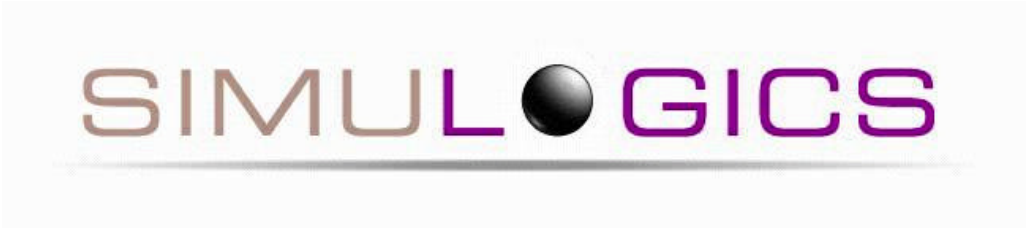
The

reNOVAte

User's Manual



P. O. Box 3581
Boulder, Colorado 80307-3581 U.S.A.

www.SimuLogics.com

Notice

SimuLogics, Inc. (SLI) has prepared this manual for use by its personnel, licensees and customers. The information existing herein contains valuable properties and trade secrets of SimuLogics embodying substantial creative efforts and confidential information, ideas and expressions, and shall not be reproduced in whole or in any part, in any form whatsoever, including electronic or magnetic, without SimuLogic's express prior written permission.

SLI reserves the right to make changes in specifications and other information contained in the document without prior notice.

Use of this product is governed by a SLI Software License Agreement, and use of this product indicates acceptance of the terms of this agreement. The text of this agreement is reprinted in the first section of this manual. The terms and conditions set forth in this agreement constitute the sole agreement between SLI, its OEMs, VARs and/or the user.

No representation, information or affirmation of fact contained in this document, including but not limited to statements concerning capacity, capability, performance, suitability for or performance of products described herein shall be deemed to be a warranty by SLI for any purpose, or give rise to any liability of SLI whatsoever.

internal version - June, 1994
second version - May, 1998
third version – October, 1998
fourth version – February, 1999
fifth version – February, 2000

part number 200-404d

phone: (303) 466-7717
fax: (303) 465-5780

www.SimuLogics.com

Re*NOVA*te License Agreement

The ***Re*NOVA*te*** Program is licensed by SimuLogics, Inc. to customers for their use only on the terms set forth below. Use of the Program indicates your full acceptance of these terms.

1) LICENSE. SimuLogics, Inc. hereby agrees to grant you a non-exclusive license to use the ***Re*NOVA*te*** program (Program) subject to the following terms and restrictions. The Program and its documentation are copyrighted and may not be copied or reproduced in any part, in any form, for any purpose, by any method, except according to the terms stated in this Agreement.

You may:

a. use the Program on a single machine at one time, or one network at a time if you have purchased an appropriate Program network license ;

b. copy the Program into any machine readable or printed form only for backup purposes in support of your use of the Program on the single machine ;

c. transfer the Program from one computer to another, provided that the Program is executed on only one computer at a time ;

Except as expressly licensed herein, you may not use, copy, disseminate, distribute, modify, translate, reverse engineer, decompile, disassemble, and/or create derivative works from the Program, sell, lease, sublicense, rent, give, lend, or in any way transfer, by any means or in any medium, the Program.

If you later receive an Update to the Program or if the Program is an Update to a prior version, any transfer must include both the Update and all accessible prior versions of the Program.

2) TERM. The license is effective until terminated. You may terminate it at any other time by destroying the Program together with all copies, modifications and merged portions in any form, and sending any corresponding software protection device back to SimuLogics. It will also terminate if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy the Program and documentation together with all copies, modifications and merged portions thereof in any form, or return shipping prepaid the Program and documentation together with all copies, modifications and merged portions thereof in any form to SimuLogics.

No refund, in full or part, shall be due or made upon termination by Licensee.

3) PROHIBITION AGAINST REVERSE ENGINEERING. You agree not to reverse engineer the Program, or any portion of it, or otherwise attempt to determine the underlying source code of the Program, or permit any such actions.

4) LIMITED WARRANTY. The Program is provided "as is" without warranty of any kind, either expressed or implied, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the Program is with you. Should the Program prove defective, you and not SimuLogics assume the entire cost of all necessary servicing, repair or correction.

Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you. This warranty gives you specific legal rights and you may also have other rights which vary from state to state.

SimuLogics does not warrant that the operation of the Program will be uninterrupted or error free.

However, SimuLogics warrants the media on which the Program is furnished, and any protection device(s), to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your shipping receipt.

5) LIMITATION OF LIABILITY. In no event will SimuLogics or anyone else who has been involved in the creation, production, maintenance or delivery of the Program be liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damages arising out of the use or inability to use this Program even if SimuLogics has been advised of the possibility of such damages, or for any claim by any party.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

6) GENERAL. You may not sublicense, assign or transfer the license or the Program except as expressly provided in this Agreement. Any attempt otherwise to sublicense, assign or transfer any of the rights, duties or obligations hereunder is void.

Any updates to the Program shall also be governed by the terms of this License Agreement.

7) ACCEPTANCE. Your use of this Program acknowledges that you have read this Agreement and agree to its terms. You further agree that this Agreement is the complete and exclusive statement of the agreement between us which supersedes any other communications between us relating to the subject matter of this agreement.

8) U.S. GOVERNMENT RESTRICTED RIGHTS The software and documentation are provided with restricted rights. Use, duplication, or disclosure by the Government is subject to restrictions set forth in the applicable commercial computer software clause(s) of The Rights in Technical Data and Computer Software, DFARS section 252.227-7013. Contractor/manufacturer is SimuLogics, Inc./P. O. Box 3581/Boulder, CO 80307-3581

9) EXPORT RESTRICTIONS. The Program may be subject to the export controls of the United States Departments of State and Commerce and you agree to fully comply with all applicable U.S. export regulations governing export, destination, ultimate end user and other restrictions relating to the Program.

This Agreement will be governed by the laws of the State of Colorado, United States of America.

Should you have any questions concerning this Agreement you may contact SimuLogics by writing to SimuLogics, Inc., Sales and Service, P.O. Box 3581, Boulder, Colorado 80307-3581.

Table Of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Chapter 1

Re*NOVA*te Overview

What is ReNOVAte?

Re*NOVA*te is SimuLogics's software product that directly runs Data General **Nova, Eclipse, Desktop Generation**, **Point 4**, Strobe Data **Swift**, **Falcon** and **Hawk**, **Bytronix** and other **Nova-compatible**[1] computer software on any popular open system platform.

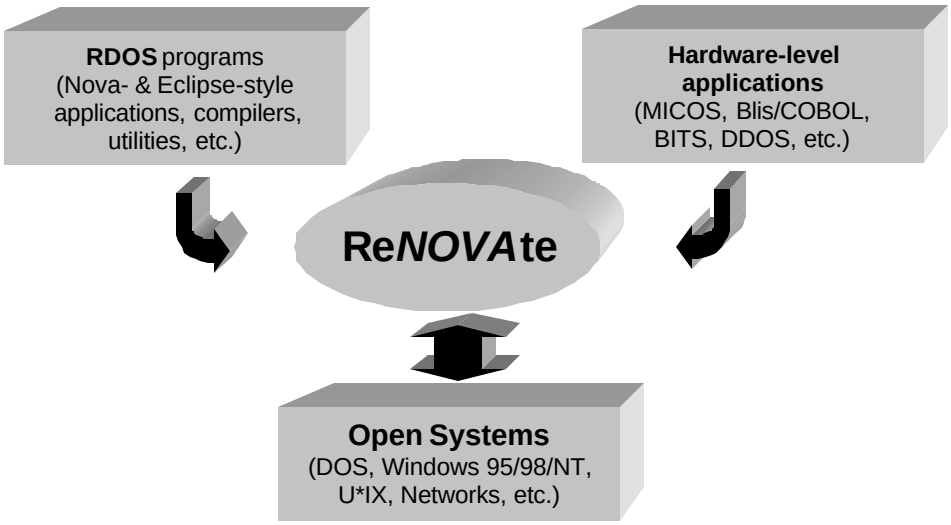
Now any **RDOS**, **BITS**, **MICOS**, **BLIS/COBOL**, **DDOS, IRIS, VMOS, RMS, IOS, EOS**, **Infomark/Summit** and other Nova or Eclipse operating system and user application can directly run unmodified on any popular computer and operating system, including MS-DOS, PC networks, Windows 95/98, Windows NT, UNIX, DG/UX, HP/UX, AIX, Sun/OS, Solaris, UnixWare, Linux, VAX/VMS and others.

Re*NOVA*te is a **totally *software-based*** solution, so it eliminates the need for additional hardware, coprocessor boards, strings, smoke or mirrors to run standard Nova- or Eclipse-style programs on open systems.

Re*NOVA*te provides two main powerful environments:

1. a complete, low-level **hardware** emulation environment, and
2. A full-featured, speed-optimized **RDOS** environment

[1] Other **Nova**-compatible systems include **Integrated Digital Products (IDP)**, **Information Now (INI)**, **SCP**, **Digidyne**, **Fairchild, Kernonix, DCC, Whetstone II/III, Star Technologies, Ardent, Ampex** and other third-party hardware.



Run Nova & Eclipse Programs on Open Systems

RDOS Environment

In **RDOS mode**, **Re*NOVA*te** creates a full-featured Data General Nova- or Eclipse-RDOS environment on any open system, from laptop to mainframe. The actual program and data files from standard RDOS-based machines are directly used on the new open systems *without any translation or conversion*; thereby insuring complete compatibility with Data General computer based applications. That is, the actual program save files (-.SV) from the DG-compatible systems are run on the target open systems; and user application data files are likewise used. And native execution of standard RDOS systems calls gives legacy programs the full benefit of open system resources and performance.

This RDOS environment exists seamlessly with the native platform, so that all features and capabilities of the underlying system can be used from within an RDOS framework. For example, the standard RDOS .EXEC (execute a program) system call can be used to run either an RDOS program or a native operating system or user program. This gives users full control of RDOS and the native operating system from the CLI.

Conversely, native operating system programs have full access to all RDOS files, giving tremendous flexibility and convenience in development situations. No "container files" are used which only hide RDOS data and limit system usefulness.

Hardware Environment

In full **hardware emulation mode**, **Re*NOVA*te** reproduces virtually all hardware aspects of the Nova, Eclipse, Desktop Generation, Point 4, Bytronix, Digidyne, etc.

architecture. Even proprietary systems - such as MICOS, BLIS/COBOL, DDOS, IRIS, BITS, etc. – can run on virtually any open system *unmodified* using **Re*NOVA*te**.

Nova and Eclipse "devices" may be associated with any file or device supported by the underlying operating system, giving the capability to handle virtually any "special" requirement.

All versions of **Re*NOVA*te** dramatically extend legacy software investment, create new dimensions of portability, and open new doors of opportunity and possibility. **Re*NOVA*te**'s faithfulness to detail lets legacy software run on systems never before possible, while blazing a path to the future.

CPU Support

Re*NOVA*te supports the standard Nova, Nova 3, Nova 4, Fairchild 9445 and Eclipse hardware instruction sets. This provides unmapped Nova, Mapped Nova 4, Fairchild 9445 and Mapped Eclipse RDOS operating system environments.

Also, full emulation for common Data General peripherals is supported so that even non-RDOS environments or software, such as MICOS, DDOS, BITS, IRIS, DDOS and BLIS/COBOL, can run on open systems without modifications. The low-level hardware emulation capabilities are so complete that even DG processor and peripheral diagnostics run using **Re*NOVA*te**.

RDOS Support

All standard RDOS development software may be run, such as MAC, ASM, BBASIC, XBASIC, ICOBOL, FORTRAN IV, FORTRAN 5, DG/L, ALGOL, etc. **Re*NOVA*te** now turns any open system as a powerful software development platform.

User RDOS-based applications can run unmodified on any open system platform, preserving previous software investment while supporting future technology.

In addition, standard RDOS system calls are all supported, including RDOS overlays, system-independent read/write line handling, etc. Advanced RDOS features such as memory remapping, user clock interrupts and multi-tasking are even supported.

These features, plus many more, make your past software investment work for your future.

Low-level Hardware Support

Data General Nova and Eclipse programs that do not use RDOS can also be supported by **Re*NOVA*te**. Complete low-level hardware emulation exists, including interrupt and

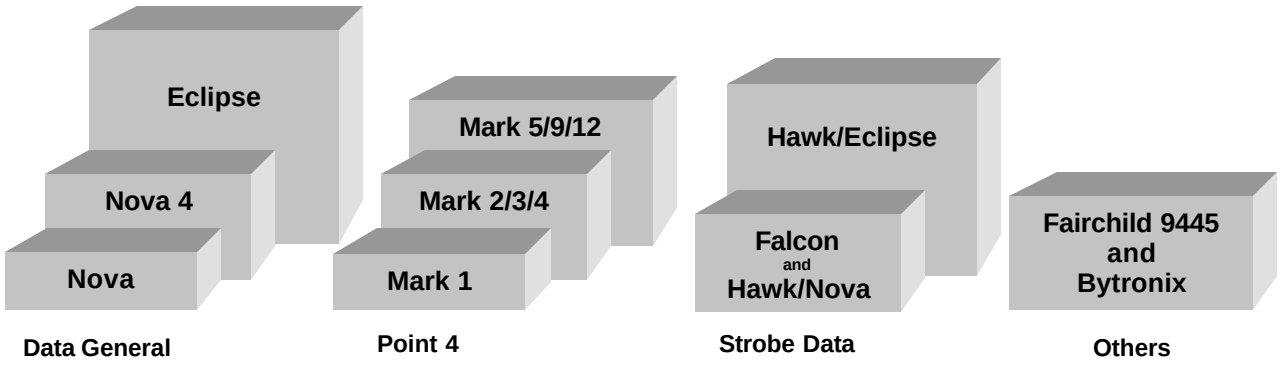
device handling. Data General hardware diagnostics even run at this level, reflecting the depth of compatibility provided.

A full complement of hardware support is provided by different versions of **Re*NOVA*te**, including CPUs, main CPU options and peripheral devices. All of these options easily integrate into any target open system hardware and software.

CPU types

Seven main CPU types are supported:

- Standard Nova with hardware multiply/divide and FPU
- Nova 3 and Nova 4 with hardware signed and unsigned multiply/divide, stack and byte instructions, MMU and Floating Point Unit
- Standard Eclipse with Floating Point Unit and Character Instruction Set
- Fairchild 9445 features with all Nova 4 capabilities and special 64 K word support
- Point 4 Nova-style
- Strobe Data Nova-style
- Strobe Data Eclipse-style



Major CPU types supported

Peripheral Support

Common CPU options and peripherals supported include:

- Unsigned multiply/divide (MUL/DIV)
- Signed multiply/divide (SMUL/SDIV)
- Floating Point Unit support (FPU)
- Eclipse Character Instruction Set extensions (CIS)

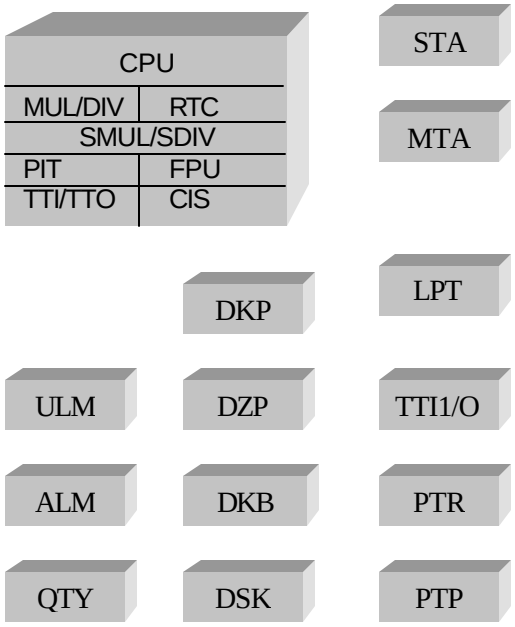
- Real Time Clock (RTC)
- Programmed Interval Timer (PIT)
- Console input/output (TTI/TTO)

Additional peripheral devices are supported, including:

- Paper tape reader (PTR)
- Paper tape punch (PTP)
- Line printers (LPT)
- Second console (TTI1/TTO1)
- Magnetic tape drive (MTA)
- Streaming tape drive (STA)
- Fixed Head Disk (DSK)
- Swapping Disk (DKB)
- "traditional" Moving Head Disk (DKP)
- "new" Moving Head disk (DZP)
- Serial port multiplexor (QTY, ALM and ULM)

All of these supported DG devices may be directed to the devices and files of the underlying operating system.

Additional device drivers may be created on a custom basis to meet the most demanding application requirement.



Major peripherals supported

Product Benefits

- Liberates legacy Nova, Eclipse, Point 4 and Hawk software from hardware-based solutions.
- Preserves, and enhances software investment.
- Can provide a complete solution, or can be a part of a larger migration process.
- Saves program rewriting and migration time.
- Provides vendor independence.
- Provides hardware independence.
- Provides operating system independence.
- No special hardware or other software required.
- Seamlessly integrates Nova and Eclipse files into open system file systems.
- Insures complete program and data file compatibility between DG and any other computer and operating system.
- Runs dedicated programs not executable using any other means.

Features

- Directly runs RDOS programs without modification on any popular open system platform.
- Runs Nova, Eclipse, Bytronix, Strobe Data Hawk and Falcon, SCP, IDP, Digidyne, Fairchild 9445, Point 4, IDP, INI, SCP, Digidyne, Fairchild, Kernonix, DCC, Whetstone II/III, Star, Ardent, and Ampex programs on open systems without modification.
- Uses current RDOS program save files and data files totally unmodified.
- Supports RDOS rev 5, 6 and 7 programs and data files.
- Adds many new powerful extensions, such as calling native programs from the CLI or user programs, and more.
- No program conversion or data translation needed.
- Supports virtually all technical features of the RDOS environment.
- Allows continued RDOS development on any open system.
- Provides seamless access to native operating system capabilities while using RDOS CLI.
- Creates a powerful RDOS development environment that works transparently with open system development environments.

- Runs many dedicated, non-standard or “special” programs such as MICOS, DDOS, BLIS/COBOL, BITS, IRIS and others.
- Supports many Strobe Data Hawk-specific extensions.

RDOS Features

- Directly runs Nova or Eclipse RDOS program and data files.
- Emulates the Nova, Nova 4 and Eclipse processors with no additional hardware or software.
- Supports standard RDOS rev 5, 6 and 7 features and system calls.
- RDOS programs can directly call open system programs (i.e. spreadsheets, word processors, editors, etc.).
- Native programs can call RDOS-based programs.
- Powerful DG to native directory and file name translation options.
- RDOS device name to native operating system logical name translations.
- Search list facilities simplify program file, data file and general file access.
- All open system file, device and network capabilities are available to RDOS programs.
- Transparent open system program access to any RDOS file and directory - no “container” files are used.
- Support for most multi-tasking environments.
- User clock support.
- Extended memory remap support.
- Extended .RDB/.WRB I/O support.
- Special support for QTY line access - QTY I/O may be redirected to native serial devices.
- RDOS system call statistics maintained.
- Multi-tasking console support.
- Inter-ground common area support.
- Powerful system extensions.
- Expanded virtual console.
- Makes ANSI terminals behave like native DG CRTs.
- Dynamic trace facility tracks program system call execution and quickly resolves problems.
- RDOS system call execution statistics maintained for performance monitoring.

Hardware Emulation Features

- Seven main processor types are supported:
  - Standard Nova
  - Nova 3 and Nova 4
  - Standard Eclipse
  - Fairchild 9445 with 64 K words of memory
  - Point 4 Nova-style
  - Strobe Data Nova-style
  - Strobe Data Eclipse-style
  - Bytronix Nova compatible
- 32 K words of memory space for standard systems (special 64Kword based systems also supported on special request).
- Up to 2 MB total main memory for Data General Memory Map Unit (MMU) - supported systems.
- Common processor options supported, including:
  - Unsigned multiply/divide (MUL/DIV
  - Signed multiply/divide (SMUL/SDIV)
  - Floating Point Unit (FPU) [with bit-level compatibility with all DG formats and quirks!]
  - Eclipse Character Instruction Set (CIS)
  - Real Time Clock (RTC)
  - Programmed Interval Timer (PIT)
  - Main console TTI/TTO
- Common DG peripherals supported, including TTI1/TTO1, MTA, SMD, LPT[1], PTR[1], PTP[1], DSK and DKP.
- Common Point 4 peripherals supported, including console mux, ST506 disk controller, Lotus-style disk controllers, custom controllers.
- Peripheral device I/O can be directed to use open system files and devices.
- Extended version of Virtual Console for complete system control.
- Seamlessly integrates into open systems.
- Emulates down to the register and interrupt level if so desired.
- Runs processor and peripheral diagnostics, insuring maximum compliance and reliability.
- Instruction execution statistics maintained for performance monitoring.
- DG-like “virtual console” gives user complete control over low-level hardware or RDOS environment, including:
- Full register and memory display and modification
- Breakpoint capabilities
- Control over I/O devices and actions

- Convenient symbolic assembly and disassembly
- Instruction execution tracing
- Execution statistics and performance monitoring
- Instruction execution history
- I/O instructions can dynamic link to native user 'C' routines.

Installing, Running and Using Re*NOVA*te

No special hardware or software is required to use **Re*NOVA*te** on any platform supported. Therefore the following guidelines apply to all systems, from PC to mainframe:

Files and transfers

Program and data files should be transferred from the Data General[-compatible] system to the new target system in binary mode – the new system should exactly reflect the contents of the DG system.

This can be performed using serial port communications, network transfers or magnetic media exchange.

