa3¢

```

This shows a list of available commands from the login.com itself. Any one of these commands is a shortcut to accomplish what one would have to type on the right side of the equation. Additional shortcuts are added as necessary to the login.com file, but showsym will always output an updated list of the jobs as they are added.

A. Logfiles:

When jobs are submitted on the VAX, whether they are production jobs or simply jobs submitted during a login session, logfiles are created either in the home directory, that is in [IBMRJE], or perhaps in the [IBMRJE.LOGFILES] subdirectory. In either case, it is a good idea to clean up any unwanted logfiles that are past usefulness. Any one privileged to the IBMRJE login, and who uses it regularly, should maintain the habit of purging or deleting unwanted files. To purge files:

```

CSA3> PURGE FILE OR
CSA3> PURGE/KEEP=NUMBER FILE

```

To delete files:

```

CSA3> DELETE FILES OR DELETE/CONFIRM FILES

```

VI. References

The purpose of this section of the document is to identify the appropriate technical and procedural manuals for understanding and using the Gateway.

Data Processing Services

Standards and Procedures Manual

See especially Sections 2.5ff for configuration.

A. Memoranda

Russ Montello, "DecNet/SNA Gateway", 6/29/88.

This memo announces the product with suggested uses.

Edward Rosenthal, "Long Records via IBM-VAX Gateway", 10/1/88.

Describes how to transmit records exceeding 255 byte lengths.

Edward Rosenthal, "FICHE via IBM-VAX Gateway", 10/12/88.

Microfiche using the Gateway.

IBM Technical Publications

BarrHasp Manual

DEC Technical Publications

"Decnet SNA VMS Remote Job Entry User's and Operator's Guide",
Number AA-P589C-TE.

"Guide to Using DCL and Command Procedures on VAX/VMS",
Order Number AA-Y501A-TE.

"VAX/VMS DCL Dictionary", Order NO. AA-Z200A-TE.

DECnet/SNA Gateway-ST Problem Determination Guide, Order
Number: AA-MA17A-TK

B. Source Codes

RJEPROD.COM

```

$!*****
$! RJEPROD.COM
$!
$! SEARCH THE PUNCH DIRECTORY FOR PRODUCTION FILES
$! IF IT EXISTS THEN
$! DETERMINE IF ALL DATA IS ACROSS THE GATEWAY
$! IF SO THEN ACTIVATE THE APPROPRIATE FILE
$! ELSE
$! RESUBMIT THIS FILE
$! TO RUN VARIABLE 'HOURS' AND 'MINUTES' TIME
$!*****
$! MAINTENANCE LOB
$! added eff0950 EFF0975 EFF0976
$! V01 E ROSENTHAL 11/11/88
$!*****
$!
$! CLEANUP
$PURGE/KEEP=1 RJEPROD.LOG
$!
$on warning then goto warning_exit
$ SET DEFAULT [IBMRJE.PUNCH]
$! GET PROCESS ID FOR THIS CONTEXT
$CONTEXT = ""
$PID = F$PID(CONTEXT)
$SHO SYM PID
$!
$sho time
$!
$! THESE VARIABLES CONTROL THE TIME PASSED BEFORE RESUBMITTING
$HOURS = 1
$MINUTES = 00
$!
$! SEARCH FOR PRODUCTION FILES
$!
$!
$eff0950_label:
$FILE = F$SEARCH("[IBMRJE.PUNCH]EFF0950.wef")
$IF FILE .EQS. "" THEN GOTO eff0975_label
$!
$submit/name=effp0950 -
    /queue=csa4_normal -
    [ibmrje.comfiles]eff0950
$!

```

SOURCE CODES:
RJEPROD.COM

