



H893V
USER-Access™ for Stratus™ OpenVOS Systems

Release 4.1

Software Reference Manual

Revision Record

Revision	Description
01 (02/2012)	Initial revision for this product.

© 2012 by Network Executive Software. Reproduction is prohibited without prior permission of Network Executive Software. Printed in the U.S.A. All rights reserved.

The U.S. Department of Commerce restricts the distribution of technical information contained in this document when exported outside the U.S. Therefore, careful attention should be given to compliance with all applicable U.S. Export Laws if any part of this document is to be exported.

You may submit comments by addressing e-mail to:

support@netex.com

or, by visiting our web site at:

<http://www.netex.com>

Always include the complete title of the document with your comments.

Preface

This manual describes the H893V USER-Access software for the Stratus system. USER-Access can be used in conjunction with Network Executive Software's NETWORK EXECutive (NetEx) software to allow the end-user to easily transfer files across the network.

This manual is intended for all users of USER-Access, and contains all of the information necessary to expand the user's ability to the fullest extent of the software.

The manual is divided into seven parts plus one appendix:

“Introduction” gives a basic description of USER-Access and a sample of a USER-Access session.

“Stratus Local User's Guide” describes the features of USER-Access on Stratus systems as seen by the local user. This section includes a description of the commands in the local interface.

“Stratus Remote User's Guide” describes features of H893V as seen by a remote user. This includes executing commands on a Stratus host remotely and transferring files to and from a remote Stratus host.

“File Handling Under Stratus USER-Access” describes the way Stratus manipulates files. This includes examples of transferring files, transfer modes supported by the Stratus USER-Access, wildcard characters, and file specifications.

“Advanced Local User's Guide” describes the advanced features of USER-Access on the IBM systems as seen by the local user.

“Command Descriptions” provides detailed descriptions of all commands available in H893V USER-Access.

Reference Material

The following manuals contain related information.

Number	Title and Description
MAN-REF-H897IPV	<i>H897IPV NetEx/IP® Requester for Stratus OpenVOS Software Reference Manual</i>

Notice to the Reader

The material contained in this publication is for informational purposes only and is subject to change without notice. Network Executive Software is not responsible for the improper use of any product options, the use of features not described in this publication, and assumes no responsibility for any errors that may appear in this publication. Refer to the revision record (at the beginning of this document) to determine the revision level of this publication.

Network Executive Software does not by publication of the descriptions and technical documentation contained herein, grant a license to make, have made, use, sell, sublicense, or lease any equipment or programs designed or constructed in accordance with this information.

This document may contain references to the trademarks of the following corporations:

Corporation Trademarks and Products

Network Executive Software: NetEx, BFX, PFX, USER-Access, NESiGate

These references are made for informational purposes only.

The diagnostic tools and programs described in this manual are **not** part of the products described.

Notice to the Customer

The installation information supplied in this document is intended for use by experienced System Programmers.

Software Modification Policy

Modifications to H893V that are not specifically authorized by NetEx Software are prohibited.

Any unauthorized modifications to H893V may affect its operation and/or obstruct NetEx Software's ability to diagnose problems and provide corrections. Any work resulting from unauthorized modifications shall be paid by the customer at NetEx Software's then-current support rates and may result in the immediate termination of warranty/support coverage.

Document Conventions

The following notational conventions are used in this document:

Format	Description
displayed information	Information displayed on a CRT (or printed) is shown in this font.
<i>user entry</i>	<i>This font</i> is used to indicate the information to be entered by the user.
UPPERCASE	The exact form of a keyword that is not case-sensitive or is issued in uppercase.
MIXedcase	The exact form of a keyword that is not case-sensitive or is issued in uppercase, with the minimum spelling shown in uppercase.
bold	The exact form of a keyword that is case-sensitive and all or part of it must be issued in lowercase.
lowercase	A user-supplied name or string.
Value	Underlined parameters or options are defaults.
<label>	The label of a key appearing on a keyboard. If “label” is in uppercase, it matches the label on the key (for example: <ENTER>). If “label” is in lowercase, it describes the label on the key (for example: <up-arrow>).
<key1><key2>	Two keys to be pressed simultaneously.
No delimiter	Required keyword/parameter.