Special features exist to directly use IRIS and BITs DISCUTIL backup tapes with **Re*NOVA*te** without conversion or modification(!).

SimuLogics can provide additional media conversion and transfer services using its own in-house facilities, or additional consulting services.

Disk space

The amount of disk space required depends upon the applications being run; **Re*NOVA*te** uses less than 1 MB maximum for system installation. Most PCs now have more disk space than the largest legacy DG systems used to, so even the most sophisticated application can typically run without concern on an entry-level PC.

Main memory

Re*NOVA*te exists as a full 32-bit application for maximum performance and capability on all systems, even under DOS. It typically requires about 800 KB of user space for execution, which is usually available on even the smallest DOS-based PC. On multi-user operating environments most of this space is “shared” code, so that additional users only require about 200 KB of extra space each.

File and Devices

User programs can use any device or disk that the underlying operating system supports. Low-level hardware emulation file and device assignment can be done using the virtual console.

RDOS programs can use a powerful logical name translation facility to conveniently access native files and devices without changing any programs.

Hardware-level emulation can assign emulated devices to any device/file supported by the underlying operating system.

Performance

Like automobile mileage, application performance "mileage" may vary. There are a vast number of variables affecting system speed and overall performance, including:

- Type and speed of processor(s)
- Type and speed of disk(s)
- Type of application program
- **Re*NOVA*te** execution mode selected: hardware level or optimized RDOS level
- instruction mix of applications (CPU and I/O)
- Any special tuning performed
- Single-user versus multi-user application
- Other resource loads on target system

Given the previous disclaimer, three major areas significantly contribute to performance and efficiency.

RDOS mode or low-level emulation

RDOS programs can run in "RDOS mode" where **Re*NOVA*te** handles all RDOS system calls at maximum, native speed; or run at "low-level hardware emulation" mode where the operating system code along with the application program is run by **Re*NOVA*te**. Letting **Re*NOVA*te** handle system calls natively is significantly faster than emulating the RDOS operating system at the hardware level. Therefore, RDOS programs running in "RDOS mode" are generally much faster than running in "low-level hardware emulation" mode, and RDOS mode is therefore the preferred method.

CPU performance

Target system CPU performance directly affects system performance, especially in a CPU-bound computing environment. If an application is more I/O oriented, then processor speed becomes less significant than peripheral I/O speed.

Since computer technology rapidly continuously enhances computer performance, any performance "benchmarks" are out-of-date by the time they are published. However, some general guidelines can illustrate the performance gains provided by **Re*NOVA*te** while providing open systems compatibility.

As a general indication of a totally CPU-bound environment, **Re*NOVA*te** running under Windows 95 on a 150 MHz Pentium is more than two (2) times as fast as the original Nova 3 hardware. The same system running on a 333 or 400 MHz Pentium II runs between four (4) to five (5) times faster than the Nova 3 hardware.

Performance on RISC workstations are even faster, with the DEC Alpha system running over 10 times faster than the original Nova 3 system.

I/O performance

Applications that are I/O bound show significant improvements on open systems compared to the original Nova or Eclipse system – more than the processor performance difference on open systems.

Disk speed and caching are very critical to application performance, and are usually more configurable on open platforms than on original hardware and operating systems. A 90 MHz Pentium PC laptop running an RDOS COBOL or FORTRAN compile under DOS may be four (4) times faster than on a comparable Eclipse S/120. Target systems with better disk management (i.e. Windows NT or U*IX) exhibit even better performance increases, sometimes ranging at eight (8) or ten (10) or more times faster performance than the corresponding application running on the original DG hardware.

Related Documents

The concepts described in the appropriate original vendor documents, including those from Data General, Point 4, Strobe Data and others, are applicable for the corresponding version of **Re*NOVA*te** in low-level hardware emulation mode.

RDOS manuals are all applicable for **Re*NOVA*te** in the RDOS emulation mode. These include the following DG publications:

- Introduction to the RDOS System
- Learning to Use Your RDOS/DOS System
- RDOS Reference Manual
- RDOS CLI Reference Manual

DG Documents

You should refer to the appropriate DG publications to answer questions about the DG RDOS system, languages and utilities.

Processor, peripheral and miscellaneous hardware information is also contained in the appropriate Data General Nova and Eclipse manuals, and may be referenced as required.

Pertinent information is also contained in Data General software release notes. These are normally provided with each release of a corresponding DG product, and contain the latest information about the use, features, restrictions and limitations about the product and language.

Point 4 Documents

You should refer to the appropriate Point 4 *Computer System Manual* documentation to answer questions about the Point 4 computer hardware.

Processor, peripheral and miscellaneous hardware information is also contained in the appropriate Point 4 *Peripherals Interface Manuals*, and may be referenced as required.

Pertinent software information is also contained in Point 4 IRIS software release notes. These are normally provided with each release of a corresponding Point 4 product, and contain the latest information about the use, features, restrictions and limitations about the product and language.

Strobe Data Documents

You should refer to the appropriate Strobe Data *Hawk Technical Reference Manual* to answer questions about the Point 4 computer hardware.

Processor, peripheral and miscellaneous hardware information is also contained in the appropriate Strobe Data *Hawk User's Manuals*, and may be referenced as required.

Wild Hare Documents

Separate manuals exist to document the various Wild Hare products, including:

- *Axis* User's Manual
- *Choice!* User's Manual
- *HareStylist* User's Guide
- **WH/Sort** User's Guide

The appropriate manual should be consulted to answer questions about the corresponding product.

Each **Re*NOVA*te** release usually contains a release note text file on the release media - normally called **readme.txt** - which should be reviewed for the latest information.

SimuLogics Documents

Separate manuals exist to document the various SimuLogics products, including:

- **Re*NOVA*te** User's Manual (this document)

Operating System

You should also be familiar with the operation of your computer and its operating system. The hardware and software documentation will vary depending upon system you are using, but usually the various system reference guides and manuals will lead you through the system utilities and editors.

Typical operating systems run under **Re*NOVA*te** include RDOS, DOS (the original DG version from 1970-1972), IRIS, BITS, MICOS, BLIS/COBOL, DDOS and other special-purpose, proprietary application environments/operating systems.

How To Use This Manual

The **Re*NOVA*te** system is an important component in the set of solutions for Data General users provided by Wild Hare. Other Wild Hare tools include the *Axis* compiler, *Choice!* runtime system, *HareStylist* object editor and the **WH/Sort** utility system. You should refer to the appropriate documentation for further information about these other products.

Manual Organization

This User's Guide describes main **Re*NOVA*te** features, how to install the system and use the various options. The System Reference provides a complete technical description of the various runtime features. The Appendices will eventually contain reference information and summaries for quick use.

User's Guide

The first four chapters in the User's Guide section cover information you need to learn to install and use the **Re*NOVA*te** system.

Chapter 1 presents an overview of the **Re*NOVA*te** product.

Chapter 2 describes how to install **Re*NOVA*te** on various systems. You should be able to install **Re*NOVA*te** and run the small sample program in a

few minutes by following the simple steps described in this chapter for your specific operating system.

Chapter 3 describes the many **Re*NOVA*te** command line options and how to use them.

Chapter 4 presents information about the Virtual Console utility, its functions and use.

Reference Manual

The remaining chapters describe the two main two ways of using **Re*NOVA*te**, and provide subsequent reference information.

Chapter 5 gives you more detailed information about specific RDOS technical areas and their corresponding support and emulation.

Chapter 6 gives you more detailed information about the low-level hardware emulation technical features, options and their corresponding implementation.

Chapter 7 provides compatibility notes about various aspects of hardware and software emulation.

Chapter 8 describes the features and use of user 'C' routines in the **Re*NOVA*te** environment. This is an extremely powerful way to extend the system capabilities in a compatible, version-independent convenient manner.

Appendices

Handy reference material and summaries exist in the appendices:

Appendix A summary of RDOS system calls and their implementation notes

Appendix B ASCII reference chart

Appendix C Virtual Console command summary

Appendix D Peripheral device codes

Notations

Many operating system command line examples appear in this manual. Since some operating systems and/or user command processors are very key and case sensitive, it is very important to ensure you honor the exact file names given in any example.

command line terminators

Most operating system command line and batch file processors require each user command line to be terminated with a <return> or <enter> keystroke. Unless

otherwise explicitly stated in the accompanying text, we assume that each command line example is so terminated before the command line will be processed.

examples

Also, some of the sample command lines, text file contents and user displays may be enclosed in a box for clarity:

```
# instrenovate
```

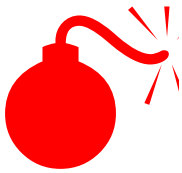
special notes

A “helping hand” in the margin indicates helpful hints or general notes of interest. These items may save you time and make your job easier.

I am a sample helpful hint. Read me to benefit from a timesaving comment, or perhaps some other suggestion to make your job easier.

Important points or key items are highlighted by using a separate “key” away from the rest of the text, such as this one:

This is an important note or key information. It is set apart from the rest of the surrounding text because it is of unique importance.



Using a separate “bomb” sign positioned in the margin shows critical warnings. Be sure to read these items since they may dramatically affect your system's operation.

This is a critical warning. It is set apart from the rest of the document because it is of critical importance to system operation.

Chapter 2

Installing Re*NOVA*te

Re*NOVA*te is quite simple to install even though it runs on a wide range of computers and operating systems. This chapter describes the steps required to properly install it on any system.

Installation Steps

All systems follow these common steps in some form to successfully install and run **Re*NOVA*te**:

1. create a subdirectory to contain the **Re*NOVA*te** files
2. copy the **Re*NOVA*te** files from the release media to the appropriate subdirectory
3. **DOS** and **Windows 3.x users** should install the hardware device into the PC's parallel port,

   -or -

   all other users, such as **PC Network**, **Windows 95/98/NT**, **OS/2, UNIX**, **XENIX** users, must run the appropriate software installation program
4. test proper system installation by running the sample RDOS program included on the release media
5. set up the various system options for your own needs and enjoy using the most advanced and convenient way to run Nova and Eclipse programs on open systems!

System Classes

Although all installations go through similar procedures, each "class" of operating system - such as DOS, UNIX, VMS, etc. - requires a unique series of commands specific to that system. The different "classes" covered in this chapter are:

DOS any standard, stand-alone, single-user PC running an MS-DOS compatible operating system, including PC-DOS, DR-DOS and others. This also includes **Re*NOVA*te** running in DOS compatibility mode, and the native Windows 3.x **Re*NOVA*te**.

Windows 9x/NT and PC Networks any Windows 95/98/NT or multi-user PC environment running under an MS-DOS compatible operating system, usually in a network arrangement. Any DOS-compatible network may be used. DOS-compatible single- and multi-CPU timesharing systems are also included in this category, such as PC-MOS, DR-DOS, TSX-32, VM/386 and others. OS/2 is included in this category.

UNIX ...or UNIX-like systems, including UNIX, XENIX, AIX, DG/UX, HP/UX, Solaris, IRIX, SCO, Risc/OS, and all of the other "IX"s offered by so many vendors (generally based on a "System V" standard).

Release Files

Each system release contains the following logical files, but the actual file names may vary depending upon the operating system:

ReNOVAte the emulator executable file

Install or instrenovate

the **Re*NOVA*te** software installation program (not normally used for DOS)

readme.txt a text file which contains latest release information

rcli.sv a sample test program

It is very important to read the readme.txt text file since it contains the latest system and documentation changes and notes.

renovate Subdirectory

We suggest that you load the contents of the release media into a subdirectory named **renovate** off a main (usually root) directory. This logically separates the **Re*NOVA*te** release files from other system and user files, and makes backups and update loading easier without changing any operating shell scripts or procedures.

You should have any operating system file and access permissions properly set up for full execute/read/write privilege for loading purposes.

Read/write permission to the **Re*NOVA*te** file is needed to install the software on any system which comes with the **instReNOVAte** or **install** program. Only read and/or

execute permission should be needed to actually run the software ***after*** it has been successfully loaded.

Software Installation

You typically run the **install** or **instrenovate** "software installation" program on all systems **except** single-user, stand-alone DOS systems. This procedure associates your copy of **Re*NOVA*te** with your particular computer and disk drive. If you want to move the compiler to a different disk drive, or if you have a hardware failure which requires reloading your system, you will probably need to reinstall the **Re*NOVA*te** system.

This simple software installation procedure is performed by the **install** or **instReNOVAte** program, and is described in detail later in this chapter.

> If a computer hardware failure requires you to reload **Re*NOVA*te**, you will probably need to perform a new software installation if your version of **Re*NOVA*te** does **not** use a software protection device.

The following pages describe the system-specific installation procedure for each "class" of system. To continue **Re*NOVA*te** installation, please refer to the page appropriate for your particular computer and operating system:

Re*NOVA*te for DOS Systems

Re*NOVA*te is typically provided to all MS-DOS compatible systems on a single 3.5" DOS 3.*x*-compatible diskette.

Each release diskette contains the following files:

ReNOVAte.exe	the compiler program's executable file
Install.exe	software installation program
readme.txt	a text file which contains the latest release information
rcli.sv	a sample test program

You should create a subdirectory named RENOVATE on your main hard disk. The contents of the release diskette should then be COPYed into this subdirectory.

Next, the **Re*NOVA*te** hardware device should be connected to a parallel port attached to the computer. This device may be attached to any parallel port which DOS recognizes, and may be used on the same port which is currently connected to a printer. If a printer cable is already connected to the computer, the hardware device should be connected between the cable and the connector on the computer. The printer will continue to function with the hardware device attached.

The following example shows the typical commands required to install **Re*NOVA*te** on the **C:** hard disk (note that the ">" symbol is the DOS ready prompt and should *NOT* be typed in by you):

```
> C:
> CD  \
> MKDIR  \RENOVATE
> CD  \RENOVATE
> COPY  A:*.*
      ...DOS lists here the release files copied...
>
```

You should now proceed to the "Running the Test Program" section on page 25.

Re*NOVA*te for Windows 9x/NT and PC Networks
(including Multi-user DOS and OS/2)

Re*NOVA*te is provided to all PC MS-DOS, PC-DOS and OS/2 networks, PC network systems, Windows 9x/NT and OS/2 on a single 3.5" DOS 3.x-compatible diskette.

The following files are on the release diskette:

renovate.exe	the compiler program's executable file
Install.exe	software installation program
readme.txt	a text file which contains the latest release information
rcli.sv	a sample test program

You should create a subdirectory named **renovate** on your main hard disk. Normally this would be a "public" directory so that all appropriate users could access the compiler from their computers. The contents of the release diskette should then be COPYed into this subdirectory.

Next, the **Re*NOVA*te** software installation program should be run to associate your copy of the software with your particular computer and disk drive.

> You should only install **Re*NOVA*te** while the network or multi-user operating system is running.

Note that Windows 95/98 and Windows NT users can install **Re*NOVA*te** from within a DOS session, an NT command line session or via the GUI interface. OS/2 users should normally install **Re*NOVA*te** from within a DOS session or command line session.

The following example shows the typical commands required to install **Re*NOVA*te** on a **F:** hard disk (note that the ">" symbol is the DOS ready prompt and should *NOT* be typed in by you):

```
> F:
> CD  \
> MKDIR  \RENOVATE
> CD  \RENOVATE
> COPY  A:*.*
      ...Windows lists here the release files copied...
> install
      ...this invokes the software installation program
      which is described later in this chapter...
```

Please proceed to the "Software Installation Procedure" section on page 25 to finish **Re*NOVA*te** installation.

UNIX (and UNIX-like) Systems

Re*NOVA*te is provided to UNIX and UNIX-like systems on a single diskette or tape in **tar** format. Most 5.25" diskettes use the high-density '96ds15' format, and are appropriately labeled. 3.5" diskettes use the high density '135ds18' format, and are also appropriately labeled. 1/4" and 1/2" tapes are formatted to the default **tar** specification for the particular system.

The following files are contained on the release media:

ReNOVAte	the compiler program's executable file
instReNOVAte	the **Re*NOVA*te** software installation program
readme.txt	a text file which contains the latest release information
rcli.sv	a sample test program

You should create a subdirectory named **renovate** on your main hard disk. The contents of the release diskette should then be copied into this subdirectory using the tar command. Next, the **instReNOVAte** program should be run to enable **Re*NOVA*te** operation.

The following example shows the typical commands required to install **Re*NOVA*te** in the root directory on the main hard disk (note that the **"#"** symbol is the shell ready prompt and should *NOT* be typed in by you):

```
# cd /
# mkdir /renovate
# cd /renovate
 tar xv
      ...the tar program lists the files copied from the default
tar device...
# instrenovate
      ...this invokes the software installation program which is
described later in this chapter...
```

You should make sure that your current directory is specified in your search path, otherwise the shell processor will not look in the directory for any programs to run. This can be accomplished with the Bourne shell command:

```
PATH=:$PATH
```

or with the csh command

```
setenv PATH :$PATH
```

You should have full read/write access privileges to the dirctory before attempting installation.

Note that U*IX file/directory names are case sensitive, so you may use any names which will be easy to type and use.

You should now proceed to the "Software Installation Procedure" section on page 25 to finish **Re*NOVA*te** installation.

Software Installation Procedure

Re*NOVA*te uses either a "hardware" or "software" product protection technique.

Target computers which have a parallel port which can be conveniently user-accessed - usually PC's - use the hardware technique. Users of these systems should skip this section and proceed directly to page 25 to check out proper system installation.

Other users must go through the simple software installation procedure described below.

The following software installation does **not** normally apply to stand-alone DOS, systems. If your system came with a hardware protection device, you should attach the device to one of the parallel ports on your computer, then proceed to the next section in this manual, "Running the Test Program".

The **Re*NOVA*te** software will not actually run on your machine until the **install** or **instReNOVAte** program is used to enable execution under unlimited PC networks, UNIX-like, Windows 95/98 and NT systems. The **install** program on DOS and Windows systems, and **instrenovate** program associates your copy of **Re*NOVA*te** with your particular computer and disk drive. If the **Re*NOVA*te** program file is reloaded, copied or otherwise moved, it will probably require reinstallation.

Note that in the following examples, DOS and Windows users should run the program **INSTALL.EXE** rather than the **instrenovate** program.

Assuming that you are in the directory which contains the release files, you install **Re*NOVA*te** with the simple command line (note that the '#' symbol is the command interpreter's ready prompt in this example and should *NOT* be typed in by you):

```
# instrenovate
```

The **instrenovate** program will identify itself by displaying a version message, then start processing the **Re*NOVA*te** program file itself. The contents of the **instrenovate.txt** file will be displayed on the console if the file exists. This is done to call attention to any items of importance which may concern the compiler and its use. If multiple CRT screens full of information exist in the file, you will be asked to hit the <enter> key after each page is viewed. After the final page of information is viewed and the return key is pressed, in about 30 seconds it will display a set of two different numbers, called "keys", and will ask you for a "user number".

For example, a typical UNIX and UNIX-like system screen at this point would look like this:

```
Software Installation Utility [3.0a.4] for UNIX ReNOVAte
[3.0a.00]
Copyright (C) 1984-1998
All right reversed worldwide

<contents of instrenovate.txt file, if any, displayed here>

Begin - UNIX ReNOVAte - Installation...
(1):  00000028
(2):  038fcab9
Please enter key code from Wild Hare>>
```

At this point you will need to call your software supplier or Wild Hare and give them the values of the two eight-digit hexadecimal numbers displayed. You will in turn receive a Wild Hare key which must be typed in.

This number is also a hexadecimal number, which means that it may have letters in it as well as regular numbers. The letters, if they exist, may be typed in **either** lower or upper case. Since the numbers are hexadecimal, the only letters which may be used are 'a'-'f', either upper or lower case. Therefore, do not think that a '0' (zero) digit is actually the letter 'O' (oh).

After this number is properly typed in, the installation program will continue processing for about 10 more seconds, then indicate that **Re*NOVA*te** is installed.

An appropriate error message will be displayed if there are any errors encountered during this installation process, and you will be prompted to try again.

Re*NOVA*te is designed to work with most "speed disk" programs which run under DOS and DOS networks by specifying that the **Re*NOVA*te** file should not be moved. However, some VAX/VMS "defragmenters" may move portions of **Re*NOVA*te**, confusing the installation information created by the **instReNOVAte** program. Please contact Wild Hare if you have any questions about your particular situation.

Running the Test Program

If everything has been done properly up to this point, you should be able to run the sample RDOS **rcli.sv** program by simply typing in the command (note that the '#' symbol is the native operating system command interpreter's ready prompt and should **not** be typed in):

```
# renovate rcli.sv
```

Sample program

You should see something similar to the following displayed on your console:

```
UNIX ReNOVAte, version 3.0a.00
Copyright (C) 1984-1998 by SimuLogics
All rights reserved worldwide

R
```

Note that the heading will reflect your particular version of **Re*NOVA*te**, and the statistical information will vary.

This sample program is the RDOS CLI program, so you should be able to perform the various DG RDOS CLI functions that you have known and loved all these years, such as LIST, GDIR, GTOD, GDAY, etc.

To exit the CLI and terminate the **Re*NOVA*te** session, simply type

```
EXIT
```

as a CLI command. This will return you to the native operating system prompt that you started with.

Common Problems

Problems may be encountered in several different areas, the most common including:

- not having your current directory in your shell path
- not having proper access privileges to the directory containing **Re*NOVA*te**
- getting errors reading the release media
- using incorrect case for commands on case-sensitive operating systems
- trying to install **Re*NOVA*te** with an older version of the **instReNOVAte** or **install** program
- on network systems, by not installing **Re*NOVA*te** when the network is actually running

If you check these things and still have problems loading **Re*NOVA*te**, please contact your software supplier or Wild Hare.

What Do I Do Now?