```

$eff0975_label:
$FILE = F$SEARCH("[IBMRJE.PUNCH]EFF0975.mef")
$IF FILE .EQS. "" THEN GOTO eff0976_label
$!
$submit/name = effp0975 -
    /queue = csa1_normal -
    [ibmrje.comfiles]eff0975
$!
$eff0976_label:
$! CHECK TO SEE IF BOTH THE YTD EFF0976 FILE
$! AND THE SYS3.TABLES(DIVRNGS) FILES ARE THERE...
$FILE = F$SEARCH("[IBMRJE.PUNCH]EFF0976.ytd")
$IF FILE .EQS. "" THEN GOTO normal_exit
$! else if that is there then check this
$FILE = F$SEARCH("[IBMRJE.PUNCH]EFF0976.div")
$IF FILE .EQS. "" THEN GOTO normal_exit
$!
$! if both files then do this
$submit/name = eff0976 -
    /queue = csa1_normal -
    [ibmrje.comfiles]eff0976
$!
$!
$! NORMAL AND WARNING EXIT
$! SUBMIT THIS JOB TO RUN IN CSA1_NORMAL
$!
$normal_exit:
$warning_exit:
$submit/after = " + "hours':"minutes" -
    /name = rjeprod -
    /queue = csa1_normal -
    [ibmrje.comfiles]rjeprod
$exit

```

EFF0950.COM

```

$!*****
$! PROC eff0950.COM    LAWRENCE BERKELEY LABORATORY
$! CALLED BY RJEPROD TO COPY WEEKLY EFFORT FILE TO
$! ADMINFILES
$! AND RUN FOCUS BATCH JOB
$!
$! NARRATIVE:
$!
$! DETERMINE IF ALL DATA IS ACROSS THE GATEWAY
$! IF IT IS
$!   COPY THE file to adminfiles
$!   RUN THE FOCUS BATCH JOB
$! ELSE
$! EXIT
$!*****
$! MAINTENANCE LOG
$! V01 E ROSENTHAL 10/27/88
$!
$!*****
$!
$! HOUSEKEEPING
$PURGE /KEEP=4 EFF0950.LOG
$!
$ SET DEFAULT [IBMRJE.PUNCH]
$!
$!
$! CHECK TO SEE IF IT ACROSS THE GATEWAY
$! 1) IF THE FILE IS ABLE TO BE OPENED FOR READ
$! 2) then it can be safely copied
$wef=f$search("eff0950.wef")
$!
$CHECK_LOCK:
$ON WARNING THEN GOTO POSSIBLE_LOCK
$OPEN/READ SCRATCH 'wef'
$CLOSE SCRATCH
$!
$! HAVING FINISHED PROVING THE DATA IS THERE,
$! copy it to appropriate spot
$!
$! obtain the current disk of adminfiles
$!
$disklogical admin_files
$!
$on warning then goto properexit

```

SOURCE CODES:
EFF0950.COM

```

$copy eff0950.wef -
    admin_files_never_quota:[adminfiles]wef.dat
$!
$! CLEANUP
$delete eff0950.wef;*
$!
$! send mail to distribution
$create eff0950.txt
THE WEEKLY EFFORT DATA FILE
HAS ARRIVED FROM THE DATA PROCESSING
SERVICES IBM MAINFRAME, AND
A VAX FOCUS UPDATE WILL SOON HAPPEN
$eod
$mail /edit=send eff0950.txt
@[ibmrje.bin]effort.dis
$!
$del eff0950.txt;*
$!
$! THIS EXEC SHOULD HAVE BEEN SUBMITTED
$! CSA1_NORMAL SO IT SHOULD BE SAFE to
$! EXECUTE A FOCUS BATCH JOB
$!
$define foc$dir1 admin_files_never_quota:[adminfiles]
$define foc$dir2 admin_files_never_quota:[adminfiles.jos]
$define foc$dir3 -
    admin_files_never_quota:[adminfiles.toolkit]
$define labwide admin_files_never_quota:[adminfiles]
$set def admin_files_never_quota:[adminfiles]
$focus
ex bwef echo = on
fin
$exit
$!
$properexit:
$!
$EXIT
$!
$POSSIBLE_LOCK:
$WRITE SYS$OUTPUT " POSSIBLE LOCK ON INFORMATION FILE"
$close scratch
$exit

```

*SOURCE CODES:
EFF0950.COM*

ACTION.COM