Glossary

asynchronous: A class of data transmission service whereby all requests for service contend for a pool of dynamically allocated ring bandwidth and response time.

ASCII: Acronym for American National Standard Code for Information Interchange.

buffer: A contiguous block of memory allocated for temporary storage of information in performing I/O operations. Data is saved in a predetermined format. Data may be written into or read from the buffers.

code conversion: An optional feature in the adapter or host DX interface that dynamically converts the host data from one character set to another. An adapter configured with the code conversion has a special 1K RAM that is used for code conversion. This RAM can be loaded with any type of code (for example, ASCII, EBCDIC, et cetera).

Configuration Manager: A utility that parses a text NCT file into a PAM file.

header: A collection of control information transmitted at the beginning of a message, segment, datagram, packet, or block of data.

host: A data processing system that is connected to the network and with which devices on the network communicate. In the context of Internet Protocol (IP), a host is any addressable node on the network; an IP router has more than one host address.

Internet Protocol (IP): A protocol suite operating within the Internet as defined by the *Requests For Comment* (RFC). This may also refer to the network layer (level 3) of this protocol stack (the layer concerned with routing datagrams from network to network).

ISO: Acronym for International Standards Organization.

Network Configuration Table (NCT): An internal data structure that is used by the NETEX configuration manager program to store all the information describing the network.

NETwork EXecutive (NETEX): A family of software designed to enable two or more application programs on heterogeneous host systems to communicate. NETEX is tailored to each supported operating system, but can communicate with any other supported NETEX, regardless of operating system.

NETEX can reside on the host, on a processor interface board (obsolete), or in a DX unit. The latter two cases use host-resident drivers as interfaces.

NETEX is a registered trademark of Network Executive Software.

Open Systems Interconnection (OSI): A seven-layer protocol stack defining a model for communications among components (computers, devices, people, and et cetera) of a distributed network. OSI was defined by the ISO.

path: A route that can reach a specific host or group of devices.

TCP/IP: An acronym for Transmission Control Protocol/Internet Protocol. These communication protocols provide the mechanism for inter-network communications, especially on the Internet. The protocols are hardware-independent. They are described and updated through *Requests For Comment* (RFC). IP corresponds to the OSI network layer 3, TCP to layers 4 and 5.

Contents

Revision Record	i
Preface.....	iii
Reference Material.....	iv
Notice to the Reader.....	v
Corporation Trademarks and Products.....	v
Notice to the Customer	v
Software Modification Policy	v
Document Conventions.....	vi
Glossary	vii
Contents	ix
Figures.....	xvi
Tables	xvi
Introduction.....	1
USER-Access Overview	1
How USER-Access Works	1
Introduction to USER-Access and Stratus	2
Introduction to USER-Access and NETEX	2
USER-Access, NETEX and the ISO Model	3
Sample Stratus USER-Access Session.....	5
Stratus Local User's Guide	11
Introduction.....	11
Invoking USER-Access on Stratus	11
Local Stratus USER-Access Startup Files	12
Remote USER-Access Startup Files	13
Getting Started	13
USER-Access Commands and Command Qualifiers	13
Displaying the Valid Qualifiers for a Command	13
Displaying the Current Value of a Qualifier	14
Setting a Command Qualifier	14
Overriding a Command Qualifier	15
Online Help.....	15
Controlling USER-Access Input and Output.....	16
USER-Access Error Messages.....	17
Aliasing.....	18
Terminating a USER-Access Session.....	19
Establishing a Connection to a Remote Host.....	19
Using CONNECT to Establish a Connection	20
Using LOGIN to Establish a Connection.....	21
Exchanging Host Information on Connect	21
Establishing Multiple Host Connections	23