Running the **rcli.sv** sample RDOS program is intended to make sure that all of the release files were properly loaded and that you are able to run a simple RDOS program with all default options. But there are many **Re*NOVA*te** options and features which exist to make your life easier. The next chapter describes the many options and features available and how you can best use them in your own development environment.

...and have fun using the best RDOS, Nova and Eclipse development and execution environment available!

Chapter 3

Using Re*NOVA*te

Overview

This setion answers the following important questions about using **Re*NOVA*te**:

- What do I want **Re*NOVA*te** to do?
- What program and data files should be acted upon?
- Where are the files located?
- What operating system features can I use?
- What system shall be emulated?
- What if I am a "power user"?

...and how do I tell **Re*NOVA*te** all of this?

Controlling Re*NOVA*te

The answers to these questions and others, such as configuration and subsystem control, are given to **Re*NOVA*te** through several different methods:

- default values
- command line options
- command option files
- environment variables
- specific definition files

Default values

Default values exist for every option in the system and determine system behavior if you don't specify any overriding option. Their values have been chosen to provide the most flexible system operation, or support the most commonly-used features.

Command line options

Command line options are the most frequently-used way to control **Re*NOVA*te** actions. They are typed on the operating system shell's command line, or are included on the command line in a "script" or "batch" file.

Option files

Command option files are identical to the command line options typed in on the operating system's command line, but are contained in disk files. This gives you the simple command line option format with the convenience of saving many options in a disk file in an operating system independent form.

Environment variables

Environment variables supported by most modern operating systems allow you to control **Re*NOVA*te** operation without using command line options or requiring command option files. They also allow you to selectively modify a specific user's operation when using a common batch or script file for all users.

Definition files

Definition files are files which specific names that control certain **Re*NOVA*te** subsystems. These files are automatically searched for when **Re*NOVA*te** starts execution and have pre-defined names as described later in this section.

Option priority

Any of these methods may be used individually or in any combination. If multiple methods are specified, the priority to determine the final option chosen is (from highest priority to lowest):

- option specified on the command line
- options contained in definition files
- environment variables
- default values

Note that the command line options contained within option files is considered the same as options specified on the command line.

Command Lines

Most user command interfaces interact directly with a user through a terminal, or in a "batch" mode through command line procedure files or "shell" scripts. In either case, the operating system "command line" is used to tell your computer to run the **Re*NOVA*te** system. A typical command line to invoke **Re*NOVA*te** has the following format:

```
renovate  [-options] [initial save file]
```

Re*NOVA*te program name

The first item in the command line is the name of the **Re*NOVA*te** runtime program itself. You may specify the name with or without any directory specifier prefix, and you must have the proper operating security or access permissions to execute the runtime. Some operating systems have a "path" or "search list" feature which consists of the directory names to search when looking for the **Re*NOVA*te** program. Some operating systems (i.e. any of the UNIX-type systems, including UNIX, XENIX, AIX, etc.) require you to honor case sensitivity of file names, meaning that renovate and RENOVATE are two totally different program names.

Due to a "feature" of UNIX[-like] operating systems, a program can not determine how it was invoked (i.e. where the ReNOVAte program was "found") if started up via a shell path search. Therefore, use explicit directory specifiers when starting up ReNOVAte on UNIX[-like] systems so the software security can properly operate.

command line options

[-options] refer to the "command line options" which tell **Re*NOVA*te** what to do and how to do it. The square brackets ('[' and ']') are not actually typed on the command line, but are shown here to indicate that the options are...well...optional and are not required. These options give you the flexibility to handle almost any special situation you encounter, yet they all follow a few simple guidelines which are described below. Each option and its use is fully described later in this chapter.

positional dependence

Although the command line to run the sample rcli.sv program would take the form of the previous example, you have many alternatives when running your own programs. Command line options are not position-dependent in the **Re*NOVA*te** command line. For example, if you wanted to allow input time-out aborts and initially run the program MYMENU, you could use either of these command line :

```
renovate -wait MYMENU
```

or

```
renovate MYMENU -wait
```

option formats

Re*NOVA*te supports many command line options to give you the most flexibility, compatibility, portability and power possible. In addition, all of the command line options are available on all computers under all operating systems. In spite of the large number of options, they all follow some simple guidelines:

option delimiter

Each command line option is preceded with a hyphen ('-') delimiter to indicate that it is a command line option and not an initial program name. There should be no space or other characters between the delimiter and the option name. This means a typical option may look like this on a command line:

```
-nocolor
```

case sensitivity

Command line option names are not letter case sensitive, that is, upper and lower case letters are treated identically by **Re*NOVA*te**. However, any filenames used as option arguments or initial programs to run are case sensitive as treated by the operating system.

option name uniqueness

You must specify the full command line option name after the delimiter. For example, suppose -delete and -detour were command line options which **Re*NOVA*te** recognized. In prior releases you could use the command line option -DEL or -del to specify the -delete option, but not -d or -de because **Re*NOVA*te** would not know whether you meant -delete or -detour. Now **Re*NOVA*te** will display an error message if it detects

an unrecognized command line argument, and terminate system initialization and return to the command line or batch file that called **Re*NOVA*te**.

option arguments

Some command line options require an argument, usually a number, file name or text string. In these cases the command line option and its argument are separated with an equal sign character ('='). There should be no spaces or other characters between the command line option and the equal sign, or the equal sign and the argument. For example, a numeric argument to the -PDsize option would look like this:

```
-PDsize=64
```

Numeric arguments have certain valid ranges depending upon the option. An appropriate fatal initialization error message is displayed if an invalid number is entered.

option separation

Each command line option must be separated from other options by one or more space and/or tab characters, but no spaces may normally be embedded within any option itself or any option arguments and the option. This is also true of options contained in command files as described later in this section.

The exception to this rule is when double quotes are placed around a command line option and/or its possible argument. Most operating system command line interpreters will allow imbedded spaces within a double-quoted (") command line option.

filename translations

Any program and data file may be translated according to a user-supplied "logical name translation" file. This file contains a list of "logical" file names and their corresponding "equivalent" file names. This powerful logical name translation facility is fully described in chapter 5.

embedded environment variables

A very powerful command line feature exists which positionally substitutes the value of environment variables in command line text. This is especially helpful if you want ot use common startup techniques and command lines but have options and arguments vary depending upon different users.

The percent sign (%) character is recognized as a special "environment variable substitution" indicator within command line text. If two '%' characters are separated

by one or more characters in an area of text, then the text enclosed by the '%' characters are considered an environment variable name. The operating system is asked to return a corresponding string value for that environment variable, which then replaces the two '%' characters and text in-between. For example, if the environment variable MY_TERM had the value ADM3A, the command line option

-term=%MY_TERM%

would be treated as

-term=ADM3A

If no corresponding environment variable is defined, then the text would be removed from the command line text. The previous example would be translated to:

-term=

Two '%' characters next to each other is translated to a single '%' character. This lets you unambiguously specify the '%' in command line text.

A single '%' character in a command line option will not be translated. Continuing the prevous example, the command line

-term=%MY_TERM

would be treated without translation.

Note that the '%' translation characters may be used anywhere in an option string. For example, the original option could be:

-term=silly%MY_TERM%terminal

and would be translated to

-term=sillyADM3Aterminal

continuing to use the previous example.

Each command line option (including possible arguments) is scanned from left to right for embedded environment variable substitution. This occurs up to 10 times or until no further substitution is found for each command line option. This provides the powerful capability of "recursion", whereby environment variables may be used as the name of other environment variables that contain the information desired.

command files

Re*NOVA*te supports text files which contain command line arguments so you don't have to type all of the options on the command line. These "indirect command files"

are similar to RDOS ".CM" files and may include any **Re*NOVA*te** command line options or source file names just as if they were directly typed in on the command line. This powerful feature is described in the next section in this chapter.

Command Files

Re*NOVA*te supports text files that contain command line arguments. These "indirect command files" are similar to RDOS ".CM" files and may include **Re*NOVA*te** command line options or source file names just as if they were directly typed in on the command line. The files are specified by prefixing the file name with an '@' character on the command line. Each command file may include other command files, up to a maximum of five (5) nested files deep.

Comment lines

Comments may appear in these command files by placing a semi-colon (;), pound sign (#) or exclamation point (!) in the first position in a line. Everything else in the current input line to the right of the ';', '#' or '!' will be ignored.

Option Format

The contents of command files follows the rules of standard shell command line options. Each option must be normally separated from others by at least one space and/or tab character.

Arguments to an option must not have any spaces between the option name, the '=' character and the argument itself.

Multiple arguments to a single option may normally be specified by separating the arguments with a comma (',') character.

Examples

For example, if the file OPTIONS contains the following:

```
! sample command line option file:  OPTIONS
! this file uses another "include" or "command" file
! called MYDIRS which contains the list
! of the directories in our "search list"

-console=dgcrt

! notice this comment after the -allow option

@MYDIRS
```

```
! end of file OPTIONS
```

and the file MYDIRS contains:

```
! sample command line option file:  MYDIRS
! this file contains the list of the directories we want to use

-path=/usr/data;/usr/temp

! end of file MYDIRS
```

The main command line to invoke **Re*NOVA*te** would then look like this:

renovate @OPTIONS

Any file name case sensitivity requirements of the operating system should be recognized.

Note that blank lines in command files are ignored.

ReNOVAte automatically searches for an option file with the default name of renovate.ini. This makes it very easy to keep most of the desired command line options in one place (file), yet still allows additional commands be typed on the command line.

Definition Files

Some **Re*NOVA*te** facilities require more information than can be conveniently placed on a single command line. Therefore, ReNOVAte will look for controlling information in a file with a default name to control a corresponding feature. For example, the logical name translation information is assumed to be contained in the file renovate.log.

Definition files are not required for the system to run. Default values for the corresponding **Re*NOVA*te** facility will be assumed if the corresponding file does not exist, or the file contains no information.

File format

Each definition or control file is a text file which can be created and modified using any standard text editor. The format of each individual file is consistent across all hardware and operating systems, insuring maximum portability. Each feature's definition file may have a different internal format dependent upon the features individual unique requirements.

Comments may be added to any file by starting a text line with the '!', '#' or ';' character. This is demonstrated in the various sample definition files provided with each **Re*NOVA*te** release.

Automatic "search list"

Re*NOVA*te searches for each of the definition files when it initially starts running for each user. Since each definition file is a text file read in only when **Re*NOVA*te** is started, it may be modified any time up to invoking the runtime system.

Re*NOVA*te will look for each definition file in the current "default" directory before giving up and using the default values for the corresponding facility:

default directory

The current default directory is the one in which the runtime is started.

prefixed directory

The directory prefix is that potential directory specifier used to tell the operating system where the **Re*NOVA*te** runtime file was. For example, if the **Re*NOVA*te** system

under a UNIX system is contained in the /renovate directory, the command line to invoke **Re*NOVA*te** might be:

/renovate/renovate

In this case /renovate is the directory prefix, and is the directory name added to the beginning of each of the definition file names to search of it if the file is not found in the current default directory.

This implied "search list" exists **only** for the startup definition files, and is **not** used for any user program or data files. User file search lists may be implemented with the **-path, -progpath** and/or **-datapath** option.

A program invoked on most UNIX[-like] systems can not determine if the shell's search path was used to find the program. A program can not therefore find out what directory the executable file resides in. This "feature" prevents **Re*NOVA*te** (or any other program) from knowing its startup directory unless you use a directory specifier on the program's name. For example, use **/renovate/renovate** on the shell command line (or in a script file) to explicitly tell **Re*NOVA*te** that was started from the **/renovate** directory. **Re*NOVA*te** will then look in that directory for all of its definition files.

Optional files

Each of the remaining definition files is optional, and need not exist for **Re*NOVA*te** to run. A detailed description of the various definition files is contained in the Chapter 5 section describing each feature.

renovate.ini *default command line option file*

This is the default command line option file that is searched for when the system is started.

renovate.log *logical name translation file*

Filename substitution, string and substring replacement and certain **Re*NOVA*te** facility definitions are contained in this file.

renovate.vc *default virtual console file*

Filename substitution, string and substring replacement and certain **Re*NOVA*te** facility definitions are contained in this file.

Other files may be included as required for a specific release. The **readme.txt** file should be consulted for a complete list if there are additional files.

Environment Variables

Another way of overriding default **Re*NOVA*te** option and feature values is by using environment variables. It is assumed that an environment variable facility, or its equivalent, is supported by your operating system. Environment variable facilities exist under the following operating systems:

DOS, **DOS networks**, **OS/2**, **Windows**

Environment variables are supported by the operating system and manipulated with the SET command.

UNIX, XENIX, AIX, DG/UX and other "IX"es

Environment variables are supported by the operating system and maintainable by all shells.

VAX/VMS

Environment variables are DCL logical names and are maintained with DCL SET and SHOW LOGICAL commands.

Environment Variable Structure

Environment variable definitions override the default **Re*NOVA*te** features and options, but may in turn be overridden by both command files and command line arguments.

The use of environment variables is entirely optional, since all of them have a corresponding command line option.

Environment variables perform two main **Re*NOVA*te** functions:

- provide alternative to command line option control
- override default filenames of definition files

Option Environment Variables

The first reason to use environment variables is to control **Re*NOVA*te** options and features without using command line options. **Re*NOVA*te** will search for and use the following environment variables to obtain default values if the variables exist in your application environment:

RENOVATE_PATH

This determines the default directory search path to use when searching for data and program files. It has the same function and format as the **-path** command line option. If this environment variable exists along with the **-path** command line option, the environment variable's value will be ignored. This is the same function as the **-PATH=** command line option.

Definition Filename Environment Variables

The second reason to use environment variables is to override the default **Re*NOVA*te** definition file filenames. The file name extension of each definition file is used to create a unique environment variable name. If an corresponding environment variable exists, its value is used as the full file name - including the full path, filename and extension - of the desired file.

Environment Variable	Corresponding Definition File
RENOVATE_LOG	renovate.log
RENOVATE_INI	renovate.opt
RENOVATE_VC	renovate.vc

Use of these environment variables is entirely optional. The corresponding option or file name's default value will be used if no environment variable exists.

Command Line Options

Command line options can fall into one of three main groups:

- a logical flag value
- a numeric item, or
- some other miscellaneous option

logical options

A logical flag can usually be turned on or off, and therefore each such flag usually has a command line option to turn on and turn off the flag. For example, a -abort command line option exists to enable automatic console time-outs, and the corresponding -noabort command line option disables the feature. Since each logical option usually has both a name to enable the option and a corresponding "no"-name to disable the option.

Remember that each option must be separated from other command line options with "whitespace" (that is, one or more spaces and/or tabs), but no space should exist between the "-" and the option name.

numeric options

Numeric options usually control functions such as the number of lines per page or the size of internal tables. The option's numeric argument is separated from the option itself with an equal sign (=), with no spaces between the option name, the equal sign and the number. Numbers should usually be less than 2000000000, two billion, and should not contain any commas.

miscellaneous options

Miscellaneous options usually control file-oriented features, such as search path directory lists. The option's argument is also separated from the option itself with an equal sign, with no spaces between the option name, the equal sign and the text argument.

Some command line option descriptions may be duplicated in different logical sections, but this is due to the option's function possibly fitting into more than one area.

Alphabetic Summary

The following section lists the **Re*NOVA*te** command line options in alphabetic order. Full explanation of the various features is contained in the appropriate section in Chapter 5 and Chapter 6.

The default option value is specified where applicable.

-?

Display a summary of the command line options. Details about possible arguments for a specific option may be displayed by typing the command line option name with a "?" as the argument

-console=<*option*>

Specify whether or not DG terminal emulation will occur for .WRL, .WRS and .PCHAR system calls. Valid options are:

DGcrt	emulate DG terminal using ANSI control sequences
NoDGcrt	do not emulate DG terminal codes This is the default.
uppercase	translate console input to uppercase This is the default
nouppercase	do not convert console input to uppercase.

-datapath=<*pathlist*>

Define a search path to be used by the RDOS user environment for data file opens. This will not be used for .EXEC, .EXFG, .CHKP or .OVOPN system calls, but will be used for every other type of file open call.

-deviceDisable=<*dev_list*>

Specify the device names and/or numbers that are to be disabled when the system is started. The list of device names and numbers may be separated with ',', ':', ';', '/', ' ', and '\t' characters.

-deviceEnable=<*dev_list*>

Specify the device names and/or numbers that are to be disabled when the system is started. The list of device names and numbers may be separated with ',', ':', ';', '/', ' ', and '\t' characters.

-deviceio=<*option*>

Determine how unsupported I/O device instructions are handled. <*option*> may be one of the following items:

Break	Break to the Virtual Console if unsupported I/O device instruction is encountered. This is the default
Ignore	Ignore unsupported I/O devices.

This may be overridden on a device by device basis with the VC 'set device' command.

-dirpath=<*pathlist*>

Define a search path to be used by the RDOS .DIR system call. No other file operations are affected by this option's value.

-DKBcache=<*boolean*>

Force main memory to be used to "cache" the entire contents of the disk. This may dramatically speed up disk I/O operations, but will consume memory in the amount of the size of the entire disk.

-DKBdebug=<*boolean*>

Display debug information concerning disk I/O operations on the console.

-DKBdevice=<*integer*>

Specify the exact Data General DKB disk device code. This should be a decimal number in the range of 0 - 63. The default is <026> octal, 30 decimal. The standard secondary device code is <066>.

-DKBdiscutil=<*boolean*>

File used by the disk controller is in IRIS/BITS "discutil" format.

-DKBnative=<*boolean*>

This command line option allows use of the native operating system's byte endian format for the disk file associated with DKB units. Strobe Data Falcon and Hawk board container files use native format on PCs, so supports direct use these files without modifications.

The default is FALSE, meaning that the Data General big-endian format is default for DKB "container" file.

-DKBoffset=<*integer*>

Specify the first disk sector offset into the raw disk file (the "container" file) for the DKB device. This is 0 by default, but may be set to 512 to directly use BITS 'diskutils' backfiles directly without conversion

-DKPcache=<*boolean*>

Force main memory to be used to “cache” the entire contents of the disk. This may dramatically speed up disk I/O operations, but will consume memory in the amount of the size of the entire disk.

-DKPdebug=<*boolean*>

Display debug information concerning disk I/O operations on the console.

-DKPdevice=<*integer*>

Specify the exact Data General DKP disk device code. This should be a decimal number in the range of 0 - 63. The default is <033> octal, 27 decimal. The standard secondary device code is <073>.

-DKPdiscutil=<*boolean*>

File used by the disk controller is in IRIS/BITS “discutil” format.

-DKPgeometry=<*integer*>

Specify the exact Data General DKP disk geometry using an integer which has the format CCCCHHHSSS, where CCCC is the number of cylinders, HHH is the number of heads and SSS is the number of sectors per track. The default is 589007032, meaning 589 cylinders, 7 heads and 32 sectors/track (all numbers are decimal).

Note that decimal numbers do not start with a ‘0’ character, but octal numbers do.

-DKPnative=<*boolean*>

This command line option allows use of the native operating system's byte endian format for the disk file associated with DKP units. Strobe Data Falcon and Hawk board container files use native format on PCs, so supports direct use these files without modifications.

The default is FALSE, meaning that the Data General big-endian format is default for DKP "container" file.

-DKPoffset=<*integer*>

Specify the first disk sector offset into the raw disk file (the "container" file) for the DKP device. This is 0 by default, but may be set to 512 to directly use BITS 'diskutils' backfiles directly without conversion.

-DLPdebug=<*boolean*>

Display debug information concerning DLP disk I/O operations on the console.

-DLPdevice=<*integer*>

Specify the exact disk device code. This should be a decimal number in the range of 0 - 63. The default is <045> octal, 37 decimal.

-DLPdiscutil=<*boolean*>

File used by the disk controller is in IRIS/BITS “discutil” format.

-DLPgeometry=<*integer*>

Specify the exact DLP disk geometry using an integer which has the format CCCCHHHSSS, where CCCC is the number of cylinders, HHH is the number of heads and SSS is the number of sectors per track. The default is 589007032, meaning 589 cylinders, 7 heads and 32 sectors/track (all numbers are decimal).

Note that decimal numbers do not start with a ‘0’ character, but octal numbers do.

-DLPnative=<*boolean*>

This command line option allows use of the native operating system's byte endian format for the disk file associated with DLP units. Strobe Data Falcon and Hawk board container files use native format on PCs, so supports direct use these files without modifications.

The default is FALSE, meaning that the Data General big-endian format is default for DLP "container" file.

-DLPoffset=<*integer*>

Specify the first disk sector offset into the raw disk file (the "container" file) for the DLP device. This is 0 by default, but may be set to 512 to directly use BITS 'diskutils' backfiles directly without conversion.

-DSKcache=<*boolean*>

Force main memory to be used to “cache” the entire contents of the disk. This may dramatically speed up disk I/O operations, but will consume memory in the amount of the size of the entire disk.

-DSKdebug=<*boolean*>

Display debug information concerning disk I/O operations on the console.

-DSKdevice=<*integer*>

Specify the exact Data General DSK disk device code. This should be a decimal number in the range of 0 - 63. The default is <020> octal, 16 decimal. The standard secondary device code is <060>.

-DSKdiscutil=<*boolean*>

File used by the disk controller is in IRIS/BITS “discutil” format.

-DSKnative=<*boolean*>

This command line option allows use of the native operating system's byte endian format for the disk file associated with DSK units. Strobe Data Falcon and Hawk board container files use native format on PCs, so supports direct use these files without modifications.

The default is FALSE, meaning that the Data General big-endian format is default for DSK "container" file.

-DSKoffset=<*integer*>

Specify the first disk sector offset into the raw disk file (the "container" file) for the DSK device. This is 0 by default, but may be set to 512 to directly use BITS 'diskutils' backfiles directly without conversion

-DZPdebug=<*boolean*>

Display debug information concerning DZP disk I/O operations on the console.

