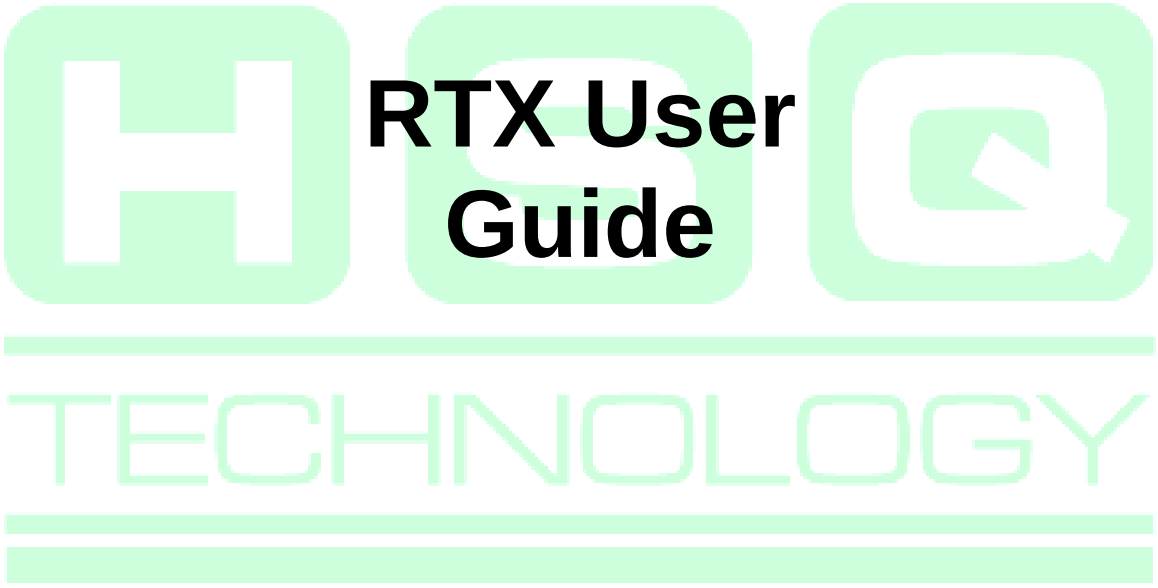
RTX User Guide



Introduction

We have had the ability to do file transfers to/from 25x86 RTUs via Kermit for a long time now. However, this method has at least two significant disadvantages:

- You need to shut down the RTU to do this
- You (typically) need to go to the RTU site to do this

To rectify this situation, RTU and host code has been upgraded to allow RTU-to-host file transfers to be done *while the RTU is running*.

This new feature allows us increased flexibility in handling RTU issues such as retrieving point maps, configuration info, RTU executable info, etc. Basically, any file on the RTU solid-state disk can now be transferred to or from the host via host commands without disrupting RTU operation. Furthermore, since HSQ technical staff members can dial into most of our customer's host machines from our desks, it allows us an easy access path to retrieve RTU data and files which previously would have required a site visit by either HSQ or customer personnel.

The user interface and basic functionality of this new RTU facility was inspired by (and patterned after) the TCP/IP FTP utility - thus users familiar with FTP will find no real surprises. Transfers of files both to and/or from the RTU are possible.

The transfer occurs within HSQ protocol – this means that all normal RTU functions such as VCL algorithms, control blocks, COS processing, alarm processing, etc still continue while the transfer is in process. Operating within HSQ protocol does mean that it can take a long time to transfer large files - each frame transfers only a relatively small piece of data so as to operate within the specifications of our protocol and NCC software.

Requirements

Requirements are as follows:

- RTU: software version X86V1_4R07g (known colloquially simply as X86V1_4R07g) and higher.
- Host: needs to have the RTX program.
- RTU and host must be in communication via either HSQ8 or HSQ16 protocols

Note that RTX works in *both* HSQ8 and HSQ16 protocols.

RTX does *not* work for any of the following:

- 2504-based RTUs
- Slash-86 RTUs
- 25X86-based RTUs running V1_3 software
- 25X86 RTUs equipped with software *prior to* X86V1_4R07g
- Communications protocols other than HSQ8 or HSQ16

Technical Details for Basic RTX Commands

The RTX program has a very simple user interface patterned after the FTP program.