Disconnecting from a Host.....	24
Transferring Files as a Local User.....	24
Sending Files to a Remote Host	24
Receiving Files from a Remote Host.....	25
Send and Receive Qualifiers	26
Executing Remote Host Commands.....	27
Executing Local OpenVOS Commands.....	28
Issuing Local OpenVOS Host-Independent Commands	30
Editing Remote Files with an Editor	31
Interrupting a Command within Stratus USER-Access.....	32
Stratus Remote User's Guide	33
Connecting into a Stratus OpenVOS Host.....	33
CONNECT Qualifiers Used by Stratus USER-Access	33
Remote Stratus USER-Access Startup Files.....	33
Transferring Files to a Stratus Host.....	34
Executing Remote OpenVOS Commands.....	34
Issuing Remote OpenVOS Host Independent Commands	35
File Handling Under Stratus USER-Access	39
Stratus File Transfer Qualifiers and Default Values.....	39
Definition of DIRECTORY Under Stratus USER-Access.....	40
Stratus File Name Specifications.....	41
Stratus File Specification Examples	41
File Transfer Examples from a Local Stratus Host	42
Example 1.....	42
Example 2.....	42
Example 3.....	42
File Transfer Examples to a Remote Stratus Host.....	42
Example 1.....	42
Example 2.....	43
Source Wildcard Support for Stratus File Transfers.....	43
Destination Wildcard Support for Stratus File Transfers	43
Accessing a Stratus Tape Device.....	44
Transfer Modes Supported Under Stratus USER-Access.....	45
Advanced Local User's Guide	47
Introduction	47
Special Characters	47
USER-Access String Substitution	48
String Variables.....	49
String Literals.....	52
String Functions	52
Arithmetic Operations	55
CHR Function.....	56
CMP Function	57
DATE Function	58
DEC and INC Functions.....	59
DFN and NDF Functions.....	60
ENCRYPT Function.....	61
ENV Function.....	62

EQS and NES Functions	63
EXT Function.....	64
INDEX Function.....	65
LEN Function.....	66
Logical Operations.....	67
LOWER and UPPER Functions	68
MSG Function.....	69
PARAMS Function.....	70
SLEEP Function.....	71
STATUS Function	72
TIME Function.....	73
Disabling String Substitution.....	74
Nested String Substitution	74
Developing USER-Access Scripts Using Input Files and Aliases	75
USER-Access Input Files	75
Echoing Input Scripts to the Terminal	76
Displaying Output and Accepting Input within a Script.....	76
Passing Parameters to a Script	77
Using String Functions within a USER-Access Script.....	78
Using USER-Access Labels and GOTOs	80
Using the ON (ERROR/INTERRUPT) Command.....	81
Checking Command Status.....	82
Creating USER-Access Aliases	82
USER-Access Aliases vs. Host Aliases	83
Creating Multicommand USER-Access Aliases.....	84
Passing Parameters to an Alias	84
Accepting Input within a USER-Access Alias	85
Abbreviating Alias Names	86
Defining Multiword Alias Names.....	86
Debugging a USER-Access Alias or Input Script	87
Error Message Formatting	88
USER-Access Code Conversion.....	89
USER-Access Data Verification.....	89
USER-Access Data Compression	90
Character Mode Compression.....	91
Stratus USER-Access SEARCH Keywords (SITE), (USER), and (NONE)	92
User-Definable HELP Files Under Stratus	94
Running USER-Access as a Batch Job Under Stratus	96
Running a Stratus Stand-Alone USER-Access Server.....	97
Advanced Stratus Transfer Modes	97
Central Archiving	101
Introduction.....	101
Using Archiving.....	103
Archiving from a Satellite Site.....	104
Archiving from a Central Site.....	105
File Transfer Options	107
Stratus Privileges Required for Archiving.....	108
NETEX Considerations for Archiving.....	108
Stratus as a Central Site	108
The Central Archiving OpenVOS Command Macros	109