-DZPdevice=<*integer*>

Specify the exact disk device code. This should be a decimal number in the range of 0 - 63. The default is <027> octal, 23 decimal.

-DZPdiscutil=<*boolean*>

File used by the disk controller is in IRIS/BITS “discutil” format.

-DZPgeometry=<*integer*>

Specify the exact DZP disk geometry using an integer which has the format CCCCHHHSSS, where CCCC is the number of cylinders, HHH is the number of heads and SSS is the number of sectors per track. The default is 589007032, meaning 589 cylinders, 7 heads and 32 sectors/track (all numbers are decimal).

Note that decimal numbers do not start with a ‘0’ character, but octal numbers do.

-DZPnative=<*boolean*>

This command line option allows use of the native operating system's byte endian format for the disk file associated with DZP units. Strobe Data Falcon and Hawk board container files use native format on PCs, so supports direct use these files without modifications.

The default is FALSE, meaning that the Data General big-endian format is default for DZP "container" file.

-DZPoffset=<*integer*>

Specify the first disk sector offset into the raw disk file (the "container" file) for the DZP device. This is 0 by default, but may be set to 512 to directly use BITS 'diskutils' backfiles directly without conversion.

-execnative=<*boolean*>

This determines if RDOS .EXEC and .EXFG calls the native operating system if no RDOS program exists with the specified file. This allows transparent use of native operating system programs in addition to CLI operations.
The default is true.

-filename=<*option*>

Control default file name conversions from RDOS program to the native operating system:

truncate	truncate base filenames to 8 characters. This is normally used only on DOS systems.
notruncate	do not truncate base filenames to 8 characters. This is the default.
DOS	translate filenames to DOS filenames through special translation methods. This lets ICOS and RDOS style filenames be used under DOS systems while still retaining uniqueness, even if 10-character original names are used. This is the default on DOS systems.
noDOS	do not translate filenames to DOS filenames through special translation methods. This is the default on non-DOS systems.
RDOS	truncate base filenames to 10 characters and extensions to 2 or 3 characters.
noRDOS	do not truncate base filenames to 10 characters and extensions to 2 or 3 characters. This is the default.
autocase	attempt to determine user filename case by the initial program request. All subsequent program and data file searches will be done in this case, just as if the -filename=lowercase or -filename=uppercase options were specified. This is the default.
noautocase	do not attempt to determine user filename case.
lowercase	force RDOS-specified filename to lower case. This is done before any special user path or directory specifier(s) are added to the name.
nolowercase	do not force RDOS-specified filename to lower case. This is the default.
uppercase	force RDOS-specified filename to upper case. This is done before any special user path or directory specifier(s) are added to the name.
nouppercase	do not force RDOS-specific filenames to upper case. This is the default.
alllowercase	force RDOS-specified filename to lower case. This is done after any special user path or directory specifier(s) are added to the name, effectively making the whole name - including directory specifiers - lower case.
noalllowercase	do not force entire RDOS-specified filename to lower case. This is the default.

alluppercase	force RDOS-specified filename to upper case. This is done after any special user path or directory specifier(s) are added to the name, effectively making the whole name - including directory specifiers - upper case.
noalluppercase	do not force entire RDOS-specified filename to upper case. This is the default.
case	use both upper and lower case conversions if a file name can't be found.
nocase	don't do any filename translation or case conversion. This is the default.
colons	replace all ":" in filenames with "/", except a filename that starts out with one letter followed by a ':'. Subsequent ':'s in the filename will be converted.
nocolons	do not modify ':' in a filename. This is the default for many operating systems.
dollar	translate all '$' characters in filenames to a '_'.
nodollar	do not translate any '$' characters in filenames. This is the default.
dots	do not change '.' characters in a file name.

This is the default.

nodots	change all '.' characters in a filename to '_' characters, except for the last '.' in a file name.
hyphen	change all '-' characters in a filename to '_' characters.
nohyphen	do not change '-' characters in a filename.

This is the default.

spaces	change all embedded space characters in a filename to '_' characters.
nospaces	do not change embedded space characters in a filename.

This is the default.

environment	emulate DG's facility that searches the operating system's environment variables for any filename that starts with '$'. That is, any filename that starts with a '$' will have the rest of the name considered as an environment variable that should be looked up. If a match is found, the corresponding environment variable value is used in place of the original filename. Filenames that contain a '\', '/', ':' or '[' character will not be translated, as these are already assumed to contain a full file name plus specifier.
noenvironment	do not emulate DG's facility that searches environment variables for filenames that start with '$'.

-FPUmodel=<*integer*>

Specify the exact Eclipse FPU model number supported. This advanced setting is not normally used, but may be required by DG diagnostic programs. Refer to the release notes or contact Wild Hare for the latest settings.

-mdir=<*name*>

Define the name of the master (disk) device of the RDOS environment. This is the device name returned by the RDOS .GDIR system call.

-memsize=<*integer*>

Determine the total memory supported in mapped system emulation. This must be in the range of 32 - 2048, indicating a maximum of 2 MB memory. Hardware memory mapped system support is only available as a special option from Wild Hare.

-mpath=<*path*>

Define the native operating system path that corresponds to the master (disk) device of the RDOS environment.

-MTAblocksize=<*integer*>

Tape data files are usually in a special format that prefixes a data record byte count to each actual data record. This option tells the system that the tape file contains records of a fixed record size with no block size prefix information. This is **not** the default format.

-MTAdebug=<*boolean*>

Tape operation trace information is sent to the user console if this flag is set to TRUE. The default is FALSE.

-MTAdevice=<*integer*>

Specify the exact magnetic tape device code. This should be a decimal number in the range of 0 - 63. The default is <022> octal, 18 decimal. The standard secondary device code is <062>.

-MTAnative=<*boolean*>

This command line option allows use of the native operating system's byte endian format for the disk file associated with MTA units. This capability is provided for completeness, as files are usually in DG-compatible [big endian] format when used by the system.

The default is FALSE, meaning that the Data General big-endian format is default for MTA data files.

-MTAoffset=<*integer*>

Specify the first tape record offset into the raw data stream file for the MTA device. This is 0 by default, but may be set to other values if required by a special situation.

-path=<*pathlist*>

Define a search path to be used by the RDOS user environment for all file open operations.

-program=<*filename*>

Explicitly specify the initial RDOS user program to run. The initial program does not need to have the **/program=** command line option prefix, but this eliminates ambiguity..

-programpath=<*pathlist*>

Define a search path to be used by the RDOS user environment for program load requests.

-QTY=<*option*>

Various QTY multiplexor options are controlled with the command line option. The current valid arguments are:

ALM	emulate primary ALM hardware (device code <034>)
ALM1	emulate secondary ALM hardware (device code <044>)
RoundRobin	round robin synchronous polling of all active multiplexor lines
NoRoundRobin	poll active multiplexor lines from first line to last line
in7bits	mask all input to 7 bits.
in8bits	mask all input to 8 bits.
out7bits	mask all output to 7 bits.
out8bits	mask all output to 8 bits.

debug	enable special mux debugging feature
debug	disable special mux debugging feature
ULM	emulate primary ULM hardware (device code <034>)
ULM1	emulate secondary ULM hardware (device code <044>)

QTYdebugfile=<filename>

Define the QTY trace file.

-RDOS=<*option*>

Control the various RDOS support features and capabilities. Current arguments are:

Break	console break invokes Virtual Console rather than RDOS control/A handler
Nobreak	console break invokes RDOS control/A handler. This is the default
Multitask	multi-task .DELAY, .GCHAR and console .RDS calls This is the default.
Nomultitask	don't multi-task .DELAY, .GCHAR and console .RDS calls
Svcbreak	enter Virtual Console after every SVC call
Nosvcbreak	don't enter Virtual Console after every SVC call. This is the default
RTC10	set RDOS RTC rate to 10 Hz. This is the default
RTC100	set RDOS RTC rate to 100 Hz.
RTC1000	set RDOS RTC rate to 1000 Hz.
RTC50	set RDOS RTC rate to 50 Hz.
RTC60	set RDOS RTC rate to 60 Hz.
Background	RDOS environment assumed to be background. This is the default.
Foreground	RDOS environment assumed to be foreground.

-registers=<*registerlist*>

Define the CPU register values for the initial program. <registerlist> refers to a list of separated octal, decimal or hexadecimal numbers that specify the values of the program counter (pc), accumulators (acs) and carry.

The numbers may be separated by a space (' '), forward slash ('/'), semi-colon (';'), colon (':') or comma (','). This allows command line compatibility with different operating systems.

To set the initial program counter to 2, acs 0-3 to 1, 3, 5 and 7 respectively, and the carry to a 1, the following command line option may be used:

-registers=02/1/3/5;7;1

To ignore or not set an initial value, a minus sign ('-') alone, or no characters between separators, may be used. To not set ac1 in the previous example:

-registers=02/1/-/5;7;1

If values are not specified for all registers, then only the number of registers specified will be affected. For example, to only set the initial program counter, the previous command line option would be:

-registers=02

The program counter will be masked to 15 bits.

The carry flag will be set to 1 if it is defined as any non-zero value, else set to a 0.

All other registers will be truncated to 16 bits.

The commands in any initial Virtual Console command file will override the register values specified in this command line option.

-remap=<*option*>

Various RDOS .REMAP facility features are controlled by this command line option. Note that usually the **-vmem** command line option is used to specify the amount of extended memory to be emulated. Current valid arguments are:

disk	use disk for containing extended memory area information rather than physical main memory. This may be used if main memory available is smaller than extended memory size requested, typically on 16-bit DOS systems. This slows performance, but allows programs to still run on smaller hardware configurations. It may also be used in certain debugging situations.
memory	use only main memory for emulating RDOS extended memory area. This is the default.
check	test for remap conflicts after each remap operation. The physical Nova and Eclipse hardware map unit allows multiple logical address pages to be mapped to a single physical page. This can not be emulated by any software scheme if the user program changes the same physical location by using different logical address. This

argument tells **Re*NOVA*te** to test for multiple logical to single physical .REMAP operations, and call the Virtual Console if this condition is detected.
This is the default.

nocheck	this argument inhibits checking multiple logical to single physical page remapping after each .REMAP request.
compare	If multiple logical to single physical page support is forced, this additional option determines if a test is made after each .REMAP to see if any difference between logical pages exists.
nocompare	do not test for multiple logical page to single physical page inconsistencies after each .REMAP operation. This is the default.

-statistics=<*boolean*>

RDOS system call, RDOS "secret" call and and hardware instruction execution statistics are maintained if this command line option is specified

RDOS statistics are sent to file rdosstat.inf when the **Re*NOVA*te** system is terminated, and CPU instruction statistics are sent to file inststat.inf.

-switches=<*integer*>

Specify the initial computer switch register value.

-tracefile=<*filename*>

Define the file to receive RDOS system call trace information when the TRACERDOS option is active. This file is appended by default.

-traceRDOS=<*boolean*>

Trace the initial RDOS program. This is useful to trace an initial program before any other tracing options can be set from within the RDOS environment.

-userdevice=<*integer*>

Device a specific I/O device to be considered the special "user device". Any I/O reference to this device may cause an external user 'C' routine to be executed, allowing convenient, powerful support for special applications. The <integer> argument must be in the range of 0- 63 (decimal), and must be an unused device code. This reference has lower priority than the -userinstruction argument.

-userinstruction=<*integer*>

Device a specific I/O instruction to be considered the special "user I/O instruction". Any use of this I/O instruction may cause an external user 'C' routine to be executed, allowing convenient, powerful support for special applications. The argument must specify a valid I/O instruction. This option is tested before the user device code option is checked, giving a very powerful feature for extending "legacy" applications with minimal effort.

-userlibrary

Specify the external user library name which contains user 'C' code. Currently this overrides the default USERCODE.DLL library name used under Windows 9x and NT systems when supporting the special user interface feature.

-userroutine

Specify the external user routine name which contains user 'C' code executed whenever a special user instruction or device is referenced. Currently this overrides the default 'user_routine' library name used under Windows 9x and NT systems when supporting the special user interface feature.

-vc=<*boolean*>

This command line option determines if Virtual Console support is desired. If VC support is specified, it will be automatically entered when **Re*NOVA*te** is started up. It will also be entered when:

- a HALT instruction is encountered
- an unsupported instruction is encountered
- an unsupported I/O instruction is encountered and breaks upon unsupported I/O instructions has been requested
- an internal exceptional condition is found
- a console interrupt is hit in VC mode

By default, the Virtual Console will be started if no initial RDOS program is specified.

-vcinput=<*filename*>

Define an initial Virtual Console command input file to be executed when the system is started.

-vmem=<*number*>

The integer <number> determines how much "extended memory" will be made available to RDOS user programs for remap purposes. This number should be in the range of 1 - 2048, or else a fatal error message is displayed and the system terminated.

The default is zero - no extended memory remapping area will be available

-wait

Wait for user console keyboard confirmation before terminating session.

-WDCdebug=<*boolean*>

Display debug information concerning WDC disk I/O operations on the console.

-WDCdevice=<*integer*>

Specify the starting disk device code. This should be a decimal number in the range of 0 - 60. The default is <050> octal, 40 decimal.

-WDCdiscutil=<*boolean*>

File used by the disk controller is in IRIS/BITS “discutil” format.

-WDCgeometry=<*integer*>

Specify the exact WDC disk geometry using an integer which has the format CCCCHHHSSS, where CCCC is the number of cylinders, HHH is the number of heads and SSS is the number of sectors per track. The default is 0589007032, meaning 589 cylinders, 7 heads and 32 sectors/track (all numbers are decimal).

-WDCnative=<*boolean*>

This command line option allows use of the native operating system's byte endian format for the disk file associated with WDC units. Strobe Data Falcon and Hawk board container files use native format on PCs, so supports direct use these files without modifications.

The default is FALSE, meaning that the Data General big-endian format is default for WDC "container" file.

-WDCoffset=<*integer*>

Specify the first disk sector offset into the raw disk file (the "container" file) for the WDC device. This is 0 by default, but may be set to 512 to directly use BITS 'diskutils' backfiles directly without conversion.

New command line option allows specification of initial

default console window size on Win/32 system;

-windowsize=<integer>

Specify the initial default console window size on Win/32 systems. Argument <integer> is the <rows> * 1000 + <columns> (3 digits).

-wrlmask=<*option*>

The masking of characters during write line (.WRL) operations is controlled by this option. The two valid values are:

7bits	mask all characters to 7 bits.
8bits	mask all characters to 8 bits.

The default is **-wrlmask=7bits -** mask all characters to 7 bits. This emulates DG write line file operations, and eliminates problems with non-DG systems that support 8-bit character sets. This will affect all write line (.WRL) operations requested, whether to disk files or devices.

Startup Error Messages

Re*NOVA*te displays various status information when it starts up to indicate normal progress, and indicate any error situation. Normal startup output contains a heading of the type and revision of **Re*NOVA*te** which is running, plus related legal copyright information.

A line is also displayed for each option file process, i.e. the renovate.ini file, to indicate what files have been found and processed, and where they were found. Most optional definition files are opened for only reading, and can therefore be set with the "read only" operating system permission or attribute. You must still have access and reading privileges to the file, along with access to all directories that exist in the pathname to the option file.

Errors may be found at any time during the startup process, and full information is displayed for you about the error condition. Most are "fatal", that is, they terminate the running of the **Re*NOVA*te** program and return control to the program, shell or shell script which invoked **Re*NOVA*te**. Most of the errors display one of the following messages and have the corresponding meaning:

I can't let you run *ReNOVAte*

The hardware device is not properly attached to the parallel port on DOS-like systems, or **Re*NOVA*te** has not been installed with the instrdossim (or install) software installation utility. This may also occur if you have several copies of **Re*NOVA*te** on your disk and are trying to run one that is not software installed, or has been moved after being software installed. If you move or copy a software-installed **Re*NOVA*te** system, it will need to be re-installed.

I can't open file file_name

The file file_name is required but can not be opened. This is usually caused by not having proper privileges to access the file, or all of the subdirectories in the pathname leading to the file.

file file_name is already in use

A required file is already in exclusive use by another user. The files privileges and your access rights should be verified for proper privileges.

I can't read file file_name

Re*NOVA*te could not read the contents of a file which was opened without error. This may indicate incorrect access permissions or privileges, or somebody changed the file attributes while **Re*NOVA*te** was using the file.

unknown command line option: option

There is no such command line option as option. Normally this results from incorrect spelling of a valid command line option.

duplicate command line option: option

The command line option 'option' was specified twice in the command line. This error is designed to let you know when duplicate options are specified, since experience has shown that this is not what is normally intended.

unknown or illegal option argument: argument

argument in not a valid argument for the command line option specified. In most cases the valid arguments are displayed after this error message so you can determine the proper argument and/or spelling to use.

conflicting option argument: argument

The argument 'argument' conflicts with a previously-specified argument for the same listed command line option.

illegal number: number

The command line option was expecting a number, but found the illegal or malformed number shown instead.

not enough memory for specific_name area

Not enough main memory exists to allocate space for the required specific_name area. This mostly occurs on DOS network systems which do not have sufficient user space left for **Re*NOVA*te** execution. The amount of user space available is shown using the DOS CHKDSK (or MEM) command, and may be substantially different than the total memory you think is in the machine. Try to reduce the network software and operating system space requirements by removing any TSR routines, reducing the number of disk buffers specified in the CONFIG.SYS file to 6, and not invoking **Re*NOVA*te** from within a batch file.

undefined terminal type: term_name

The terminal specified was not found in the renovate.trm file. The default terminal type assumed is BIOS for most single-user and network DOS[-like] systems, and ANSI for all other systems. This is then overridden in the following order: the TERM environment variable, the RENOVATE_TERM environment variable and the -TERMINAL command line option. That is, the -TERMINAL option overrides the RENOVATE_TERM environment variable, which overrides the TERM environment variable.

This is not yet implemented.

too many initial programs: program_name

When **Re*NOVA*te** encounters a command line argument it does not recognize, it assumes that it is the name of the initial RDOS program to be executed. If two or more command line arguments are unrecognized, then this error is displayed along with the name of the second unknown argument since you can logically have only one initial RDOS program. Normally this error is a result of a misspelled command line option. The -program command line option eliminates any ambiguity about what command line argument is intended to be the initial program.

bad entry in file : entry_line

Many option files, such as the renovate.trm terminal definition file, the renovate.log logical name definition file, etc. require sophisticated processing of the contained information. If a syntax or other such error is encountered while ReNOVAte is processing a file's contents, then an appropriate specific error message is display along with this general error message. The most frequent cause is invalid syntax or misspelled required words in the entry listed.

unknown startup error

An unknown startup error has occurred. Please contact your software supplier or Wild Hare.

Startup Error Status Codes

Corresponding to the initial startup error messages are status codes returned to the program invoking the **Re*NOVA*te**. These status codes are:

Mnemonic	Value	Description
ERR_OK	1	OK return [all systems]
ERR_NO_JOY	2	RDOS/Sim not properly installed or no hardware device present
ERR_OPEN	3	can't open required file
ERR_ACCESS	4	required file already in use
ERR_READ	5	error reading required file
ERR_UNKNOWN	6	unknown command line option
ERR_DUP	7	duplicate command line option
ERR_OPT	8	unknown or illegal option
ERR_CONFLICT	9	conflicting command line option
ERR_NUMBER	10	illegal option number
ERR_MEM	11	not enough memory for item
ERR_CRT	12	Unknown CRT type
ERR_PROG	13	too many initial programs
ERR_LOAD	14	unknown error
ERR_LINE	15	bad entry in file
ERR_WHAT	16	unknown startup error

Initialization error code
Table 3-1

Mnemonic	Value	Description
ERR_OK	1	OK return [all systems]
ERR_TIMEOUT	17	Timeout abort
ERR_FATAL	18	fatal runtime error encountered
ERR_PANIC	19	Internal system "panic"
ERR_SIGNAL	20	Some signal interrupted process
ERR_RDOS	21	Internal RDOS inconsistency/error
ERR_LAST	21	last pre-assigned return status

Runtime returned status codes
Table 3-2

The returned status code may be tested by the program that invoked **Re*NOVA*te**, and the calling program may perform different actions based on the result.

For example, a DOS-oriented batch script to run **Re*NOVA*te** could be:

```
RENOVATE
if ERRORLEVEL 1  GOTO  GOOD
echo “error running ReNOVAte”
pause
GOOD:
REM  continue on with batch file
```

A corresponding Bourne shell script on a typical UNIX system would be:

```
renovate
if [! $? -eq 1 ] then
    echo “error running ReNOVAte”
    sleep 5
fi
#  continue on with shell script
```

Chapter 4

Virtual Console

Overview

Re*NOVA*te provides a facility that emulates the “virtual console" (VC) supported by later model DG Nova and Eclipse computers. The virtual console permits you to perform traditional control panel functions and various extended operations through means of a simple terminal keyboard and display interface.

While similar to the corresponding DG VC capabilities, an extended set of capabilities is also supported. These capabilities are a superset of the traditional DG VC functions, and allow fine tuning, monitoring and control of the entire virtual machine from the lowest to highest level

The basic functions of examining and modifying memory locations and registers are supported. In addition, display and control of various aspects of the **Re*NOVA*te** environment, including devices and file assignments, can be accomplished through the VC.

The VC normally takes its input from the standard input file and displays its results on the standard output file, both of which are normally assigned to a user console. However, commands may be taken from special command files which provides a very powerful facility to handle unusual requirements.

User program input/output activities are suspended when the virtual console is active. The underlying operating system may may buffer user data for later processing depending upon specific system settings and default handling – but this is action operating system dependent.

Entering the Virtual Console

By default, the Virtual Console will be started if no initial RDOS program is specified. In addition, the

-VC

command line option can be used to force entering the Virtual Console upon system startup.