The basic commands are:

- HELP - displays a short summary of commands
- OPEN - opens an RTU in preparation for file transfers
- BINARY - specify that file operations are BINARY
- ASCII - specify that file operations are ASCII
- GET - retrieves a file from the RTU, placing it on the host
- PUT - sends a file to the RTU, reading it from the host
- CLOSE - indicates you are done working with this RTU
- QUIT - exits the program

The RTX program has been subject to a near-constant stream of improvements since it was initially created. All of the above commands *with the exception of the ASCII command* are available in all versions of RTX. The very earliest versions of RTX did not support the ASCII command – if you have such a version, you may want to contact HSQ to request that the RTX program be upgraded to a more current version. How can you tell what commands are supported by your version of RTX? Type HELP at the RTX prompt for more details on commands supported by *your* installed version of RTX.

Note that later versions of the RTX command support an array of new commands – these are documented in the section below entitled “Technical Details for Advanced RTX Commands”.

As a simple example, a sample program run illustrating how to get the file RTUFATAL.RTU (the file which contains a report on the RTUs last encountered fatal error) from the RTU and onto the host is shown below. Note how simple it is - it uses only the following four basic commands: OPEN, ASCII, GET, QUIT.

```
UNJMVA$ run mnet$exe:rtx
```

```
RTX - HSQ RTU file transfer program

RTX> open 99
RTU-UNJMVA::1:99    opened
RTX> ascii
Transfer mode set to ASCII
RTX> get rtufatal.rtu
Opening file RTUFATAL.RTU in ASCII mode

Reading file, offset=          0
Reading file, offset=         10
…
Reading file, offset=        860
Reading file, offset=        870
Closing file
RTX> quit
UNJMVA$
```

Restrictions on the PUT command

It should be clear that *sending* a new file to the RTU via the RTX "PUT" command while the RTU is in operation can have disastrous consequences. Furthermore, the consequences might not be immediately visible, as the problem may not come to light until the next RTU reboot. At the risk of stating the obvious, it is recommended you *not* use the RTX “PUT” command unless you know exactly what you are doing.

Restrictions and Known Bugs

I know it looks like a long list, but I didn't want to leave anything out.

- Some early versions of RTX had a bug in the logic to detect RTU up/down status – in these cases, you might get the following message:

  Warning: RTU is currently not up

  In such cases, this warning can safely be ignored. Other than printing an incorrect warning, this bug does not affect program operation in any other way.
- RTX file transfers on RTUs with switchable dual comm media may not succeed if the comm line is switched during the transfer – BLA for example.

- Just like FTP, RTX requires that the file name be the same on both platforms. There is no facility to rename files as you copy them. Please make sure your file name meets all requirements for both the DOS and VMS file systems. If you are not entirely sure about the relevant file name restrictions for both platforms, ask someone who is. Do not include any path names (either DOS or VMS flavors) in your file specifications. On the RTU end, files go to the root directory. On the VMS end, files go to your default directory.
- If you try to GET a file which does not exist on the RTU (commonly the case w/ regard to typing errors) you get the error message

  MNET-E-NOTIMPLMNTD, Feature not implemented

  which is not terribly indicative of the real problem! Other than some confusion due to an unclear error message, no damage is done.
- There is a bug in rtu sw versions *prior to* R07r that will cause any file transfers which do not complete within 5 minutes to fail. In an emergency, it is possible to work around this restriction, but it is not simple. Versions R07r and better do not have this problem.
- It is recommended you DO NOT terminate the RTX program early via control-C or some other such mechanism. Just in case you do, version R07h and later handle this more gracefully than prior versions.
- Some RTU sw versions do not handle RTX “PUT” commands correctly if the file ends with an extension of “.TMP” or “.OLD”
- RTX is susceptible to problems on communications media where the NCC receives an echo of its own data transmission. This is not really an RTX issue, it is instead a sign of a poorly configured NCC and comm circuit, but because RTX generates long data packets it is more likely to run into problems in such situations.
- While running RTX, it is recommended that you refrain from running the RDR command on the same RTU (running RDR on a different RTU is ok). Also, it is recommended you refrain from running a *second copy* of RTX program on the same RTU (running a second copy of RTX on a different RTU is ok). While the RTU itself has no trouble with these cases, the host HSQ NCC programs generally do not handle this situation gracefully. Typically you will get the error message

  Point already has a command outstanding

  from the NCC, and one of the two programs will fail. Some later versions of RTX and RDR do have logic to help minimize these problems, but even

in these later versions there is no absolute guarantee that you won't run into trouble.

File Formats and Kermit and FTP

All versions of RTX are capable of doing file transfers in BINARY mode. Later versions of RTX can also do ASCII-mode transfers.

Having an early binary-mode-only version of RTX does *not* prohibit one from transferring text files - it only means that the VMS file system will have file attributes for a
binary file (attributes will show fixed record size of 512 bytes) and not a normal VMS text file (variable record size).