UA_SAVE.CM OpenVOS Command Macro.....	109
Description	109
Examples	109
UA_RESTORE.CM OpenVOS Command Macro.....	110
Description	110
Examples	110
UA LIST OpenVOS Command Macro	110
Description	110
Example.....	110
The Central Archiving Aliases	111
BACKUP Alias	111
Description	111
Format	111
Example.....	111
LIST Alias	112
Description	112
Format	112
Example.....	112
RESTORE Alias.....	113
Description	113
Format	113
Example.....	113
Command Descriptions.....	115
ASK Command	116
Description	116
Format	116
Examples	117
Related Topics.....	117
CONNECT Command.....	118
Description	118
Format	119
Host Dependencies	120
Examples	121
Related Topics.....	121
CONTINUE Command	122
Description	122
Format	122
Example.....	122
Related Topics.....	122
DISCONNECT Command	123
Description	123
Format	123
Examples	123
Related Topics.....	124
ENCRYPT Command	125
Format	125
Examples	125
Example 1: Encrypting Passwords Stored in a USER-Access Input Script File.....	125
Example 2: Using USER-Access to Generate the Input Script File.....	126
Related Topics.....	126

EXIT Command.....	127
Description.....	127
Format.....	127
Examples.....	127
Related Topics	128
GOTO Command.....	129
Description.....	129
Format.....	129
Examples.....	129
Related Topics	129
HELP Command.....	130
Description.....	130
Format.....	130
Examples.....	130
INPUT Command	132
Description.....	132
Format.....	132
Examples.....	133
Related Topics	134
LOCAL Command.....	135
Description.....	135
Format.....	135
Informational Qualifiers	135
Examples.....	136
Related Topics	136
ON Command	137
Description.....	137
ON ERRor.....	137
ON INTerrupt.....	137
ON LOCAL_error	137
ON REMote_error	137
Format.....	137
Examples.....	138
Related Topics	139
OUTPUT Command	140
Description.....	140
Format.....	140
Informational Qualifiers	141
Examples.....	141
Related Topics	141
QUIT Command	142
Description.....	142
Format.....	142
Examples.....	142
Related Topics	142
RECEIVE Command	143
Description.....	143
Format.....	143
Examples.....	143
Related Topics	143
REMOTE Command.....	144

Description	144
Format	144
Informational Qualifiers	144
Examples	145
Related Topics	145
SEND Command	146
Description	146
Format	146
Examples	146
Related Topics	146
SET Command	147
Description	147
Format	147
Examples	147
Related Topics	148
SET ALIAS Command	149
Description	149
Format	149
Host Dependencies	149
Examples	150
Related Topics	150
SET GLOBAL Command	151
Description	151
Format	151
Example	151
Related Topics	152
SET HOST Command	153
Description	153
Format	153
Examples	153
Related Topics	153
SET VARIABLE Command	155
Description	155
Format	155
Examples	155
Related Topics	156
SHOW Command	157
Description	157
Format	157
Examples	157
Related Topics	157
SHOW ALIAS Command	158
Description	158
Format	158
Examples	158
Related Topics	158
SHOW GLOBAL Command	159
Description	159
Format	159
Examples	159
Related Topics	159

SHOW HOSTS Command.....	160
Description.....	160
Format.....	160
Examples.....	160
Related Topics	160
SHOW QUALIFIER Command	161
Description.....	161
Format.....	161
Examples.....	161
Related Topics	161
SHOW VARIABLE Command	162
Description.....	162
Format.....	162
Examples.....	162
Related Topics	162
TEXT Command.....	163
Description.....	163
Format.....	163
Examples.....	163
Related Topics	163
TRANSLATE Command.....	164
Description.....	164
Format.....	165
Examples.....	165
Example #1:	165
Example #2:	166
Example #4	167
Appendix A. USER-Access Error Messages for Stratus	169
Additional Descriptions	179
Index.....	189