```

$!
$! ACTION.COM ACTIVATED BY RJEMASTER WHEN IT ENCOUNTERS AN
$!   INF FILE
$!
$! SEARCH THE INF FILE FOR VARIOUS FUNCTIONS
$!   (FICHE,PRINT,FOCUS ETC)
$! AND SUBMIT TO THE APPROPRIATE COM FILE
$! SENDING IT THE DATA REQUIRED (THE OUTPUT FROM SEARCH)
$!
$! EXAMPLE: SEARCH/OUTPUT=FICHE.DAT INF_FILE FICHE
$! then   SUBMIT rjefiche.COM/PARAMETER=FICHE.DAT
$! WAIT FOR EACH JOB TO FINISH BEFORE CONTINUING WITH THE
$! NEXT ONE
$! SINCE THE JOBS MIGHT PERFORM SOME ACTION ON A FILE
$! CAUSING CONTENTION WITH ANOTHER ACTION ON THE SAME FILE
$!
$! JOBS ARE SUBMITTED WITH THE PROCESS ID
$! THE SEVERITY .EQ. 3 IS THE INFORMATION BACK FROM THE VMS
$! OPERATING SYSTEM THAT NO STRINGS WERE FOUND IN THE
$!   SEARCH
$! THERE FORE NO JOBS OF THAT TYPE ARE SUBMITTED.
$!
$if p1 .eqs. "" then goto FINISHED
$set def [ibmrje.punch]
$context = ""
$pid = f$pid(context)
$show symbol pid
$inf_file=p1
$!
$SHO SYMBOL INF_FILE
$!
$FICHE ROUTINE:
$search/output='pid'.dat 'inf_file' fiche
$severity=f$integer($severity)
$if severity .eq. 3 then goto PRINT_ROUTINE
$SUBMIT/name=fiche/queue=csa2_normal -
    /log_file=[ibmrje.logfiles]fiche.log -
    [IBMRJE.BIN]rjefiche -
    /parameter=('pid'.dat,'inf_file')
$SYNC fiche/queue=csa2_normal
$print [ibmrje.logfiles]fiche.log
$!
$PRINT_ROUTINE:
$search/output='pid'.dat 'inf_file' print
$severity=f$integer($severity)

```

SOURCE CODES:
ACTION.COM

```

$if severity .eq. 3 then goto FOCUS_ROUTINE
$SUBMIT/name=printing/queue=csa2_normal -
    /log_file=[ibmrje.logfiles]rjeprint.log -
    [IBMRJE.BIN]RJEPRINT -
    /parameter='pid'.dat
$SYNC printing/queue=csa2_normal
$print [ibmrje.logfiles]rjeprint.log
$!
$FOCUS_ROUTINE:
$search/output='pid'.dat 'inf_file' focus
$severity=f$integer($severity)
$if severity .eq. 3 then goto UNBLOCK_ROUTINE
$SUBMIT/name=ACTION'pid'/queue=csa2_normal -
    /log_file=[ibmrje.logfiles]rjefocus.log -
    [IBMRJE.BIN]RJEFOCUS -
    /parameter='pid'.dat
$SYNC ACTION'PID'/queue=csa2_normal
$print [ibmrje.logfiles]rjefocus.log
$!
$UNBLOCK_ROUTINE:
$search/output='pid'.dat 'inf_file' unblock
$severity=f$integer($severity)
$if severity .eq. 3 then goto RENAME_ROUTINE
$SUBMIT/name=ACTION'pid'/queue=csa2_normal -
    /log_file=[ibmrje.logfiles]rjeunblock.log -
    [IBMRJE.BIN]RJEUNBLOCK -
    /parameter='pid'.dat
$SYNC ACTION'PID'/queue=csa2_normal
$print [ibmrje.logfiles]rjeUNBLOCK.log
$!
$RENAME_ROUTINE:
$search/output='pid'.dat 'inf_file' RENAME
$severity=f$integer($severity)
$if severity .eq. 3 then goto COMPARE_ROUTINE
$SUBMIT/name=RENAME/queue=csa2_normal -
    /log_file=[ibmrje.logfiles]RENAME.log -
    [IBMRJE.BIN]rjeRENAME -
    /parameter='pid'.dat
$SYNC RENAME/queue=csa2_normal
$!
$COMPARE_ROUTINE:
$search/output='pid'.dat 'inf_file' COMPARE
$severity=f$integer($severity)
$if severity .eq. 3 then goto FINISHED_ROUTINE
$SUBMIT/name=COMPARE/queue=csa2_normal -
    /log_file=[ibmrje.logfiles]COMPARE.log -
    [IBMRJE.BIN]rjeCOMPARE -
    /parameter='pid'.dat

```

**SOURCE CODES:
ACTION.COM**