If VC support is specified, it will be automatically entered when **Re*NOVA*te** is started up. It will also be entered when:

- a HALT instruction is encountered
- an unsupported instruction is encountered
- an unsupported I/O instruction is encountered and breaks upon unsupported I/O instructions has been requested
- an internal exceptional condition is found
- a console interrupt is hit in VC mode

Whenever the VC is entered the entry reason is displayed, optionally followed by the significant CPU registers.

```
<<system startup>>
!
```

or

```
<<HALT instruction encountered>>
pc: 000001   accs: 000000 000000 000013 000010  1
!
```

If the VC is entered due to an exceptional status, an appropriate message will be displayed just before the "virtual console entered" message.

The program counter (pc), followed by the accumulators and the carry bit are then displayed in octal.

The "! " prompt indicates that the Virtual Console is ready to receive command line input, and is the same prompt as a DG system with no user map enabled.

If an invalid or unrecognized command or digit is typed, the error indicator

```
?
```

is displayed, the command is ignored, and the VC input ready prompt is again displayed.

Virtual Console Prompt

The virtual console prompt is displayed whenever command input is requested for processing. The prompt is a '!' (exclamation point) optionally preceded by a character that indicates the current state of the memory map unit:

!	the system is unmapped, or the program map is not enabled
A!	user program map A is active
B!	user program map A is active
C!	user program map A is active
D!	user program map A is active

Unmapped systems of all flavours will display simple the '!" character.

Memory Mapped Nova 3 and Nova 4 systems may support unmapped, map A or map B program map states. Some special versions of **Re*NOVA*te** may support a custom program Map C for Nova 3 or 4 style processors.

Eclipse systems may support unmapped, map A, map B, map C and map D states.

Further memory map display and control information is described later in this chapter.

Programs running in **Re*NOVA*te**'s "RDOS mode" will display a '!' prompt since RDOS user programs only "know" about logical addresses, not physical memory map addresses.

Command format

The **Re*NOVA*te** product supports three main categories of virtual console commands:

1. cell-oriented commands
2. single-keystroke commands that are similar to Data General functions; and
3. an extended set of commands that allow complete control of the environment.

"cell"-oriented commands allow convenient display, and optional modification, of user program memory and various system registers.

A single-keystroke DG-style command normally consists of a single character, and some commands may be preceded by an numeric “expression”. The character may be upper or lower case - they will be treated identically.

The extended set of commands consist of a command optionally followed by one or more parameters, as required by the specific command.

Numbers used as memory addresses may be up to will normally be truncated at 15 bits, corresponding to the standard 32Kword address space.

Fairchild 9445 64K word memory address space will use all 16 bits of a number if in 64K word mode.

Numbers used as data are truncated to 16 bits.

Cells

Several virtual console actions operate on “cells”. “Cells” are either program memory locations (memory cell), floating point registers (FPU cell) or an internal CPU registers (internal cell). All types of cells are examined and modified using similar techniques which are intended to provide quick and convenient manipulation.

Cell types

Memory Cells

Whenever a program memory location, or cell, is to be examined or modified, it must first be opened. Opening a memory location causes its contents to be printed in the current default display format – which is initially octal. Memory locations are opened by typing in the memory location number followed by the ‘/’ character.

The location may also be displayed in other formats, including decimal, hexadecimal, two octal bytes, two hexadecimal bytes, two ASCII bytes, byte pointer, RDOS system call format or assembly language dissembly, as described later.

Mapped systems with program maps enabled will display a character to the left of the logical address indicating which user map is in effect. The letter ‘A’ corresponds to program map ‘A’, ‘B’ corresponds to program map ‘B’, etc.

FPU Cells

Nova processors may support two FPU registers – FPAC and FTMP.

Eclipse processors may support four FPU registers – FPAC0 – FPAC3.

Each FPU register is considered a cell, with FPU cell 0 corresponding to the Nova FPAC or Eclipse FPAC0.

FACs may be displayed by cyling forward or backward by using the <lf> (<nl>) or '^' keys; 2 FPACs are defined for Nova CPUs and 4 FPACs are defined for Eclipse CPUs as previously described.

Internal Cells

"Internal cells" may be examined and modified in the same fashion as program memory locations, except the 'A' character is used to indicate these are internal cells instead of the '/' character indicating a program memory location. These special registers include the accumulators, the instruction match register, the program counter match (i.e. breakpoint) register, memory modified register, and others

If no number is specified as a prefixed argument, the current carry and accumulator values are displayed. If a valid internal cell number is specified before the 'A' is typed, the appropriate cell is opened.

Loc	Contents
0	AC0
1	AC1
2	AC2
3	AC3
4	Program counter
5	Carry in B15 (LSB)
6	Stack pointer
7	Frame pointer
10	Interrupt on flag in B15 (LSB)
11	Memory Map Unit status
12	Computer console switch register
13	Floating point status register
14	Floating point program counter
15	Current memory mask
16	Pending MMU status
17	Interrupt mask (MSKO result)
20	Program map state (0 = unmapped, 1 = map A, 2 = map B, etc.)
21	Data channel map state (0 = unmapped, 1 = map A, 2 = map B, etc.)
22	Mux IOCB pointer [Point 4 Mark 2/3/4

Loc	Contents
	only]
23	disk IOCB pointer [Point 4 Mark 2/3/4 only]
20	Tape/floppy IOCB pointer [Point 4 Mark 2/3/4 only]
24	DMA enable flag [Point 4 only]

internal cell numbers

Note that the Eclipse Stack Pointer is automatically transferred to/from memory location 040(8) upon Virtual Console entry/exit, as is the Frame pointer memory location 041(8) and the corresponding internal cells.

The Nova 3, 4 and Fairchild Stack Pointer and Frame Pointer hardware registers are accessed through the corresponding internal cells. These internal cells are ignored on the Nova processor.

Cell commands

A cell must be "opened" to be examined or modified. Opening a cell causes its address and contents to be displayed on the console.

Cells may be displayed in different formats, with one user-determined format always being the default format. The default format is initially 6 octal digits. Additional display formats are described later.

Command	Description
<*expr*>A	open the internal cell number <*expr*>
<*expr*>	open the program memory location <*expr*>
<carriage return>	close the current cell – do not open another
<line feed>	close the current cell and open the next cell
^	close the current cell and open the previous cell
@	close the current cell and open the cell whose corresponding to the contents of the current cell

Cell open commands

Expressions

An “expression” may be used anyplace a number is required for input, and can consist of an octal number, decimal number, hexadecimal number or double characters plus or minus another expression.

Expressions take the form of:

Expression := *number* [{ + | - } *expression*]…

number := *octal_number*
decimal_number
hexadecimal_number
“<char1><char2>

decimal_number := digits… .

hexadecimal_number := 0x<hexdigits>

All values are truncated to 16 bits.

Cell display formats

“Cells” are either memory locations (memory cell), floating point registers (FPU cell) or an internal CPU register (internal cell), and internal and register cells may examined and modified through the virtual console. These cells may be displayed in different formats, with one user-determined format always being the default format. The default format is initially 6 octal digits. Additional display formats supported are shown in the accompanying table. Floating point registers are always displayed in a fixed format described later.

Memory Cells and Internal Cells

The current default cell display may be changed by typing the ‘$’ (dollar sign) or ‘<esc>’ (escape key, octal 033) character followed by one of the display format indicator characters.

Char	Display Format
!	16 digit binary word
=	6 digit octal word
%	5 digit decimal word
$	4 digit hexadecimal word
_	Two octal bytes
|	Two decimal bytes
:	Two hexadecimal bytes
‘	Two ASCII bytes
&	Byte pointer
;	Symbolic instruction

Memory and Internal Cell display formats

Floating Point Cells

Floating Point Cells are opened using the ‘**F**’ command, and are currently display only in the following format which is patterned after the Data General format:

000000F + 053 5F BF4021 AAB55290

The first number is the floating point cell number, and the trailing ‘F’ indicates that the cell is a floating point register.

Cell 0 on a Nova is the FPAC and cell 1 is the FTMP.

Cell 0 on an Eclipse is FPAC0, with cell 3 corresponding to FPAC3.

A ‘-‘ or ‘+’ indicates the sign of the number.

The exponent is next displayed in *decimal* format. Note that this is in excess 64 representation as described in the appropriate DG literature.

The exponent is next displayed in hex format.

The final two series of hex numbers reflect the high order and low order part of the double precision mantissa., respectively.

A line feed will display the contents of the “next” floating point cell. For Nova type systems, this will display either the FPAC or FTMP, and Eclipse type systems will display the next of the four FPACs.

Likewise, an ‘^’ (up arrow) will display the “previous” floating point cell.

Symbolic display

Memory and internal cells may be displayed and have data entered in "symbolic" format. Symbolic representation of a cell is requested by using the ';' (semicolon) display format.

Symbolic input may be used instead of an expression to modify a cell's value. Symbols currently recognized for input and display output include:

- Basic Nova instruction set
- Nova 4 byte and stack instructions
- Nova floating point unit instructions
- Most RDOS system calls
- Common device names (i.e. TTO, TTI, etc.)

If symbolic input is not recognized, a "?" will be displayed and the entry will be ignored.

Cell input formats

Once a cell has been opened, its contents may be changed by typing in an expression whose value is to be placed in the cell. This expression should be terminated with a <carriage return>, <line feed> (<newline> on some keyboards), '@' or '^' (up arrow key).

Input may be specified in one of the formats previous described for output display.

Commands

The **Re*NOVA*te** product supports two main categories of virtual console commands:

4. single-keystroke commands that are similar to Data General functions; and
5. an extended set of commands that allow complete control of the environment.

DG-style commands

The following table describes the DG-style virtual console commands that are considered single-keystroke commands.

Either upper case or lower case letters may be used – they will be treated identically.

The "<esc>ape" key may be typed between an <expr>ession and the subsequent command letter; this is similar to some DG utilities and other third party command formats. The <esc>ape key is echoed as a dollar sign ('$').

Command	Command description
A	display value of all accumulators and the carry flag
<*expr*>**A**	open specified special cell
B	display all breakpoint values
<*expr*>**B**	set a breakpoint at location <*expr*>
D	delete all breakpoints
<*expr*>**D**	delete breakpoint <*expr*>
F	display all floating point registers
<*expr*>**F**	display floating point register <*expr*>
H	display instruction history buffer size
<*expr*>**H**	set instruction history buffer size to <*expr*> entries
K	"Kancel" current line (hey, it's DG compatible)
<*expr*>**L**	load bootstrap from device <*expr*>
M	display current active program and data channel map assignments
O	execute a single instruction and then return to virtual console
<*expr*>**O**	execute <*expr*> number of instructions and then return to virtual console
P	proceed from current program counter
Q	quit the current **Re*NOVA*te** session, return default status value 1 to calling program
<*expr*>**Q**	quit the current **Re*NOVA*te** session, return status value <*expr*> to calling program
R	run from the specified address
<*expr*>**X**	Display, and optionally modify, "delay" value for device <*expr*>
Z	perform instruction timing
/	open specified memory location
<CR>	close current cell location
<LF>	examine next cell location
^	examine previous cell location
<	redirect input from specified file (execute command file)

Command	Command description
>	redirect output to specified file (create trace file)
$	change default display format
<033>	change default display format
.	specify current memory location
?	display instruction summary information
!	execute a shell command line
<control>/A	display first instruction history buffer location (most recent instruction executed)
<control>/E	display last instruction history buffer location (oldest instruction executed)

DG-style Virtual Console Commands

Notes:

<*expr*> a numeric expression as described earlier in this chapter

Extended commands

In addition to the DG-similar single-character commands summarized in the previous section, a full set of commands exists to further control the system's environment, especially device and system context control.

Commands and optional arguments are separated on the command line by delimiter characters. Delimiter characters are:

','	comma
' '	space
\r	carriage return
\n	line feed (new line)
\f	form feed
\t	tab

For convenience, the extended commands may be abbreviated in most cases. That is, a *show history* command may be abbreviated as *sh hi*. An error message will be displayed if the command specified is ambiguous. Device or unit names may not be abbreviated, but register names may.

Some commands may require options or arguments that may contain spaces, commas, semicolons or other characters which would be considered delimiters. These arguments may be enclosed within double quote (") characters to allow embedded delimiters within an argument. The quote characters are removed before passed to the appropriate command handler. Embedded double quote characters are not currently allowed in a command line option or argument.

Multiple commands may be specified on a single line. Each command is separated from the others by a semi-colon (';'). A single command may not span multiple lines.

Extended Command	Description
attach { *<device_unit>* *<filename>* }…	attach operating system file to specified device, or unit of a multi-unit device
boot *<device>* [nostart]	boot from the specified device, do not automatically start program execution if the 'nostart' option is specified
detach *<device_unit>*…	detach operating system file from specified device, or unit of a multi-unit device
detach all	detach operating system file from all defined device units
disable *<device>*…	disable I/O instruction support for specified device
disable all	disable I/O instruction support for all devices
enable *<device>*…	enable I/O instruction support for specified device
enable all	enable I/O instruction support for all devices
help	display help summary
iorst [all]	send IORST to all devices
iorst *<device>*…	send IORST to specified *<device>*(s)
load memory *<filename>*	load memory from specified *<filename>*
quit [*<number>*]	exit session; optionally return value *<number>* to calling process
save memory *<filename>*	save memory into specified *<filename>*
set breakpoint *<integer>*…	set breakpoint at user memory location *<integer>*
set breakpoint -*<integer>*…	remove specific breakpoint number by using negative integer rather than positive
set breakpoint -all	remove all breakpoints
set device break { *<device_name>* | *<integer>* }	break to VC upon encountering I/O instruction for specified unsupported device
set device default { *<device_name>* | *<integer>* }	perform default I/O instruction handling upon encountering I/O instruction for specified unsupported device (controlled by CPU register DEVSTOP)

Extended Command	Description
set device ignore { *<device_name>* | *<integer>* }	ignore I/O instruction for specified unsupported device
set *<device_unit>* { *<parameter>* [*<value>*] }…	set *parameter* <*item*>[s] to <*value*>[s] for device *<device>*
set history *<integer>*	set execution history buffer to <integer> number of entries; previous entries are lost.
set option *<command_line_option>*	set command line option *<command_line_option>* as if it were specified on the command line
share *<device_unit1>* *<device_unit2>*	share data stream attached to *<dev_unit2>* with *<dev_unit1>*.
shell	start interactive session with native operating system shell or CLI; ; environment variable COMSPEC and SHELL are tested, in that order, for the name of the shell to execute.
shell *<program>* [*<argument>*…]	execute native operating system program or shell script along with optional argument(s).
show *<device>*	show unit information about the specified device
show *<device>* *<item>*…	show information about register *item*(s) for the specified device
show breakpoints	show all active breakpoints
show all	show information about all enabled devices.
show disabled	show information about all disabled devices.
show enabled	show information about all enabled devices (same as 'all')
show history	display instruction history information
show history *<integer>*	display instruction history information for the last *integer* instructions executed
show history info	display instruction history buffer size
show queue	display current event queue
show pmap [U | A | B | C | D | all]…	display current program map, or map corresponding to *<letter>* (mapped systems only)
show dmap [U | A | B | C | D | all]…	display current data channel map, or map corresponding to *<letter>* (mapped

Extended Command	Description
	systems only)
show time	display current event time
sleep <*integer*>…	delay command execution for <*integer*> seconds
step [<*count*>]	step and display <*count*> instructions, default is single instruction

Extended Virtual Console Commands

Notes:

<*device*>	I/O device name
<*device_unit*>	I/O device name for single-unit devices or device name and unit number for multi-unit devices
<*expr*>	a numeric expression as described earlier in this chapter
<*filename*>	represents any valid operating system file name
<*integer*>	may be an octal, decimal or hexadecimal non-negative integer
<*letter*>	corresponds to appropriate program or data channel map, must be upper or lower case A, B, C or D
<*value*>	represents a integer or a boolean ("true" or "false") value

Breakpoint Facility

Breakpoints may be placed anywhere in the user address space. Up to eight unique breakpoints may be set in the current version.

The virtual console will be entered when the program encounters a breakpoint during execution. The full capabilities of the virtual console may then be used as desired, and the program can be resumed transparent to user program execution. The instruction at the breakpoint location has not been executed when the virtual console is entered.

Setting breakpoints

Breakpoints may be set using either the single-character DG-compatible or extended format commands. To set a breakpoint DG-style, type:

<*expr*>B

The extended command is:

set breakpoint <*expr*>

The breakpoint will be set at the address specified by <expr>. An error message will be displayed if a breakpoint is already set at that user location, or if all possible breakpoints are already active.

The breakpoint currently is a user logical address, meaning that a user logical address may be valid in several map contexts.

The virtual console assigns numbers to breakpoints in descending order; that is, breakpoint number 7 is assigned first, then 6, etc. The unassigned breakpoint with the highest number is always assigned first.

Breakpoints may be set on addresses that are located in I/O areas since user address space is not modified.

Deleting breakpoints

A breakpoint is deleted by specifying the breakpoint number assigned to it as described in the previous section. The DG-style:

<*expr*>D

or extended command:

set breakpoint -<*expr*>

will delete the breakpoint <*expr*>. Note that the extended command format must precede the breakpoint number with a '-' (minus sign).

If no number is specified in the DG-style format, or the option **–all** is specified in the extended command format, then all breakpoints will be deleted.

Instruction stepping

One or more program instructions may be "stepped" through by using the DG-style 'O' command. A "trace" of each instruction executed will be displayed on the console, and control returned to the virtual console after the last instruction is executed.

As each instruction is executed, trace information consisting of the instruction address, active program map (if any), accumulators, carry and symbolic dissembly will be displayed.

Multiple instructions may be stepped without console intervention by preceding the 'O' command with a valid expression. For example, the following command steps 100 (octal) instructions:

100O

To step 25 (decimal) instructions, the command:

25.O

may be used.

Breakpoints will still be honored when single stepping.

Program resumption

The virtual console 'R' or 'P' DG-style commands may be used to resume program execution.

The 'P' (proceed) command continues program execution at the location specified in the program counter. This is reflected in internal cell 4.

The 'R' (run) command start program execution at location <*expr*>, which must be specified. This command does not cause any I/O or system reset. An error message will be displayed if no <*expr*> is given. This command has the same effect as setting internal cell 4 to the value <*expr*>, then issuing the 'P' command.

Program breakpoints will be active after either command is given.

Instruction History

User program instruction execution history may be optionally maintained to assist in program debugging, monitoring or control.

Keeping instruction history information decreases system performance, but only when the history system is active. When no history is active, no performance penalty is incurred.

Extended commands

Set history <*integer*> creates an instruction execution history buffer large enough to save information about the last <*integer*> instructions executed. Any current history buffer is

	released - its contents are lost – and a new buffer is created.
Show history info	displays information about the current history status and entry size.
Show history	displays the contents of the entire execution buffer, oldest instruction executed first.
Show history <*integer*>	displays the last <*integer*> instructions executed.

Each instruction entry displayed includes the instruction address, active program map (if any), accumulators, carry and symbolic dissembly.

DG commands

The 'H' DG-style virtual console command also controls this history, along with two other control codes.

The 'H' command without a leading numerical argument displays the current history option. If history is active, then the number of instructions save is displayed; if not active, and appropriate message is displayed.

The 'H' command with a leading number argument determines how many instructions will be saved in the history buffer. If history is already being maintained, the current history is lost and the new history buffer is created.

The <control>/E virtual console command displays the oldest instruction retained in the history sequence, and the <control>/A virtual console command displays the most recent (i.e. last) instruction executed. A <line feed> or <new line> will display the next instruction in the buffer, from oldest to newest instruction. An up arrow (^) will display the previous instruction in the buffer (i.e. one instruction older).

History retention may be disabled by specifying a 0 history buffer length.

Program Load

The <*expr*>L DG_style command or extended boot <*device*> command cause a program load from a given device. <*expr*> must evaluate to a number in the range of 0 – 63 (decimal) designating the device code of the device to be booted. <*device*> must be the device name of the device to be booted.

This has the same effect as performing a "high-speed" program load from a data channel device on a DG-style system, and is similar to the <*expr*>H DG virtual console command.

An appropriate error message is displayed if one of the following conditions is encountered:

- the specified device does not exist
- the device is not bootable
- no file is attached to the device (or unit 0 of a device which supports multiple units)
- an error occurs during the boot attempt

Upon successful boot, the console switch register (internal cell 012) is set to the device code of the boot device plus bit 0 (the high-order bit) set.

Also, AC0 will be set to the device code times 2. This undocumented secret “quirk” is required and assumed by some DG software and diagnostics. The other accumulator values are undefined.

The ‘L’ command will automatically start the user program after successful boot operation. The ‘boot *<device>*’ extended command will not automatically start user program execution; the ‘P’ command should be used to start the user program festivities.

Console Redirection

The '<' and '>' commands give powerful command input redirection and output redirection capabilities.

Output Redirection

Output may be directed to a specified file using the '>' console command. The '>' character is immediately followed with the name of the desired filename to receive display output.

If the output filename starts with a '>' character, then output will be appended to the designated file. This is a convenient method to create a log file.

If the filename does not start with a '>' character, then the file will be deleted if it already exists, then recreated.

The filename is logically translated before use by the special SimuLogics logical translation facility, so great flexibility and power is available to handle the most demanding situation.

Input Redirection

Input may be redirected from a specified file using the '>' console command. The '<' character must be immediately followed with the name of the desired filename which contains subsequent input. When all input from the specified file has been exhausted, then input control is returned to the input file issuing the '>' request.

Multiple levels of input redirection are supported, giving a flexible way of using common or generic command files to do different functions.

The filename is logically translated before use by the special SimuLogics logical translation facility, so great flexibility and power is available to handle the most demanding situation.