Figures

Figure 1. USER-Access Architecture 1

Figure 2. Diagram of a USER-Access Connection Sequence 2

Figure 3. ISO Model Communication 4

Figure 4. NetEx and the ISO Model 5

Figure 5. Nested String Substitution 74

Figure 6. Sample Command File 96

Figure 7. Traditional Backup 101

Figure 8. Central Archiving 101

Figure 9. Satellite and Central Sites 104

Figure 10. Data Center 106

Figure 11. Sample Central Site JCL Command Procedure 107

Tables

Table 1. ISO Model 3

Table 2. List of Functions 53

Table 3. Error Messages 170

Introduction

USER-Access Overview

The Network Executive Software® USER-Access™ software is a user interface to NETEX®. It provides the ordinary user with a means to move and manipulate files across a HYPERchannel® or Data Exchange (DX) network using simple, easily remembered commands. In addition, USER-Access provides extensive interactive help files so the user can become familiar with USER-Access.

USER-Access provides several advantages to network users. Among these are:

- User-friendly - Once USER-Access is installed, you can transfer files and exercise other USER-Access functions in very little time and with little training.
- Tailorable - The USER-Access interface can be tailored to meet your needs at the host and the user levels. Default values can be set, aliases defined, etc., in site- and user-input files that are read by USER-Access when it is invoked.
- Common Interface - The USER-Access user interface is the same on all hosts. While the definition of a command may change from one host to another, the command remains the same to you.
- Security - USER-Access uses the host computers' logon routines to provide security. You must be a valid user on both the local and the remote systems to access them. Some systems may allow a guest account, but this can be restricted by the security needs of the network.

Figure 1 shows the arrangement of USER-Access and NETEX. The design of USER-Access allows for expansion as its use increases. Possible expansion includes queued file submission, electronic mail, and similar applications.

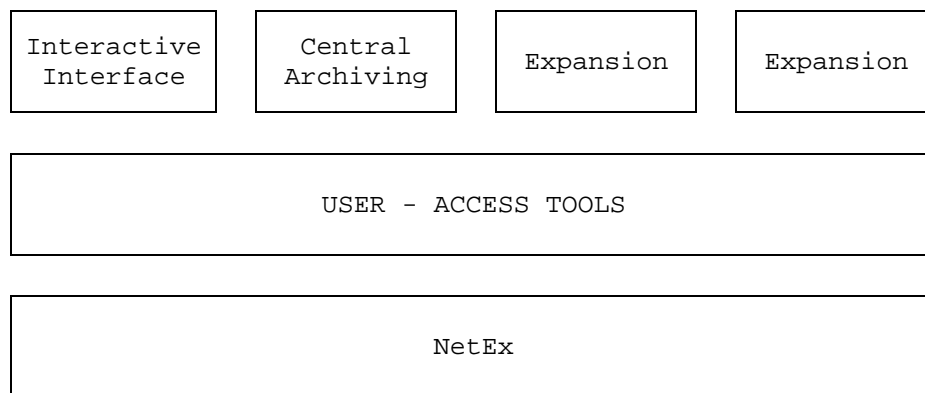


Figure 1. USER-Access Architecture

How USER-Access Works

USER-Access, whether full function, initiator, or responder, follows the same pattern of function. This pattern is illustrated in Figure 2 on page 2. As the figure shows, the user (initiator) sends a request to the remote (or responding) Service Initiator, including account and password information (1). The responding Service Initiator logs the user in and starts up a service module (2), which then offers a service and notifies the Service Initiator (3). The Service Initiator module then returns a message to the Initiator and disconnects to wait for an-

other incoming request (4). The Initiator then reconnects directly to the Service Module (5). The connection between the initiator and the Service Module is maintained until terminated by a command or a system time-out.