```
$SYNC COMPARE/queue=csa2_normal
$!
$FINISHED_ROUTINE:
$FINISHED:
$!
$! DELETE THE MISCELLANEOUS DAT FILES CREATED
$! DURING THIS PROCESS
$on warning then goto exit_a
$exists = f$search("'"pid'.dat;*)
$if exists .eqs. "" then goto EXIT_A
$del 'pid'.dat;*
$EXIT_A:
$exit
```

DEBLOCK.C

```

/*****
DEBLOCK.C
SYNTAX: DEBLOCK INPUT-FILE OUTPUT-FILE RECORD-LENGTH
the input file came thru the gateway with blocked records
of eighty chars each record, but the vax truncates the end
blanks of each record. this program puts back the record
to its original form.
*****/
#include stdio
#include file
#include time
#define MAX_RECL_LENGTH 2048
#define MAX_BUFFER_SIZE 32000
#define NEWLINE '\n'
#define BLANK ' '
#define ERROR -1
unsigned int total_nl_written=0;
main(argc,argv)
int argc;
char *argv[];
{
    FILE *infile,*outfile,*logfile;
    int handle;
    int target;
    unsigned int numread=0;
    unsigned int total_chars_read=0;
    unsigned int total_chars_tmp=0;
    unsigned int total_chars_written=0;
    char buffer[MAX_BUFFER_SIZE];
    int dot,x;
    char logfile_array[20];
    char *bintim;
    struct tm *time_structure;
    int time_val,i;
    static char *weekday[7] = {"Sunday","Monday","Tuesday",
                                "Wednesday","Thursday","Friday",
                                "Saturday"};
    static char *month[12] = {"January","February","March",
                                "April","May","June","July",
                                "August","September","October",
                                "November","December"};
    static char *hour[2] = {"AM","PM"};

```

SOURCE CODES:
DEBLOCK.C


```

if (argc != 4) {
    greet();
    exit(0);
}
if ((infile = fopen(argv[1], "r")) == NULL) {
    printf("\ncould not open file %s ", argv[1]);
    exit(0);
}

if ((outfile = fopen(argv[2], "w", "rat=cr", "rfm=var"))
    == NULL) {
    printf("\ncould not open file %s ", argv[2]);
    exit(0);
}

target = atoi(argv[3]);
if (target > MAX_RECL_LENGTH)
{
    printf("\n max record length is %d ", MAX_RECL_LENGTH);
    exit(0);
}

/**** build a log file from outfile name ****/

    dot = strchr(argv[2], ".");
    for (x = 0; x <= dot; x++)
        logfile_array[x] = argv[2][x];
    logfile_array[x] = '\0';
    strcat(logfile_array, "log");

    if ((logfile = fopen(logfile_array, "w",
        "rat=cr", "rfm=var")) == NULL) {
        printf("\ncould not open file %s ", logfile_array);
        exit(0);
    }

printf("\n UNBLOCKING IN PROGRESS...");

while((numread = fread((char *)buffer, sizeof(char),
    MAX_BUFFER_SIZE, infile)) > 0)
{
    total_chars_read = total_chars_read + numread;
    printf("\n number of bytes read was %u ", numread);
    total_chars_tmp = doblock(buffer, numread, target, outfile);
    total_chars_written = total_chars_tmp + total_chars_written;
}

```

SOURCE CODES:
DEBLOCK.C