Comments

Comments may be embedded within input command files at any time by starting the comment text with a '#' character. All characters until the next '#' character, or the next end of line will be ignored. This provides a convenient method to document the purpose or assumptions of any give input line or part of input line.

Special input characters

In order to allow convenient creation of virtual console input files using any text editor on any system, standard end-of-line and special character handling is performed. Actual <carriage return> and <new line> (<line feed>) characters are ignored in input file; the character sequence "\r" and "\n" must be used it indicate the corresponding character.

Console interrupts

Programs may be interrupted at runtime with the control/C and control/break logical keystrokes.

On U*IX systems the control/break function is defined as the “quit” character definition for your terminal, the control/C is defined as the “intr” character definition.

On DOS and Windows systems, the control/break function is really the control/break, and the control/C is the control/C combination.

Hardware-level

When running in non-RDOS mode, the functions of each of these logical keys is identical: program execution is interrupted and control is returned to the Virtual Console.

On some systems there may exist two types of interrupts or signals: a BREAK and an INT. On these systems the BREAK signal or interrupt will return control to the virtual console. The INT signal will send a <control/C> to the user TTI input stream by default, unless overridden by a command line option.

RDOS level

When an RDOS program is running, any control/C interrupt is passed on to the RDOS program or system for handling, just as it is under a standard DG system. It is treated as a <control/C> console interrupt, but no break file is currently created.

By default, the control/break will be passed to the user program as a control/A console interrupt. However, if the –rdos=nobreak command line option is specified, the control/break will return to user to the virtual console instead. Program execution may be resumed with the 'P' or 'R' command, as desired.

Initial Command File

By default, **Re*NOVA*te** will search for a file called renovate.vc when the system starts up. This is assumed to contain Virtual Console commands to be executed before any initial user program of any type is run.

Alternately, you may override the default renovate.vc filename by using the command line option:

```
-VCINPUT=<filename>
```

to tell **Re*NOVA*te** to run execute the Virtual Console commands contained in <filename> before the first user program is started.

The comment facility is very helpful in documenting the contents of a VC command file. Any line that is not a continuation line that begins with a '#' or ';' character is ignored by the Virtual Console, and the next line is processed.

Input file line terminators are ignored; command files lines must be "explicitly terminated" by specifying a "\r" or "\n" wherever the corresponding <carriage return> or <new line> (<line feed>) character is to be located. This is described in the section

on “Console Redirection”, and provides a great deal of flexibility in creating the text file with virtually any type of editor.

Chapter 5

RDOS Environment

RDOS Environment Termination

RDOS environments may be terminated by issuing the command BYE, QUIT or EXIT from the top program level. If these commands are executed from a lower push level, then they act as a .RTN system call, or POP CLI command, to return to the next higher program level.

The system may also be terminated from the Virtual Console using the E (exit) command at any time.

RDOS Environment Special Functions

Enhancements to the RDOS environment have been added in a manner consistent with the standard RDOS command line interface. Each function in invoked with a special save file name, just like regular RDOS executable programs are run. These special functions and their corresponding file names are:

TRACERDOS - Start/Stop RDOS Call Tracing

Toggle the current state of RDOS system call tracing. Trace output is sent to the file defined by the -tracefile=<filename> command line option, or to the standard error (stderr) file.

VC - Invoke the Virtual Console

The Virtual Console is invoked. The RDOS program can be returned to by using the 'R' Virtual Console command.

SHELL - Call Native Shell

The native operating system's default shell program is entered. For DOS-type systems, this is COMMAND.COM by default; for UNIX-type systems this is the sh program. The command line may be used as it normally would, and the RDOS program resumed when the shell is exited.

BYE, EXIT, QUIT - Exit System

Any one of these commands pops up one program level, and exits the **Re*NOVA*te** system if the return is from equivalent program level 0.

ENVIRON – display directory environment

Display various directory information about the current session.

EXECBREAK – enable break when program load

This RDOS CLI-level command toggles the flag that tells the system to enter the Virtual Console whenver an RDOS program is loaded. Normally this is used only for special debugging situations.

SVCBREAK – enable break upon SVC instruction

This RDOS CLI-level command toggles the flag that tells the system to enter the Virtual Console whenver an RDOS program is loaded. Normally this is used only for special debugging situations.

When using these commands from the RDOS CLI prompt, remember that no ".SV" is needed on the program name. For example, to enter the Virtual Console, the following command line can be used:

-VC

Directory Path Features

Re*NOVA*te enhances the traditional RDOS environment with powerful directory search path features. Four options exist that may be used individually or in concert with each other are:

- programpath
- datapath
- path

- dirpath

Each is controlled by its own command line option that has the same name.

These features overcome the limitations of minimal native RDOS directory support, plus add user-directed "search path" support. The overall affect is a dramatic increase in convenience and flexibility for RDOS-based program development and application execution on open systems.

Each directory path list must be a valid native operating system path list for the directories to be searched, in the order they are to be searched, when specific RDOS files are accessed.

The "path" option is similar to the U*IX path concept, where all files opened will be searched for using the specified path list.

The "programpath" search list is used only for RDOS .EXEC and .OVOPN system calls, thereby effectively making this active only for RDOS program invocations.

The "datapath" search list is used for non-.EXEC, non-.OVOPN file accesses, making it complementary to the "programpath" option.

"dirpath" affects how .DIR requests are processed, with this path used to search for any single directory name. This is a very special capability that helps overcome some of the quirks of the RDOS directory search mechanism and its translation to open system concepts.

Note that if a path option is specified, the current directory is not searched by default. This behaviour is similar to U*IX, and unlike DOS or AOS/VS. The current directory can be searched by including it in the corresponding path or, more preferrably, having a "null path" specified in the search list.

When a program .EXEC or .OVOPN request is made, first any –programpath search list is used, then any –path search list is used. If neither list is specified, then the current directory is searched by default.

When a non-.EXEC or non-.OVOPN request is made, first any –datapath search list is used, then any –path search list is used. If neither list is specified, then the current directory is searched by default.

This makes it very convenient to logically segragate products into separate subdirectories, yet still use them much more conveniently under ReNOVAte than on an RDOS system. For example, if FORTRAN5 is contained in the /FORTRAN5 directory, ALGOL is in the /ALGOL directory, DGL is in the /DGL directory and all RDOS utilities are in the /RDOS directory on a separate drive called "G:", then these tools can be used by specifying the DOS command line option:

```
-programpath=;G:\FORTRAN5;G:\ALGOL;G:\DGL;G:\RDOS
```

Note that this specifies searching in the current directory first because a ‘;’ directly follows the ‘=’ command line argument separator, indicating a “null” directory name which means the current directory (just like under U*IX).

The corresponding U*IX command line option would have the directory names separated with the ‘:’ character, just as one would expect.

An ALGOL compile of a file in the current directory, whatever it is, would still have the proper tools found to perform the compile. Thus, maintaining separate directories for various user, system and application tools and programs can be easily set up and kept separate.

This technique allows very simple, clean development environments to be set up, used and maintained.

Logical Filename Translation Facility

Just like Wild Hare products, **Re*NOVA*te** supports a powerful "logical name translation" facility. This is a full-featured facility that provides text string translation, manipulation, substitution and conversion functions for a RDOS program and/or its environment. In its simplest form this capability allows a filename specified in a program to be translated to another filename for actual use by the operating system. This feature is frequently used to emulate the concept of a "link" file entry in an operating system-independent manner, but is actually much more powerful and flexible. It really allows any text string or substring to be translated, manipulated and/or converted to any other text string or substring, and even supports environment variable substitution and case control features.

What can it be used for?

This very powerful feature may be used for a variety of purposes, but it is especially suited for handling tough filename manipulation and alteration requirements without changing any programs. It lets you control many things without reprogramming, including:

- running RDOS programs that have hard-coded directory and/or filename specifiers in them
- changing parts of filenames
- logically linking one filename to another
- logically segregating user files in different directories
- controlling other ReNOVAte features (i.e. automatic spooling requests)
- creating a convenient interface to the operating system's native spooling system
- modifying file names with environment variable contents
- modifying file names based upon ReNOVAte or user status
- overriding device file type assumptions

How does it work?

The logical name facility performs all of this by combining the ability to do five things:

- translate one complete text string to another
- translate one substring to another
- substitute environment variables in a text string
- get/use internal **Re*NOVA*te** information in file names

- control and modify case sensitivity
- control translation recursion

It also acts recursively; that is, it will attempt to translate strings and substrings up to 20 times on a given translation before assuming a circular definition condition exists. This allows sophisticated use of string, substring and environment variables in addressing demanding requirements. However, complete recursion, string and case sensitivity is under user control if desired.

The translation file

A text file that is created and maintained by any text editor controls this feature. Each line in the file normally contains one logical text string to equivalence text string definition.

The desired name translations are entered in a text file as one or more lines, which can be created and edited by any source text editor. Each line entry normally contains two strings separated by one or more spaces and/or tabs. Usually these two strings correspond to file names which are to be used. The first name on the line is called the "logical" name, and the second name is the translated ("equivalence") name. If you simply wanted to translate one file name to another, the first line item would be the file name as specified by the RDOS program, and the second item would be the file name you wanted to pass to the operating system.

Re*NOVA*te automatically looks for a file named renovate.log to contain these logical name translations upon startup. The current directory, the directory from which **Re*NOVA*te** was started, and finally the choice subdirectory is searched for this file when the system is started.

full name translation

Re*NOVA*te will try to translate filenames both before and after any default suffixes are appended to the name (i.e. ".co", ".pd", ".dd", ".lst", etc). This makes it quite easy to set up powerful translation files. For example, if you wanted the source file COPY1 to be called COPYBOOK1 any time a RDOS program referenced file COPY1, the text file renovate.log would contain an entry with the following line:

```
COPY1          COPYBOOK1
```

The original name is the first name in a text line, and may be preceded with leading space or tab characters. The original name is followed by one or more, or a combination of, spaces or tab characters. The translated name then follows. Case sensitivity is assumed for UNIX and UNIX-like operating.

Note that in this case the full source string (or filename in this example) must match the full logical name in order for the translation to occur. This means that the name

COPY11, DP0F:COPY1 or COPY1toCOPY1 will not be affected by this definition entry.

A common use for this simple form of translation is to simply "redirect" when a given ISAM data file is expected to be found, regardless of the internal RDOS filename. If the RDOS program opens up a file named "INVOICES", but you know the file is on a DOS network drive "F", you could use the following definition:

INVOICES F:\INVOICES

This way the program does not need to be modified, yet you can setup and control file placement independent of program filenames.

This is also a good way to use a single, consistent RDOS ISAM file name have its data and index files on different disks or devices. For example, the RDOS ISAM file CUSTMAST could have its data file (XD) placed on a different disk than the index file (NX) with the following sample entries in an assumed DOS environment:

CUSTMAST.XD C:\DATAFILES\CUSTMAST.XD
CUSTMAST.NX F:\DATAFILES\CUSTMAST.NX

The NX file could even be placed on a RAM disk, or other such high-speed device, giving the system manager great flexibility in system configuration.

Note that **Re*NOVA*te** filenames have trailing space characters removed before they are subjected to translation. Embedded characters are retained in filenames where applicable.

substring substitution

But the logical name translation facility lets you do much more than just full string substitution. Besides replacing a full filename, you can tell it to only replace a substring within a filename by placing the original filename - which is now considered the substring to search for - in double quote characters. The equivalence name - if any is specified - comes after the logical name. This is a great way to change hard-coded directory specifiers within filenames. This means that any directory matching specifier in a user file name will be translated to the name given in the logical name file. For example, say the translation file contains the following entry:

"DEO:" /mydisk/

Any place the directory specifier DE0: is found in a RDOS file name it will be translated to /mydisk/, even if the DE0: is in the middle of the name somewhere. This is a simple way to eliminate directory specifiers in file names by simply changing them to "nothing" in the translation file. For example, if you wanted to eliminate all references to the DP0: directory in all filenames, the following entry which has only the item DP0: could be used:

```
"DP0:"
```

This will change all referenced of DP0: to "nothing" (no characters), but the rest of the file name will be passed through unchanged (assuming no other logical translations match it). For example, this would change the name DP0:SUBDIR:FILENAME to SUBDIR:FILENAME. Likewise, the file name DJ0:DP0:WHYME would be translated to DJ0:WHYME.

Note that **Re*NOVA*te** filenames have trailing space characters removed before they are subjected to translation. Embedded characters are retained in filenames where applicable.

By default, all strings and substrings will be translated up to twenty times. This means that a substring whose equivalence name includes the substring will be translated multiple times unless you inhibit recursion. This is done by prefixing the logical string with a '~', or '`' and is described in further detail later in this section.

Recursion control

If you defined the substring "/data/" to have the equivilence name "/usr/data/", it would translate any name that had an embedded "/data/" to "/usr/usr/usr/usr/usr/usr/.../usr/data/". To tell the system to only translate the substring with one pass through the defined logical translations, you would prefix the substring logical definition with the '~' character like this:

```
~"/data/"     /usr/data
```

quoted strings

A logical or equivalence string may be enclosed within single-quote characters ('). This permits embedded spaces to be placed within strings without interpretation. The single-quote characters are removed before the corresponding definition is made.

However, the destination string may also be enclosed in double quotes. When this is done, all characters within the double quotes are assumed to comprise the equivalence string. This allows spaces and tabs to be embedded in a translated string should that be desired. Note, however, that most operating systems do not allow filenames with embedded spaces, so this technique should be used with caution.

Double quotes may also be used to indicate a "null string", which may be more meaningful for human documentation than leaving an equivalence string definition missing. For example, the previous sample entry to change all DP0: substrings found in filenames to nothing could be better documented as:

"DP0:" ""

rather than

"DP0:"

Double quotes may exist within any logical or equivalence string by simply placing the character where desired. The entry:

"0:XXYYZZ" AB"CD

will translate the string DP0:XXYYZZ to DPAB"CD.

To provide maximum flexibility and simplify rules of use, no "quote matching" restrictions are placed on the logical or equivalence names. Therefore, the previous definition could also have been:

"0:XXYYZZ" "AB"CD"

environment variable substitution

Yet another translation option tells **Re*NOVA*te** to take the EQUIVILENCE name from an environment variable. This is specified by enclosing the EQUIVILENCE name (the second entry item) with percent sign ('%') characters around the name of the desired environment variable name to be substituted in the relative position the substitution is desired, with the rest of the item being the logical environment variable whose value should be used as the translated (EQUIVILENT) name.

As an example, assume an environment variable named COPY_DIR exists which is set to the value /SRC/COPYFILES/, and a logical name file entry exists like this:

"DE0:" %COPY_DIR%

Then the file name DE0:COPY1.CB would be translated to /SRC/COPYFILES/COPY1.CB. If no matching environment variable exists, then the substitution will result in no characters being entered in the corresponding EQUIVILENCE name position.

The environment variable substitution may comprise only part of the total substituted name. If we wanted to include a substituted name in a larger substitution request, the previous example would look like this:

"DE0:" %COPY_DIR%DE0/

This would translate the previous example's logical name to /SRC/COPYFILES/DE0/COPY1.CB.

Note that the previous example was set up as a substring substitution; that is, the substitution will take place anywhere a "DE0:" is found in a name.

internal variable substitution

Yet another translation option tells **Re*NOVA*te** to take the EQUIVILENCE name from an internal **Re*NOVA*te** parameter or status value. Just like the environment variable substitution previously described this is specified by enclosing the EQUIVILENCE name (the second entry item) with percent sign ('%') characters around the name of the desired internal variable name to be substituted in the relative position the substitution is desired. The current special names are:

None in the current version

character case sensitivity

File name testing is case sensitive by default, meaning that it matches the conventions of the supporting operating system. That is, the file name AbCdEf is different than ABCDEF.

Note that VAX/VMS, AOS/VS and DOS filenames are usually case insensitive. "UNIX-like" operating systems (i.e. UNIX, XENIX, AIX, DG/UX, HP/UX, etc.) and some of the advanced file systems of new PC operating systems are case sensitive.

case sensitivity overrides

Case sensitivity may be unconditionally overridden by prefixing the appropriate LOGICAL name (the first item in an entry) with the '-' character. Case sensitivity may be unconditionally enforced by prefixing the appropriate LOGICAL name with the '=' character. For example, if the logical name translation entry:

-ThisIsATest mytest

definition overrides

were specified, the following names would be converted to mytest:

THISISATEST
ThisIsATEst
thisisatest

Likewise, if the previous example had an '=' sign as the first character rather than the '-' sign, only file name ThisIsATest would match and be translated to mytest.

The case-definition character must precede any double-quote that may indicate a substring definition. This means to define an entry that would change any substring in any character case of "DP0:" to /home/data, you could use the following entry:

-"DP0:" /home/data

command line overrides

Two command line options let you override all translation case sensitivity testing without explicit specification within a definition entry. The -logoption=case and -logoption=nocase command line options force all testing to be case sensitive and case insensitive, respectively. These command line options do not override any entry with explicit case sensitivity specification.

Note that these command line overrides are not yet implemented .

Recursion control

By default, all strings and substrings will be translated up to twenty times through all logical definitions, as long as translations are performed. That is, if a given string or substring has no translation performed by the logical definitions, then no more passes through the translations will be performed. This means that a substring whose equivalence name includes the substring will be translated multiple times unless you inhibit recursion.

within all definitions

There are two types areas "recursion" can be controlled. When a string or substring is translated, it is compared with all of the logical definitions. This occurs even if a match occurs and the original string is modified. In some cases continued translating beyond the first match is not desired. This is specified by prefixing the logical string with a '~', and tells **Re*NOVA*te** to not look at any remaining logical translation definitions after a match has been made.

within a line

If a logical definition is a substring, then all occurrences of that definition's equivalence substring in the original test string will normally be translated to the equivalence string. This testing is done from left to right in the string. For example, the substring definition

"cat" "dog"

would translate the original string

he saw the cat run into the cat's house

to

he saw the dog run into the dog's house

If you wanted only the first match within the line to be made and eliminate further testing, the backwards single apostrophe is used:

`"cat" "dog"

This definition would change the preceding example to

he saw the dog run into the cat's house

Logical name translation file

Re*NOVA*te will automatically look for the logical name translation file renovate.log in the current directory when **Re*NOVA*te** starts up. If not found, then the directory from which **Re*NOVA*te** was started is searched, and finally the choice subdirectory is searched for this file. If no file is found in any of these directories, then no translation will be performed.

This default file name may be overridden by setting the environment variable RENOVATE_LOG equal to the full filename of the desired translation file.

Example file

Each standard **Re*NOVA*te** release comes with a sample logical translation file name s_renovate.log. This file would have to be renamed to renovate.log to be recognized by the system when it was started. A sample of this file is provided below:

```
!=========================================================================
!                            R E N O V A T E . L O G                      
!=========================================================================
!
! This is the optional ReNOVAte LOGical translation file which
! allows one file name to be translated to another name.  This file is
! expressly designed to allow definitions for the whole system.
!
! a sample redirection of a printer file name...
!

MYPRINTER     $LPT
!
! now all TEST program references get translated on a DOS system...
!
TTEST         \cobol\whtest
!
```

```
! fancy RDOS directory translation is done with the following DOS-oriented
! entry:
!

DS0:          c:\subdir\
!
!
# sample definition to change any  string that has and embedded "/usr/" to
# "/home/usr/".  Notice how we must tell ReNOVAte to translate any such

# string only once rather than recursively translating...

~"/usr/"         /home/usr/

!=====================================================================
!                          end of renovate.log
!=====================================================================
```

Command line options

The logical name translation facility is controlled by the -logfile=<filename> and -logoptions=<option> command line options.

The -logfile=<filename> overrides the default choice.log file name name and search procedure.

The -logoptions=<option> control the ability to unconditionally force logical name translations to conform to a specific case sensitivity: forcecase means honor case sensitivity, forcenocase means no case sensitivity and forcedefault means use the "default" case sensitivity rules, which is currently case sensitivity. Note that individual case control override of an entry takes precedence over these options.

Summary

Here is a quick summary of the logical name translation facility:

- facility called whenever a user program or data file is opened
- optionally called for CLI calls
- default file name is renovate.log, but may be overridden
- log file is searched for in: current directory, then the directory in which **Re*NOVA*te** is started
- each log file entry text line contains logical name and optional equivalence name
- comment lines may start with '#', ';' or '!' character
- blank lines are permitted
- each logical or equivalence string may normally be up to 256 characters long

- full string matching by not using double-quote character
- partial string matching (substring) by using quotes (double-quote character: ")
- recursion controlled by '~' and '`' characters: '~' stops at current line, '`' stops at first match in line
- optional case control specified by leading logical name character: '=' means force case sensitive comparison, '-' means no case sensitivity
- environment variable substitution may be specified by enclosing the appropriate name with '%' characters
- special internal variable values may be substituted just like environment variables, also enclosed by '%'s
- recursion, case and substring control may all be combined in one entry, with definition characters existing in that order.

File Name Processing

Every file used by the **Re*NOVA*te** runtime gets processed before being passed to the underlying operating system. During this processing the the file name may be modified and the file type completely changed.