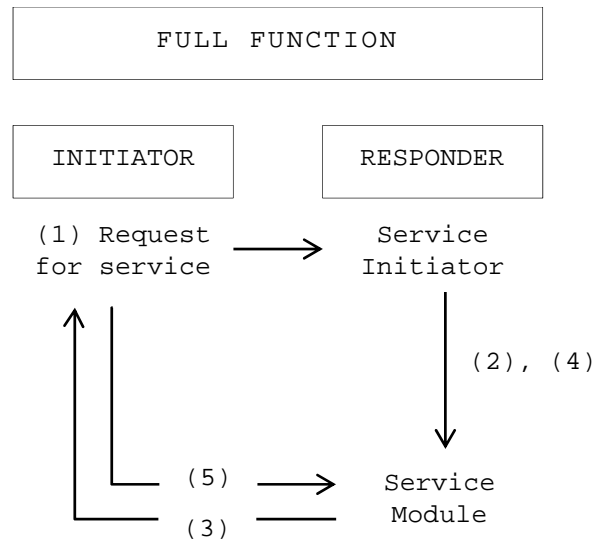


Figure 2. Diagram of a USER-Access Connection Sequence

Introduction to USER-Access and Stratus

This manual describes the Network Executive Software USER-Access software for a Stratus host using running OpenVOS and using the H897IPV NetEx/IP Requester.

USER-Access is a software product designed to simplify network communications. Network capabilities have been expanded to include the nontechnical user by reducing the user interface to a set of simple commands (CONNECT, SEND, RECEIVE, DISCONNECT, etc.).

Introduction to USER-Access and NETEX

The NETEX and USER-Access families of software products are used to enable two or more application programs (which may be on different host computers) to communicate with each other at multimegabit speeds. The NETEX family of software consists of different versions of NETEX for use with different operating systems, such as DEC, Sun or IBM. Network Executive Software also has utility programs available for use with NETEX, such as USER-Access, which simplifies the user interface to an even greater degree.

The NETEX software resides as a stand-alone, real-time program within each host involved in the connection. As an independent program, NETEX allows communications to take place at any time during host operations, independent of other functions in the system. USER-Access can be used with host-based NETEX and NetEx/IP Requester. The following sections describe USER-Access, these two NETEX products, and how NETEX uses the International Standards Organization (ISO) guidelines for open systems interconnection.

USER-Access, NETEX and the ISO Model

In creating USER-Access and NETEX, Network Executive Software followed the guidelines set by the International Standards Organization (ISO) for Open Systems Interconnection. Open Systems Interconnection refers to the exchange of information among terminal devices, computers, people, networks, etc., that are open for communication with one another.

The ISO model is composed of seven layers. USER-Access forms the application (or user) layer and the presentation layer of the ISO model. Each of the seven layers interacts only with the adjacent layers in the model (see Table 1). By using this modular structure, the internal function of each layer is self-contained and does not affect the operation of other layers.

Layer	Major Functions
Application	High level description of data to be transferred and the destination involved
Presentation	Select data formats and syntax
Session	Establish session connection, report exceptions, and select routing
Transport	Manage data transfer and provide NETEX-to-NETEX message delivery
Network	Point-to-point transfer, error detection, and error recovery
Data Link	Data link connection, error checking, and protocols
Physical	Mechanical and electrical protocols and interfaces

Table 1. ISO Model

Although each layer physically interacts only with adjacent layers, each layer appears to communicate directly with the corresponding layer of the other model. Figure 3 on page 4 illustrates this concept.