Once on VMS, files which were text files on the RTU can be viewed using the VMS "TYPE" command. However, because they are stored on VMS as binary files, they will appear "odd" if edited with a VMS text editor. All of the RTU configuration files are text files, and using the VMS "TYPE" command to view these files is often quite useful - in general, this rule applies to all RTU files with the extension ".RTU".

After transferring to the host, files can be further transferred in binary mode from VMS to Intel platforms (DOS or Windows) via the standard FTP or Kermit programs. In both cases, it is important to do the transfer in binary mode - use the FTP "BIN" command, or the Kermit command "SET FILE TYPE BIN" for the transfer. Once transported back to Intel platforms, files will have the identical file characteristics as they had on the RTU.

So generally speaking, do all transfers in BINARY mode and you will be sure to preserve file integrity.

An Example using the GET Command

The RTU pointmap is stored in the rtu file system as the file POINTMAP.RTU - you can see this file when you read the RTU directory as illustrated below.

```
UNJMVA$ rdr 99

 RDR - HSQ RTU Directory Reader

  1) 25X86DIA.EXE              04-Nov-2004 15:07:26.000      14606
  2) AUTOEXEC.BAT                30-Oct-2080 14:17:30.000        231
```

```
…
 49) POINTMAP.RTU              09-Jun-2006 16:41:02.000       10207
…
 69) WATKER.EXE                30-Nov-2000 16:45:50.000       10560
 70) WATTCP.CFG                04-May-2080 01:57:42.000         312
```

You can download POINTMAP.RTU to one of the local MISER machines (can be a host or a workstation) as shown below:

```
UNJMVA$ run mnet$exe:rtx

RTX - HSQ RTU file transfer program

RTX> open 99
RTU-UNJMVA::1:99    opened
RTX> get pointmap.rtu
Opening file POINTMAP.RTU in BINARY mode

Reading file, offset=          0
Reading file, offset=          8
…
Reading file, offset=      10208
Closing file
RTX> quit
```

The example above will download the file to the local MISER machine in binary form. If the ASCII command had been done before the transfer, then the file would have been downloaded to the MISER machine in ASCII format – this can be useful if you intend to read or modify the file on a VMS system.

Note that since RTX operates within HSQ protocol, it only downloads a small amount of information per poll/response cycle. As a result, this process will take a while on slower communications lines. Upon completion, the file will be in your local directory – see below.

```
UNJMVA$

UNJMVA$ dir pointmap.rtu/siz

Directory SYS$SYSDEVICE:[USERS.SYSTEM]

POINTMAP.RTU;1          20

Total of 1 file, 20 blocks.
UNJMVA$
```

An Example using the PUT Command

You can use the RTX “PUT” command to transfer any file from a MISER machine to the RTU. Since the file NEWRTU.EXE is the principal file containing the RTU firmware, it is also the most frequently used with the “PUT” command.

To upgrade NEWRTU.EXE on an RTU, you must first get a copy of NEWRTU.EXE onto one of the customer’s MISER machines – this can be done using C-Kermit, FTP, SSH, or any other mechanism that can transport binary files.

Then use RTX to transfer the file NEWRTU.EXE onto the RTU – see example below.

```
UNJMVA$ run mnet$exe:rtx

RTX - HSQ RTU file transfer program

RTX> open 99
RTU-UNJMVA::1:99    opened
RTX> put newrtu.exe
Opening file NEWRTU.EXE in BINARY mode

Writing file, offset=         0
Writing file, offset=        85
Writing file, offset=       170
Writing file, offset=       255
…
Writing file, offset=    209920
Writing file, offset=    210005
Writing file, offset=    210090
Writing file, offset=    210175
Closing file
RTX> quit
UNJMVA$
```

Since the file NEWRTU.EXE is typically about 200K in size, this transfer will take a while. Very rough estimates for some useful cases are:

Ethernet	7 minutes
9600 baud line dedicated to a single RTU	20 minutes
1200 baud line dedicated to a single RTU	60 minutes

Note that there are many factors which can contribute to slowing this process down to be way slower than the above numbers – these include:

- Sharing the comm line among multiple RTUs
- Comm errors
- NCC polling parameters
- Front porch/back porch settings
- Baud rates

Technical Details for Advanced RTX Commands

Later versions of the RTX program added quite a few new commands. These are recommended for advanced users of the program. These commands are briefly documented below.