File Types

Re*NOVA*te may modify file handling depending upon the the type of file being manipulated. Each file opened may be considered one of four types:

- disk files
- tty devices
- printer devices
- pipes

Name Translations

The file name may be modified by string, substring and/or environment variable substitution. Each file name gets processed in the following fashion:

1) if file type is already known, then exit

2) strip leading and trailing spaces + quotes

3) if string size >= STRING_SIZE (i.e. 255 bytes)
exit without translation

4) truncate the name down to STRING_SIZE characters

5) perform filename logical translation
(and translate environment variables in form of ${…})

6) if name starts with "{x}", check for special
type designator F/P/T

if found

a) set tipe to corresponding type

b) dequote and clean remaining name

c) return to caller

7) if name is "/dev/tty" (case insensitive)
force to lower case
set type to console and return

8) if name starts with "/dev/tty" or "/dev/lp" or
"/dev/" (case insensitive)
force to lower case

9) set type to printer and return

10) if file is a special OS file name
return type to caller

11) if FN_ALL_UPPER specified
convert filename to uppercase

12) if FN_ALL_LOWER specified
convert filename to lower case

13) if FN_NO_DOLLAR specified
convert all '$' characters in name to '_'

14) if FN_NO_HYPHEN specified
convert all '-' characters to '_'

15) if FN_NO_SPACES specified
convert all ' ' characters to '_'

16) if name matches a special AOS[/VS] name (*), then
a) set name to corresponding name
b) set type to corresponding type
c) return to caller

17) if name is in form of RDOS "QTY:x[y]"
a) modify name to form of "qtyx[y]" (lower case)
b) set type to printer
c) return to caller

18) if name is in form of AOS[/VS] "@CONx[y[z]]"
a) modify name to form of "conx[y[z]]" (lc)
b) set type to printer
c) return to caller

19) if ! FN_COLONS
replace all ':' in file name with '/', except for
initial "x:" that is assumed to be DOS disk names
if FN_DOS specified

20) check for special RDOS filenames (+) (even after
any final ':' characters)

21) if FN_UPPER specified, convert BASE filename to upper case

22) if FN_LOWER specified, convert BASE filename to lower case
change any except last '.' in name (NOT dirs) to '_'.

23) if FN_TRUNC_8 (8-byte truncation)
truncate file to 8/3 characters

24) if FN_RDOS (10-byte truncation)
truncate file to 10/2 characters

25) if FN_DOS (special WH filename DOS handling)
do special WH file name translation

26) return to caller, filename is assumed a disk file

notes:

(*) AOS[/VS] names include: @CONSOLE, @INPUT, @OUTPUT, @DATA, @NULL, @LPT and @LPT1

(+) RDOS names include: $TTI, $TTO, $TTI1, $TTO1, $LPT and $LPT1

Special File Names

RDOS, ICOS and AOS[/VS] all have special file names which are given to hardware devices and files. This simplifies application programming in many cases since these devices may be manipulated just like any disk file. Fortunately, most other operating systems on which **Re*NOVA*te** runs follows a similar philosophy. **Re*NOVA*te** permits your RDOS programs which directly reference these special ICOS/RDOS/AOS devices by translating the file names to a corresponding target operating system equivalent. This allows you to run programs which are normally tied to a specific DG system on other operating systems, all without changing program file name references.

Re*NOVA*te does this by translating each special Data General file name listed in table 6-3 and 6-4 to the corresponding file name listed under the appropriate operating system heading. For example, if an RDOS program running **Re*NOVA*te** under MS-DOS/PC-DOS opened up the file name "$TTI", that file name would automatically be translated by **Re*NOVA*te** to the file name "CON". If that same exact program were run under a UNIX system, the file "/dev/tty" would be opened.

There are also special pre-defined or "special" file names that have significance to Data General operating systems. These special files are automatically translated from the original DG filename to the target file name unless overridden by a user definition.

Original File Name	Translated Name			
	MS-DOS	UNIX	VMS	AOS/VS
@CONSOLE	CON	/dev/tty	Sys$Input	@CONSOLE
@INPUT	CON	/dev/tty	Sys$Input	@CONSOLE
@OUTPUT	CON	/dev/tty	Sys$Output	@DISPLAY
@DATA	CON	/dev/tty	Sys$Output	@DISPLAY
@NULL	NUL	/dev/null	Sys$Null	@NULL
@LPT	LPT1	/dev/lp	Sys$Printer	@LPT
@LPT1	LPT2	/dev/lp1	Sys$Printer1	@LPT1

Special AOS/VS file names
table 5-3

Original File Name	Translated Name			
	MS-DOS	UNIX	VMS	AOS/VS
$TTI	CON	/dev/tty	Sys$Input	@CONSOLE
$TTO	CON	/dev/tty	Sys$Output	@DISPLAY
$TTI1	CON	/dev/tty	Sys$Input	@CONSOLE
$TTO1	CON	/dev/tty	Sys$Output	@DISPLAY
$LPT	LPT1	/dev/lp	Sys$Printer	@LPT
$LPT1	LPT2	/dev/lp1	Sys$Printer1	@LPT1

Special RDOS file names
table 5-4

Extension Substitution

Here's the problem: RDOS and ICOS allow file names with up to 10 characters and an optional extension of 1 or 2 additional characters. MS-DOS, on the other hand, only allows file names of 8 characters and an extension of up to 3 characters. This could present a very large problem if your programs made use of the 10-character ICOS/RDOS file names.

Re*NOVA*te maintains file name compatibility so that you do NOT have to change any of your programs or data files. **Re*NOVA*te** does this by translating, or "folding", the ICOS/RDOS names which are larger than 11 characters in total size (name plus extension) into file names which MS-DOS can handle.

Briefly, RDOS/ICOS file names are used unmodified if they are 11 characters or less in length, including extension. The file name extension may be shifted left one character if needed. For example, the RDOS/ICOS name 'MYTESTFILE' would be translated to 'MYTESTFI.LE'. If the RDOS/ICOS file name is 12 characters long the final two characters will be translated to a special unique MS-DOS character according to Table 6-5. If the last two characters are not recognizable (i.e. not found in Table 6-5) then the first 9 characters will be retained and the final character will be a "!".

When you do a MS-DOS DIRectory list you may notice long file names with these somewhat unusual (at least to us DG fans) extensions. These will be the ones which have been translated according to the rules which you have just read.

In order to retain as many significant file name characters as possible under operating systems that do not retain as many characters as RDOS, the following name extension substitutions may apply:

RDOS Extension	Substituted Character
.XD	(
.NX	}
.PD	{
.DD	}
.CF	_ (underscore)

Extension Folding Characters
table 5-5

To maintain compatibility between all **Re*NOVA*te**-supported systems, this folding capability is available on all systems, not just MS-DOS based versions. This allows files to be translated from system to system without worrying about the underlying platform's capabilities.

DG Directory Emulation

RDOS, ICOS, AOS, UNIX and MS-DOS (2.0 and later) all support the concept of file directories. Due to the differences in the concepts, however, several things should be noted:

- the MS-DOS directory specifier delimiter is a '\' (backslash)
- MS-DOS supports multiple subdirectory levels
- ICOS/RDOS device names are 'D<letter><number>:'
- MS-DOS device names are '<letter>:'
- no specifier name length checking is performed by **Re*NOVA*te**

The Data General operating systems use the colon (':') character to separate directory specifiers from their corresponding file names, whereas MS-DOS 2.0 and later uses the backslash ('\') character. **Re*NOVA*te** therefore translates all colon characters in each user filename to the backslash character, with certain exceptions shortly listed.

MS-DOS supports multiple subdirectory levels whereas ICOS and RDOS normally only support one level. Unless explicitly specified, all MS-DOS subdirectory searches start from the current subdirectory. For example, if an ICOS or RDOS directory named PAYFILES were successfully initialized and an open attempt was made of the existing file 'PAYFILES:MYFILE', the file could be opened no matter from which subdirecotry (or subpartition) the attempt was made. Under MS-DOS, the file name would be translated to 'PAYFILES\MYFILE', and the open's success would definitely depend upon the current default directory. In this particular case, the 'PAYFILES' directory would have to be within the current default directory for the open to be successful.

ICOS and RDOS specifies each of its major disks with a consistent naming convention. If any one of these standard disk names is encountered as first part of a file name, the disk name is preceded with a '\'. This tells MS-DOS to start the directory search from the current disk's root directory. This allows emulation of the ICOS/RDOS environment by having a subdirectory off of the disk's root directory of 'D<letter><number>'. If the current default MS-DOS disk were 'C:', the default directory 'ACCOUNT', and a directory existed called 'DP0' off of the root directory, the filename 'DP0:TESTFILE' would translate to the MS-DOS filename '\DP0\TESTFILE', which would then properly reference the 'DP0' subdirectory.

Note that NO directory name length is checked, so that you must be certain that your directory names are 8 characters or less for MS-DOS.

MAP.DR and SYS.DR Emulation

Renovate creates a fake MAP.DR or SYS.DR file whenever an RDOS program attempts to reference one of these specific names. This allows RDOS programs to work properly, even if they rely on specific layout information in the SYS.DR/MAP.DR files.

When creating the SYS.DR file, file names of files existing in the current directory are added to the SYS.DR file for all files which follow the standard Data General RDOS 10.2 file name convention. File which have more than two (2) character extensions are ignore, as are files which have more than ten (10) characters in their base filename.

The MAP.DR file is created with a specific number of blocks in use and blocks free, and is version dependent.

Chapter 6
Hardware Environment

CPU Support

The following CPUs are supported, and must be specified when ordering the **Re*NOVA*te** software.

- Nova
- Nova 3 or 4, with or without MMPU support
- Fairchild 9445, with or without 64K word support
- Eclipse, with or without MMPU support

Current systems maintain internal statistics on instruction use for performance and optimization purposes. This information is sent to the **inststats.inf** file when using the **–statistics** command line option.

All machines support their corresponding hardware unsigned multiply and divide. All except the original Nova have signed multiply and divide as well.

The Nova 4 incorporates full stack, trap and byte access instructions. FPU device support is currently emulated using native hardware double-precision floating point (normally IEEE format on most modern processors).

The Fairchild 9445 optionally adds support for 64Kwords of main memory, which is controlled by the special Fairchild I/O instructions to device code 001.

The Eclipse supports the Character Instruction Set.

All systems assume 32Kwords of addressable user memory, except the 64Kword memory option for the F9445. The optional MMU support for the Nova 3 and Nova 4 may handle up to 256 K words, and the optional Eclipse MMU support may handle up to 2048 K words.

Device Support

The following devices can be supported in a typical **Re*NOVA*te** system:

- CPU
  - UMUL/UDIV
  - SMUL/SDIV
  - CIS
  - WCS
  - FPU
  - MMU
  - PAR
  - TTI
  - TTO
  - RTC
  - PIT
  - DSK
  - DKB
  - DKP
  - MTA
  - QTY
  - PTR
  - PTP
  - PTP1
  - SMD
  - STA
  - TTI1
  - TTO1
  - LPT
  - LPT1
  - HawkDOS
  - HawkDSF

By default, only the RTC, TTI, TTO, FPU and PIT are active without initialization from the Virtual Console or an initialization file. Other devices must have a native operating system file associated with the device using the "YA" virtual console command.

The RTC driver will attempt to maintain an approximate interrupt frequency base corresponding to the speed of the processor on which **Re*NOVA*te** runs. This allows single-tasking systems to maintain extremely close clock interrupt handling during shifting processor loads.

Device polling control

Each input device uses a "delay" value to determine how often **Re*NOVA*te** polls the corresponding device for input. This delay value is the number of instructions **Re*NOVA*te** executes between device polls, and dramatically affects the system overhead and responsiveness to any given device in any application environment.

The Virtual Console 'X' command examines, and optionally modifies, the delay value for any given device.

The absolute time between polls for any given numeric delay value depends upon the speed of the processor, the type of application and operating system in use. The 'V' command displays the approximate number of instructions executed per millisecond on any given **Re*NOVA*te** system.

User Procedure Facility

Custom user code may be implemented using the powerful **Re*NOVA*te** "User Procedure Facility".

This allows Nova and Eclipse programs to invoke native programs and procedures whenever specific I/O instructions are executed, or specific I/O device codes are referenced. The user code is given full capabilities to control **Re*NOVA*te** operation after routine execution.

The routines may be dynamically loaded under Windows 9x and Windows NT environments; the routines are linked with the **Re*NOVA*te** system libraries on all other platforms.

Complete system information is available to the user code.

This allows custom user code to be added with minimal impact or effort to legacy systems. It also promotes the use of one source and object program for use on all platforms, from the original Nova and Eclipse systems to **Re*NOVA*te** -supported open systems.

Contact SimuLogics for further information.

Chapter 7

Compatibility Notes

CPU Support

The following CPUs are supported, and must be specified when ordering the **Re*NOVA*te** software.

- Nova
- Nova 3 or 4, with or without MMPU support
- Fairchild 9445, with or without 64K word support
- Eclipse, with or without MMPU support

Current systems maintain internal statistics on instruction use for performance and optimization purposes. This information is sent to the **inststats.inf** file when using the **–statistics** command line option.

All machines support their corresponding hardware unsigned multiply and divide. All except the original Nova have signed multiply and divide as well.

The Nova 4 incorporates full stack, trap and byte access instructions. FPU device support is currently emulated using native hardware double-precision floating point (normally IEEE format on most modern processors).

The Fairchild 9445 optionally adds support for 64Kwords of main memory, which is controlled by the special Fairchild I/O instructions to device code 001.

The Eclipse supports the Character Instruction Set.

All systems assume 32Kwords of addressable user memory, except the 64Kword memory option for the F9445. The optional MMU support for the Nova 3 and Nova 4 may handle up to 256 K words, and the optional Eclipse MMU support may handle up to 2048 K words.

Note that the “cache memory” described by some vendors is really a form of high-speed peripheral, not an integral processor memory map scheme, and is not therefore reflected in the following table.

Vendor	CPU	mapped support	user memory	physical memory
Data General	Nova	unmapped	32 KW	32 KW
	Nova 3	unmapped	32 KW	32 KW
			64 KW	64 KW (custom option)
		mapped	32 KW	128 KW
				256 KW
				2048 KW (custom map)
	Nova 4	unmapped	32 KW	32 KW
			64 KW	64 KW (custom option)
		mapped	32 KW	128 KW
				256 KW
				2048 KW (custom map)
	Eclipse	unmapped	32 KW	32 KW
		mapped	32 KW	2048 KW (special request)
Strobe Data	Hawk Nova	unmapped	32 KW	32 KW
	Hawk Eclipse	unmapped	32 KW	32 KW
Point 4	Mark 2/3	unmapped	64 KW	64 KW
	Mark 4/4E	unmapped	64 KW	64 KW
	Mark 12	unmapped	64 KW	64 KW
Bytronix	60xx	unmapped	64 KW	64 KW
Fairchild	9445	unmapped	64 KW	64 KW

Device Support

The following devices can be supported in a typical **Re*NOVA*te** system:

- LMD
- FPU
- MUX
- MTA
- PIT

- PTR/PTR1
- PTP/PTP1
- RTC
- SMD
- STA
- TTI/TTI1
- TTO/TTO1
- DSK
- DKP

By default, only the RTC, TTI, TTO, FPU and PIT are active without initialization from the Virtual Console or an initialization file. Other devices must have a native operating system file associated with the device using the "YA" virtual console command.

The RTC driver will attempt to maintain an approximate interrupt frequency base corresponding to the speed of the processor on which **Re*NOVA*te** runs. This allows single-tasking systems to maintain extremely close clock interrupt handling during shifting processor loads.

Device polling control

Each input device uses a "delay" value to determine how often **Re*NOVA*te** polls the corresponding device for input. This delay value is the number of instructions **Re*NOVA*te** executes between device polls, and dramatically affects the system overhead and responsiveness to any given device in any application environment.

The Virtual Console 'X' command examines, and optionally modifies, the delay value for any given device.

The absolute time between polls for any given numeric delay value depends upon the speed of the processor, the type of application and operating system in use. The 'V' command displays the approximate number of instructions executed per millisecond on any given **Re*NOVA*te** system.

Chapter 8

Extending Re*NOVA*te

Extending Re*NOVA*te

Re*NOVA*te has many convenient yet powerful features that allow it to handle almost any legacy situation. However, there may be instances where it may be desired to enhance the system to:

- Transparently add new capabilities to legacy code
- Access native operating system facilities from within current applications
- Perform secure functions independent of current code
- Add new user-interface features to legacy applications
- Provide marketing distinctions between legacy- and **Re*NOVA*te**-based systems

These custom needs can be all addressed in a version-independent, modularized and powerful fashion in two ways:

1. All **Re*NOVA*te** systems allow users to write 'C' routines and link them with the standard product, and

2. In addition, Windows-based systems support dynamically-loaded link libraries (DLLs) to provide a powerful superset to the 'C' routine capabilities.

While two main implementation methods exist, they both provide similar capabilities. Also, not all features provided by each method need to be implemented by user routines.

The user routine(s) may be written in different languages, and can even interface with other subsystems, giving unprecedented power to legacy applications.

User Routine Functions

All **Re*NOVA*te**-to-user interfacing occurs through the calling of a single user routine. This routine is passed various information, including the "reason" the routine was called.

A selected user routine may be called in three situations:

1. when **Re*NOVA*te** is started;
2. when **Re*NOVA*te** is terminated,
3. when **Re*NOVA*te** is running user code and encounters specific I/O operations.

In each case the user routine may indicate the desired action taken when the user routine returns to its caller.

The user interface routine is always called for the first two situations – startup and termination.

The third situation may be called whenever a reference is made within a user program to a specific device code and/or specific I/O instruction to a specific device.

Other features may be added to the system, so the release notes should be consulted for the exact information corresponding to a particular product version.

Startup

The user routine is called when the **Re*NOVA*te** system is initially started, but before user code is executed. The user routine can then perform any required initialization and tell the system to:

1. Continue system execution
2. Enter the virtual console
3. Terminate system operation

Convenient definitions are provided to do this in a system- and version-independent fashion as shown later.

Termination

The user routine is notified when the **Re*NOVA*te** system is about to terminate for normal or abnormal reasons. The termination reason is indicated in a parameter passed to the routine.

The return code from the user routine is currently ignored by the caller.

Instruction routine

The user routine may be called whenever a specific I/O instruction is executed within the Nova or Eclipse program, or whenever a specific I/O device code is accessed from within the program.

The reason for the user routine call, plus information concerning the state of the Nova or Eclipse “machine”, is passed to the user routine.

Thus, user code has access to the full system state; it can be accessed and/or modified as desired by the user code.

The user code indicates what it wants **Re*NOVA*te** to do when the user code returns by passing an appropriate return code to the caller. The current actions are:

1. Continue system execution
2. Enter the virtual console
3. Terminate system operation

Convenient definitions are provided to do this in a system- and version-independent fashion as shown later.

The user routine is entered whenever an I/O device code is referenced and/or whenever a specific I/O instruction is seen. Both conditions may exist simultaneously, with the specific instruction having priority over the general device code reference.

The controlling device code and instruction are specified with two command line options:

-userdevice=<*integer*>

Device a specific I/O device to be considered the special "user device". Any I/O reference to this device may cause an external user 'C' routine to be executed, allowing convenient, powerful support for special applications. The <integer> argument must be in the range of 0- 63 (decimal), and must be an unused device code. This reference has lower priority than the -userinstruction argument.

-userinstruction=<*integer*>

Device a specific I/O instruction to be considered the special "user I/O instruction". Any use of this I/O instruction may cause an external user 'C' routine to be executed, allowing convenient, powerful support for special applications. The argument must specify a valid I/O instruction. This option is tested before the user device code option is checked, giving a very powerful feature for extending "legacy" applications with minimal effort.

Periodic invocation

A new feature allows user routines to be called based on some internal Renovate time base.

Linked User “C” Routines

All system support the capability to link user ‘C’-compatible routines with the **Re*NOVA*te** system.

The 'C'-compatible user routines need to be compiled with the same memory model, if applicable, and calling convention as the RE system. The resulting output code or library file then needs to be linked with SimuLogics provided **Re*NOVA*te** library files using the native operating system's object file linker program.

These capabilities are provided in the form of a "link kit", and are available upon special request.

Contact SimuLogics for further details.

Windows Dynamically-Loaded DLLs

Windows 95/98/NT systems provide an even more convenient capability to transparently interface with **Re*NOVA*te** without performing any static linking.

The user creates a DLL separate from the **Re*NOVA*te** system using whatever tools and/or languages desired. **Re*NOVA*te** then searches for and opens the appropriate DLL when the system starts up.

The default user library file name is "usercode.dll", and the default user routine name is "user_routine". These defaults may be overridden with the following command line options:

-userlibrary

Specify the external user library name which contains user 'C' code. Currently this overrides the default USERCODE.DLL library name used under Windows 9x and NT systems when supporting the special user interface feature.

-userroutine

Specify the external user routine name which contains user 'C' code executed whenever a special user instruction or device is referenced. Currently this overrides the default 'user_routine' library name used under Windows 9x and NT systems when supporting the special user interface feature.