```

printf("\ntotal chars read was %u",
       total_chars_read);
printf("\ntotal chars written was %u",
       total_chars_written);
printf("\ntotal number of new lines written was
       %u",
       total_nl_written);

/* write out to logfile the result of totals */

fprintf(logfile, "\n%s %s",
        "results of deblocking", argv[1]);
fprintf(logfile, "\n%s %u",
        "total chars read was", total_chars_read);
fprintf(logfile, "\n%s: %u",
        "total number of new lines was",
        total_nl_written);
fprintf(logfile, "\n%s: %u",
        "total number chars written",
        total_chars_written);

time(&time_val);
time_structure = localtime(&time_val);

fprintf(logfile, "\nToday is %s, %s %d, 19%d",
        weekday[time_structure->tm_wday],
        month[time_structure->tm_mon],
        time_structure->tm_mday,
        time_structure->tm_year);

if (time_structure->tm_hour > 12)
{
    time_structure->tm_hour =
        (time_structure->tm_hour) - 12;
    i = 1;
}
else
    i = 0;

fprintf(logfile, "\n%s %d:%02d %s",
        "time now is ",
        time_structure->tm_hour,
        time_structure->tm_min,
        hour[i]);

printf("\n logfile created %s ", logfile_array);

```

SOURCE CODES:
DEBLOCK.C

```

fclose(infile);
fclose(outfile);
fclose(logfile);
exit();
}

doblock(buffer,numread,target,outfile)
char *buffer;
unsigned int numread;
int target;
FILE *outfile;
{

    static int index_tmp_buffer=0;
    static int num_chars_of_80=0;
    static int num_nl_read=0;
    static int c,position;
    char tmpbuffer[MAX_RECL_LENGTH];
    static int num_nl_per_block;
    static int lastnon;
    static int index=0;
    static int num_written;
    unsigned static int total_chars_written=0;
    static int pad_limit=80;
    static int pad_start;
    static int pad_index;
    static int counter=0;

    num_nl_per_block=figure(target);

    for (position=0;position<numread;position++)
    {
        c=buffer[position];
        if (c == NEWLINE)
        {
            num_nl_read++;
            if (num_nl_read == num_nl_per_block )
            {
                tmpbuffer[lastnon+1]=NEWLINE;
                tmpbuffer[lastnon+2]='\0';
                fprintf(outfile,"%s",tmpbuffer);
                counter++;
                total_nl_written++;
                if(! (counter % 100))
                    printf("\nwrote %u characters
                        ",total_chars_written);
                total_chars_written=total_chars_written+lastnon;
                num_nl_read=0;
            }
        }
    }
}

```

SOURCE CODES:
DEBLOCK.C

```

        index_tmp_buffer=0;
        num_chars_of_80=0;
        lastnon=0;
    }
    else if ((num_nl_read < num_nl_per_block)
        && (num_chars_of_80 <= 80))
    {
        for(pad_index=num_chars_of_80+1;
            pad_index <= pad_limit; pad_index++)
            tmpbuffer[index_tmp_buffer++] = BLANK;
            num_chars_of_80 = 0;
        }
    }
    else if (c != NEWLINE)
    {
        tmpbuffer[index_tmp_buffer] = c;
        num_chars_of_80++;
        if (c != BLANK)
            lastnon=index_tmp_buffer;
        index_tmp_buffer++;
    }
}
return(total_chars_written);
}
greet()
{
printf("\nSYNTAX: DEBLOCK INFILE OUTFILE RECORD-LENGTH-OF-
MAINFRAME-FILE ");
}

figure(num)
int num;
{
    int x=0;
    while ((x * 80) < num)
        x++;
    return(x);
}

```