- TYPE file - type a file from rtu onto screen
- DELETE file - delete a file on rtu
- MGET filespec - get multiple files from rtu
- MPUT filespec - future
- MTYPE filespec - type multiple files from rtu onto screen
- MDELETE filesp - delete multiple files on rtu
- RENAME old new - rename a file on rtu
- STATUS - show current ascii/binary mode
- GETLEN n - data len for GET operations
- PUTLEN n - data len for PUT operations
- MNET$FIELD i - ignore unit status
- DELAY n - delay for n seconds
- REBOOT cmt req - Reboot the rtu, log comment, startup pgm request
- DIR filespec - directory matching filespec (full)
- LS filespec - directory matching filespec (quick)
- ! - spawn a cmd line (then type LOGOUT to return to RTX)

Not all versions of RTX support all of the above commands – type HELP at the RTX prompt for more details on commands supported by *your* installed version of RTX.

Using the RTX REBOOT Command

Users often reboot their RTUs via the host INR command. Sadly, the INR command has the undesirable side effect of deleting all point definitions stored in

the RTU. So if a simple reboot is desired, the process becomes needlessly slow if using INR.

Later versions if the RTU and RTX software added the required data protocol and commands to do a simple reboot – one that does not delete the stored point definitions. An RTU rebooted this way will come back into operation faster than one initialized with the INR command.

To use the REBOOT command, you need to have both host RTX sw and RTU sw that supports this feature.

- The RTU must be running R12 sw or better
- The RTX program needs to support this command – type HELP at the RTX prompt to see if it is in the list of commands

The REBOOT command allows you to also include a comment as part of the command – this will be recorded in the RTU fatal error report providing a way to track intentional reboots. Unless you have reason to do otherwise, it is recommended you type your name as the comment, so if anyone later needs to know who initiated the reboot process it is easy to find them. The comment must be all one word without blanks, so multi-word comments need to have a dash or underscore to connect the words. See the example below.

```
RTX> open 5:1
RTU-UNJMVA::5:1     opened
RTX> reboot John_Doe
Requesting a reboot...
RTU successfully acknowleged reboot command
RTX>
```

Making RTX Run Faster

People often ask if there are ways to make RTX run faster – often the answer is, it depends! RTX operates at the speed that it does because it makes many round-trip exchanges with the RTU, transmitting only a modest amount of data on each round trip. Basically, the most effective way to speed up RTX is to increase the amount of data transmitted for each of those round trips. There are a few requirements that need to be met for this to work properly:

- The RTU sw must be able to handle the longer data packet
- The host NCC sw must be able to handle the longer data packet

- The communications line must be able to deliver the longer packet in an error-free manner with a reasonably high probability of success

Unless these requirements are all met, you are unlikely to get a good result. More details below…

Making RTX Send Files Faster

By default, the RTX program will send files to the RTU by transmitting 85 bytes of useful data per round trip. This is a good number because it is supported by all RTU sw that is RTX capable, and by all host NCC programs ever delivered.

If you are sending a file to an RTU running R11q or better, you can use the RTX PUTLEN command to increase the amount of useful data per round trip. Since RTX handles data in 512-byte bundles, 103 is suggested as a number that yields a good benefit – this will transmit the 512 byte bundle in 5 round trips.

As of this writing, the absolute useful ceiling for the PUTLEN command is 109 bytes in HSQ8 protocol, and 112 in HSQ16. However, because of the above RTX data bundling, values above 103 do not yield any further improvement in speed and therefore are not recommended.

Making RTX Receive Files Faster

By default, the RTX program will receive files to the RTU by receiving 10 bytes of useful data per round trip. This is a good number because it is supported by all RTU sw that is RTX capable, and by all host NCC programs ever delivered.

If your host system has an NCC capable of receiving longer inbound RTX packets, you can use the RTX GETLEN command to increase the amount of useful data per round trip – 15 is suggested as a number that yields a good benefit. If your NCC does not support this longer length, your RTX GET commands will simply fail by timing out. Other than displaying an error message no damage is done, so if you are not sure you can simply try it and see if it works.

RTU Version Issues

As with all our RTU sw, the RTX portion has evolved over time. Here are some important milestones:

- X86V1_4R07g: First RTX-capable released version – this version has no safeguards to handle interrupted file transfers and is therefore highly sensitive to incomplete transfers
- X86V1_4R07h: Added safeguards to handle interrupted file transfers by closing files after 5 minutes of inactivity.
- X86V1_4R07r: Fixed bug there file transfers that took longer than 5 minutes of elapsed time would fail.
- X86V1_4R07u: Added logic to NOT overwrite the target file until the transfer was successfully completed.
- X86V1_4R11: changed file transfer inactivity timer from 5 minutes to 2 minutes.
- X86V1_4R11n: Added support for RTX “DELETE” and “RENAME” commands, and also saves previous version of file as “.OLD”.
- X86V1_4R11q: Can handle larger RTX “PUTLEN” values for faster PUT operations (109 bytes in HSQ8 protocol, 112 in HSQ16)
- X86V1_5R12: Added support for RTX “REBOOT” command