Sample User Routines

The following sample user routine simply displays a message on the main console whenever the user 'C' routine is called. The reason for the call is displayed, then return is made to the caller with the "OK" status flag indicating the system should continue execution.

```
/*************************************************************************/
/*                                                                       */
/*                          U S E R C O D E . H                          */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*                 Copyright (C) 1997-1999  SimuLogics                   */
/*                   all rights reserved worldwide                       */
/*                                                                       */
/*     These coded instructions,  statements and  computer programs      */
/*     contain  unpublished proprietary  information of SimuLogics,      */
/*     and are protected by United States of America Copyright Law.      */
/*     They may NOT be  disclosed  to  third parties, or copied, or      */
/*     duplicated in any form, in whole or part,  without the prior      */
/*     written consent of SimuLogics.                                    */
/*                                                                       */
/*************************************************************************/


#if    ! defined(_USERCODE_H_)

#define_USERCODE_H_

#if    ! defined(_WHPORT_H_)

#defineS_word        unsigned short       /*  16-bit unsigned datum  */
#defineL_word        unsigned int         /*  32-bit unsigned datum  */

#define_WHPORT_H_

#endif /*  _WH_PORT_H_  */


       /*--------------------------------------*/
       /*        structure User_info           */
       /*--------------------------------------*/

typedef struct
       {
       S_word        ac0 ;
       S_word        ac1 ;
       S_word        ac2 ;
       S_word        ac3 ;
       S_word        carry ;
       S_word        pc ;
       S_word        sp ;
       S_word        fp ;
       S_word        ion ;
       S_word        mem_size ;
       S_word        mem_mask ;
       S_word        swr ;
       S_word        map_status ;
       S_word        map_new_status ;
       S_word        map_pnum ;
       S_word        map_dnum ;
       S_word        fpu_status ;
       S_word        fpu_pc ;
       L_word        fpac0_h ;
       L_word        fpac0_l ;
       L_word        fpac1_h ;
       L_word        fpac1_l ;
       L_word        fpac2_h ;
       L_word        fpac2_l ;
       L_word        fpac3_h ;
       L_word        fpac3_l ;
```

```
       /* ------------- */
       S_word        endian ;
       S_word        machine ;
       S_word        revision ;
       S_word        options ;
       S_word        mode ;
       S_word        RDOSmode ;
       }    User_info ;


              /*--------------------------------------*/
              /*     user routine entry reasons  */
              /*--------------------------------------*/

enum   {
       USER_INIT,                  /*  system startup          */
       USER_TERM,                  /*  system termination      */
       USER_RESET,                 /*  IORST instruction seen  */
       USER_INST,                  /*  user instruction seen   */
       USER_DEV,                   /*  user device reference seen    */
       USER_TICKS,
       USER_SECONDS,
       USER_VC_OK,
       }    User_entries ;


              /*--------------------------------------*/
              /*     user routine exit reasons   */
              /*--------------------------------------*/

enum   {
       USER_RET_OK,                /*  user routine OK exit    */
       USER_RET_VC,
       USER_RET_VC_EX,
       USER_RET_HALT,
       USER_RET_EXIT,
       USER_RET_TICK,
       USER_RET_SECOND,
       }    User_exits ;


       /*---------------------------------------------------------------*/
       /*                   global USERCODE prototypes        */
       /*---------------------------------------------------------------*/

typedefint (* User_routine ) ( int mode, S_word * memory, User_info * state,
char * cmd ) ;


#endif /*  _USERCODE_H_  */


/*****************************************************************/
/*                          end of USERCODE.H                    */
/*****************************************************************/
```

```
/*****************************************************************/
/*                                                               */
/*                        U S E R C O D E . C                    */
/*                                                               */
/*****************************************************************/
/*                                                               */
/*                  Copyright (C) 1985-1999  SimuLogics          */
/*                    all rights reserved worldwide                  */
/*                                                               */
```

```
/*     These coded instructions,  statements and  computer programs  */
/*     contain  unpublished proprietary  information of SimuLogics,  */
/*     and are protected by United States of America Copyright Law.  */
/*     They may NOT be  disclosed  to  third parties, or copied, or  */
/*     duplicated in any form, in whole or part,  without the prior  */
/*     written consent of SimuLogics.                                */
/*                                                                   */
/*********************************************************************/

#define_USERCODE_C_


#include <windows.h>
#include <stdlib.h>
#include <stdio.h>

#include "usercode.h"


#if    defined(_NOT_NEEDED_)

BOOL WINAPI DLLEntryPoint( HINSTANCE hinstDLL, DWORD fdwReason, LPVOID
lpvReserved )
       {
       printf( "\r\n DLLEntryPoint called \r\n" ) ;
       switch ( fdwReason )
              {
       case DLL_PROCESS_ATTACH :
              //  the DLL is being mapped into the process address space
              break ;

       case DLL_THREAD_ATTACH :
              //  a thread is being created
              break ;

       case DLL_THREAD_DETACH :
              //  a thread is exiting cleanly
              break ;

       case DLL_PROCESS_DETACH :
              //  the DLL is being unmapped from process address space
              break ;
              }
       return ( TRUE ) ;
       }      /*  end of 'DLLEntryPoint'  */

#endif /*  _NOT_NEEDED_  */


       /*===============================================================*/
       /*                          user_routine                         */
       /*===============================================================*/

int user_routine( int mode, S_word * memory, User_info * state, char * cmd )
       {
       switch ( mode )
              {
       case USER_INIT :     /*  system initialization  */
                            printf( "\r\nuser_routine INIT \r\n" ) ;
                            break ;

       case USER_TERM :     /*  system termination  */
                            printf( "\r\nuser_routine TERM \r\n" ) ;
                            break ;

       case USER_RESET :    /*  IORST encountered  */
```

```
                              printf( "\r\nuser_routine RESET [%06o] \r\n",
(unsigned) state->pc ) ;
                              break ;

       case USER_INST :       /*  user instruction seen  */
                              printf( "\r\nuser_routine INST [%06o] \r\n",
(unsigned) state->pc ) ;
                              break ;

       case USER_DEV :               /*  user device code referenced  */
                              printf( "\r\nuser_routine DEV [%06o] \r\n",
(unsigned) state->pc ) ;
                              break ;

       default :              /*  huh?  */
                              printf( "\r\nuser_routine huh? \r\n" ) ;
                              break ;
              }
       return ( USER_RET_OK ) ;
       }      /*  end of 'user_routine'  */


       /*=============================================================*/
       /*                        user_routine_alt                     */
       /*=============================================================*/

int user_routine_alt( int mode, S_word * memory, User_info * state, char * cmd )
       {
       /*-------------------------------------------------------------*/
       /*  Test for alternate entry point capability under command*/
       /*  line control (using -userroutine=<name> option)       */
       /*-------------------------------------------------------------*/

       printf( "\r\n<user_routine_alt>  " ) ;
       return( user_routine(mode,memory,state,cmd) ) ;
       }      /*  end of 'user_routine_alt'  */


/*************************************************************************/
/*                           end of USERCODE.C                           */
/*************************************************************************/
```

Appendix A

RDOS Call Summary

System Call	Description	Special Implementation Notes
.ABORT	abort a task	
.AKILL	kill all task of a given priority	
.APPEND	open a file for appending	Uses -datapath or -path argument
.ARDY	ready all task of a given priority	
.ASUSP	suspend all tasks of a given priority	
.BOOT	bootstrap a new system or stand-alone program	Uses -execpath or -path argument
.BREAK	interrupt current program and save memory state in a save file	'[f]break.sv' file is created
.CCONT	create contiguous file, initialize file to zeros	File is created at specified size, but may not be "contiguous" disk space due to native operating system constraints
.CDIR	create a subdirectory	
.CHATR	change file attributes	ignored
.CHLAT	change line access attributes	ignored
.CHSTS	get open file's status	
.CLOSE	close a file	
.CONN	create contiguous file, do **not** to zeros	File is created at specified size, but may not be "contiguous" disk space due to native operating system constraints
.CRAND	create a random file	
.CPART	create a secondary partition	Creates a subdirectory, but size is ignored due to native operating system constraints
.CREAT	create a sequential file	
.DDIS	disable user access to device	Ignored (device code must be in range of 0 – 63)
.DEBL	enable user access to device	Ignored (device code must be in range of 0 – 63)
.DELAY	delay execution of a task	delay precision may be dependent upon native operating system constraints
.DELET	delete a file	
.DIR	change current directory	Uses -dirpath argument
.DQTSK	dequeue a previously queued task	
.DRSCH	disable the task scheduler	
.DUCLK	define a user clock	

System Call	Description	Special Implementation Notes
.EOPEN	open file for exclusive access	Uses -datapath or -path argument
.EQIV	assign a temporary name to a device	returns error ERICM
.ERDB	extended memory read block	
.ERSCH	re-enable the task scheduler	
.ERTN	return from program with error code	"panics" system just like standard RDOS if issued from background level 0
.EWRB	extended memory write block	
.EXBG	checkpoint a background program	Uses -execpath or -path argument; Background not suspended if called from background, separate process started; Native program optionally called if RDOS program not found; Call of RDOS program yet implemented
.EXEC	swap or chain to a new program	Uses -execpath or -path argument
.EXFG	execute a program in the foreground	Uses -execpath or -path argument ; separate process started for program; Native program optionally called if RDOS program not found; Call of RDOS program yet implemented
.FGND	determine foreground program information	Results determined by -foreground option
.GCHAR	get console character	
.GCHN	get free channel number	
.GCIN	get input console name	
.GCOUT	get output console name	
.GDAY	get date	
.GDIR	get current directory name	Last 10 bytes of current directory returned (prefix stripped)
.GHRZ	get real time clock frequency	May be specified by -RDOS option
.GMCA	get current MCA unit number	returns error ERICM
.GPOS	get current file position	
.GSYS	get current operating system name	
.GTATR	get file attributes	
.GTOD	get time of day	
.ICMN	define inter-program communications area	Defined as disk file in current version
.IDEF	define a user interrupt	Ignored in current release
.IDST	get a task's status	
.IINFOS		returns error ERICM
.INFOS		returns error ERICM
.INIT	initialize a device or directory	Directory must exist; directory name is checked for validity; no "full" option supported (or needed)
.INTAD	define a program interrupt task	

System Call	Description	Special Implementation Notes
.IOPC	initialize the OPCOM package	
.IXMT	transmit interrupt-level message	
.KILAD	define a kill -processing address	
.KILL	kill the calling process	
.LEFD	disable the LEF mode	
.LEFE	enable the LEF mode	
.LEFS	get the LEF mode status	
.LINK	create a link entry	Error in current release
.MAPDF	define a window map	
.MDIR	get master device name	May be controlled by -mdir option
.MEM	determine available memory	
.MEMI	change memory allocation	
.MTDIO	perform direct tape I/O	Uses special tape file formats
.MTOPD	open a tape for direct I/O	Uses -datapath or -path argument
.MULTI	restore the multi-task environment	
.ODIS	disable console interrupts	
.OEBL	enable console interrupts	
.OINFOS		returns error ERICM
.OPEN	open a file	Uses -datapath or -path argument
.OVEX	release overlay, return to address	
.OVKIL	kill calling task, release overlay	
.OVLOD	load user overlay	
.OVOPN	open overlay file	Uses -exepath or -path argument
.OVREL	release overlay	
.OVRP	replace an overlay file	returns error ERICM
.PCHAR	write a character to the console	
.PINFOS		returns error ERICM
.PRI	change a task's priority	
.QTSK	queue a task	
.RDB	read disk block(s)	
.RDCMN	read inter-program common area	Defined as disk file in current version
.RDL	read line	
.RDOPR	read operator message	returns error ERICM
.RDR	read a random record	
.RDS	read sequential bytes	
.RDSW	read console switches	May be controlled by -switches= command line option or Virtual Console instructions
.REC	receive a message	
.REMAP	perform window map	
.RENAM	rename a file	
.RESET	close all files	
.RHIST	secret start histogram call	ignored

System Call	Description	Special Implementation Notes
.RLSE	release a directory or device	Terminates session if master device is released; directory name is checked for validity
.ROPEN	open file for reading	Uses -datapath or -path argument
.RSTAT	ge a resolution file's status	Uses -datapath or -path argument
.RTN	return (pop up) one program level	"panics" system just like standard RDOS if issued from background level 0
.RUCLK	remove a user clock	
.SDAY	set date	ignored
.SECI	reschedule every second	ignored
.SHIST	secret stop histogram call	ignored
.SINGL	disable multi-task environment	
.SMSK	modify interrupt mask	
.SPDA	disable device spooling	ignored
.SPEA	enable device spooling	ignored
.SPKL	delete current spool file	ignored
.SPOS	set current file position	Does not extend data file if position exceeds current file size
.STAT	get file's status	Uses -datapath or -path argument
.STMAP	set data channel map	ignored
.STOD	set time of day	ignored
.SUSP	suspend the calling task	
.SYSI		returns error ERICM
.TASK	create a new task	
.TIDK	kill a task	
.TIDP	chan a task's priority	
.TOVLD	load a user overlay	
.TRDOP	read operator message	
.TUOFF	turn tuning off	ignored
.TUON	turn tuning on	ignored
.TWROP	write operator message	
.UCEX	exit from user clock routine	
.UIEX	exit from user interrupt routine	
.UNLK	delete a link entry	returns error ERICM
.UPDAT	update file information to disk	
.UPIEX	user power fail routine exit	
.VMEM	get extended memory (virtual memory) information	Determined by –vmem command line option
.WCHAR	wait for console character (special .LPART call in Nanos)	returns error ERICM
.WRB	write block(s)	
.WRCMN	write to inter-program common	Defined as disk file in current version
.WREBL	remove memory write protection	ignored
.WRL	write a line	Writes to console may go through simple DG to ANSI terminal control code emulation

System Call	Description	Special Implementation Notes
.WROPR	write operator message	returns error ERICM
.WRPR	write protect memory area	ignored
.WRR	write a random record	
.WRS	write sequential bytes	
.XMT	transmit a message	
.XMTW	transmit message and wait	

Appendix B

ASCII Reference

OCTAL	*DECIMAL*	*HEX*	*EBCDIC*	*CHARACTER*
000	0	00	00	NUL
001	1	01	01	SOH (↑ A)
002	2	02	02	STX (↑ B)
003	3	03	03	ETX (↑ C)
004	4	04	37	EOT (↑ D)
005	5	05	2D	ENQ (↑ E)
006	6	06	2E	ACK (↑ F)
007	7	07	2F	BEL (↑ G)
010	8	08	16	BS (BACKSPACE)
011	9	09	05	HT (TAB
012	10	0A	15	NL (NEW LINE)
013	11	0B	0B	VT (VERT TAB)
014	12	0C	0C	FF (FORM FEED)
015	13	0D	0D	RT (RETURN)
016	14	0E	0E	SO (↑ N)
017	15	0F	0F	SI (↑O)
020	16	10	10	DLE (↑ P)
021	17	11	11	DC1 (↑Q)
022	18	12	12	DC2 (↑R)
023	19	13	13	DC3 (↑S)
024	20	14	3C	DC4 (↑T)
025	21	15	3D	NAK (↑U)
026	22	16	32	SYN (↑ V)
027	23	17	26	ETB (↑ W)
030	24	18	18	CAN (↑ X)
031	25	19	19	EM (↑ Y)
032	26	1A	3F	SUB (↑ Z)
033	27	1B	27	ESC (ESCAPE)
034	28	1C	1C	FS (↑ \ )
035	29	1D	1D	GS (↑])
036	30	1E	1E	RS (↑↑)
037	31	1F	1F	US (↑←)

OCTAL	*DECIMAL*	*HEX*	*EBCDIC*	*CHARACTER*
040	32	20	40	SPACE
041	33	21	5A	!
042	34	22	7F	" (QUOTE)
043	35	23	7B	#
044	36	24	5B	$
045	37	25	6C	%
046	38	26	50	&
047	39	27	7D	' (APOS)
050	40	28	4D	(
051	41	29	5D	)
052	42	2A	5C	*
053	43	2B	4E	+
054	44	2C	6B	, (COMMA)
055	45	2D	60	-
056	46	2E	4B	. (PERIOD)
057	47	2F	61	/
060	48	30	F0	0
061	49	31	F1	1
062	50	32	F2	2
063	51	33	F3	3
064	52	34	F4	4
065	53	35	F5	5
066	54	36	F6	6
067	55	37	F7	7
070	56	38	F8	8
071	57	39	F9	9
072	58	3A	7A	:
073	59	3B	5E	;
074	60	3C	4C	<
075	61	3D	7E	=
076	62	3E	6E	>
077	63	3F	6F	?
100	64	40	7C	"
101	65	41	C1	A
102	66	42	C2	B
103	67	43	C3	C
104	68	44	C4	D
105	69	45	65	E
106	70	46	C6	F
107	71	47	C7	G
110	72	48	C8	H
111	73	49	C9	I
112	74	4A	D1	J
113	75	4B	D2	K
114	76	4C	D3	L
115	77	4D	D4	M
116	78	4E	D5	N
117	79	4F	D6	O

OCTAL	*DECIMAL*	*HEX*	*EBCDIC*	*CHARACTER*
120	80	50	D7	P
121	81	51	D8	Q
122	82	52	D9	R
123	83	53	E2	S
124	84	54	E3	T
125	85	55	E4	U
126	86	56	E5	V
127	87	57	E6	W
130	88	58	E7	X
131	89	59	E8	Y
132	90	5A	E9	Z
133	91	5B		[
134	92	5C	E0	\
135	93	5D		]
136	94	5E	5F	↑ or ^
137	95	5F	6D	← or -
140	96	60	79	` (GRAVE)
141	97	61	81	a
142	98	62	82	b
143	99	63	83	c
144	100	64	84	d
145	101	65	85	e
146	102	66	86	f
147	103	67	87	g
150	104	68	88	h
151	105	69	89	i
152	106	6A	91	j
153	107	6B	92	k
154	108	6C	93	l
155	109	6D	94	m
156	110	6E	95	n
157	111	6F	96	o
160	112	70	97	p
161	113	71	98	q
162	114	72	99	r
163	115	73	A2	s
164	116	74	A3	t
165	117	75	A4	u
166	118	76	A5	v
167	119	77	A6	w
170	120	78	A7	x
171	121	79	A8	y
172	122	7A	A9	z
173	123	7B	C0	{
174	124	7C	4F	|
175	125	7D	D0	}
176	126	7E	A1	~ (TILDE)
177	127	7F	07	DEL (RUBOUT)

↑ means CONTROL

Appendix C

Virtual Console Command Summary

DG-style commands

Letter	Command description
A	open specified special cell, or display value of all accumulators and the carry flag.
B	set a breakpoint location, or display all breakpoint values
D	delete a breakpoint, or all breakpoints
F	display floating point cell(s)
H	set/display instruction history buffer size
I	send I/O reset to specific device, or reset entire device system (IORST)
K	"Kancel" current line (hey, it's DG compatible)
L	load bootstrap from specified device
M	display current active program and data channel map assignments
O	execute specified number of instructions and then return to virtual console
P	proceed from current program counter
Q	quit the current **Re*NOVA*te** session, return optional status value
R	run from the specified address
X	display/modify specified device "delay" value
Z	perform instruction timing
/	open specified memory location
<CR>	close current cell location
<LF>	examine next cell location
^	examine previous cell location
<	redirect input from specified file (execute command file)
>	redirect output to specified file (create trace file)
$	change default display format
<033>	change default display format
.	specify current memory location

Letter	Command description
?	display instruction summary information
!	execute a shell command line
<control>/A	display first instruction history buffer location (most recent instruction executed)
<control>/E	display last instruction history buffer location (oldest instruction executed)

DG-style Virtual Console Commands

Extended commands

Extended Command	Description
attach { <*device_unit*> <*filename*> }…	attach operating system file to specified device or unit of a multi-unit device
boot <*device*> [nostart]	boot from the specified device, do not automatically start program execution if the 'nostart' option is specified
detach <*device_unit*>…	detach operating system file from specified device or unit of a multi-unit device
disable <*device*>…	disable I/O instruction support for specified device
enable <*device*>…	enable I/O instruction support for specified device
iorst [all]	send IORST to all devices
iorst <*device*>…	send IORST to specified <*device*>(s)
set breakpoint { -all | [-]<*integer*> }…	set or remove breakpoints: positive integer sets breakpoint at location <*integer*>; negative integer removes breakpoint number <*integer*>; -all removes all breakpoints
set <*device*> { <*item*> [<*value*>] }…	set register <*item*>[s] to <*value*>[s] for device <*device*>
set history <*integer*>	set execution history buffer to <integer> number of entries; previous entries are lost.
show <*device*>	show unit information about the specified device
show <*device*> <*item*>…	show information about register *item*(s) for the specified device
show breakpoints	show all active breakpoints
show all	show information about all enabled devices.
show disabled	show information about all disabled devices.
show enabled	show information about all enabled devices (same as 'all')
show history	display instruction history information
show history <*integer*>	display instruction history information for the last *integer* instructions executed
show history info	display instruction history buffer size
show queue	display current event queue
show map	display current active program and data channel

Extended Command	Description
	map (mapped systems only)
show time	display current event time
sleep <*integer*>…	delay command execution for <*integer*> seconds

Extended Virtual Console Commands

Notes:

<*device*>	I/O device name
<*device_unit*>	I/O device name or device name and unit number
<*filename*>	represents any valid operating system file name
<*integer*>	may be an octal, decimal or hexadecimal non-negative integer
<*value*>	represents a integer or a boolean ("true" or "false") value

Appendix D

I/O Device Codes

Device code	mnemonic	priority mask	DG	Point 4	Strobe Data	Description
00	---	---	x	x	x	unused
01	MDV	---	x			multiply/divide
02	MMU	---	x			Memory management unit
03	MMU1	---	X			
04						
05						
06	MCAT					
07	MCAR					
10	TTI					
11	TTO					
12	PTR					
13	PTP					
14	RTC					
15	PLT					
16	CDR					
17	LPT					
20	DSK					
21	ADCV					
22	MTA					
23	DACV					
24	DCM					
25						
26						
27						
30	QTY					
31	IBM1					
32	IBM2					
33	DKP					
34	CAS					
35	CRC					
36	IPB					
37	IVT					
40	DPI					
41	DPO					

42	DIO					
43	DIOT					
44	MXM					
45						
46	MCAT1					
47	MCAR1					
50	TTI1					
51	TTO1					
52	PTR1					
53	PTP1					
54	RTC1					
55	PLT1					
56	CDR1					
57	LPT1					
60	DKS1					
61	ADCV1					
62	MTA1					
63	DACV1					
64	FPU1					
65	FPU2					
66	FPU					
67						
70	QTY1					
70	SLA1					
71						
72						
73	DKP1					
74	CAS1					
74	FPU1					
75	FPU2					
76	FPU					
77	CPU					

Notes:

<*device*> I/O device name

<*device_unit*> I/O device name or device name and unit number

Index

Index

D

E

F

Index

Index

Index

W