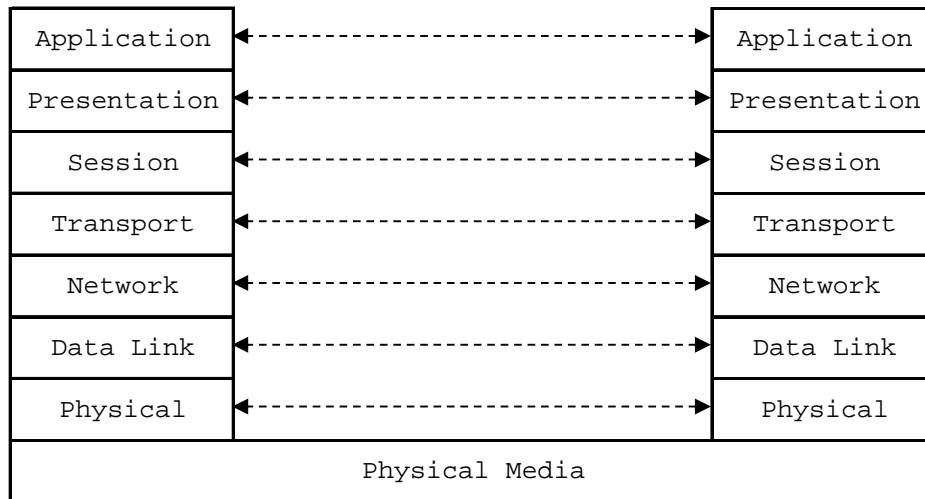


Figure 3. ISO Model Communication

Notice that the corresponding layers appear to communicate directly as indicated by the dotted lines. In fact, they communicate only by progressing down through the layers of one model, through the physical media, and up through the layers of the other model.

As shown in Figure 4 on page 5, supported network hardware and firmware form the lower two layers. NETEX and the user's operating system comprise the next three layers. (NETEX software provides complete session and transport layer services, plus a network interface at the driver level.) This leaves the user free to use Network Executive Software utilities or write the application programs for use with NETEX. USER-Access completes the model by providing presentation and application level software.

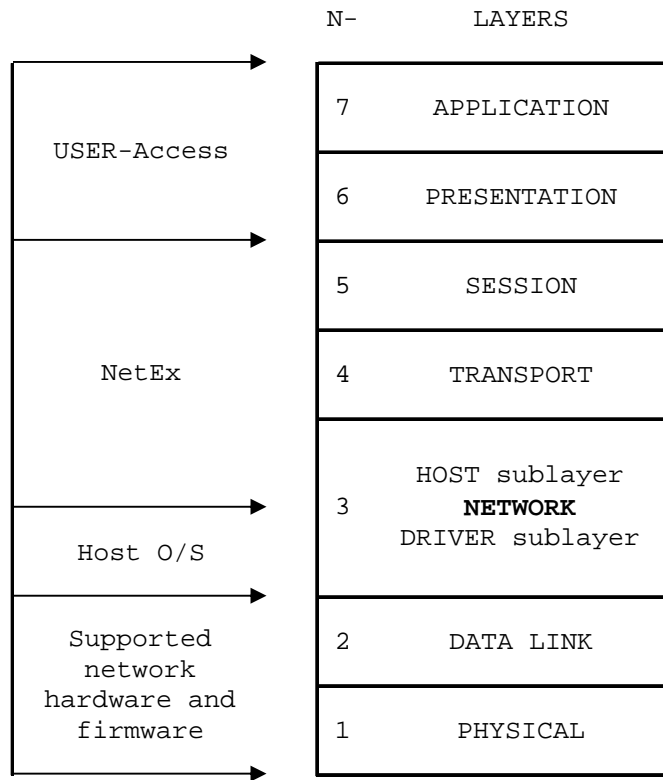


Figure 4. NetEx and the ISO Model.

Sample Stratus USER-Access Session

This section gives a very brief example of a few of the functions that can be accomplished during a USER-Access session. This sample session is meant to be only a simple introduction to USER-Access and how it may appear to the local Stratus user. The sections following this provide a more detailed look at the product and its features. Users that have never seen USER-Access may spend a couple of minutes following through this sample session. Users that are familiar with the product may skip directly to the next section.

To invoke USER-Access, the user command is entered from the Stratus command line as:

```
ready 13:30:00
user
user>
```

The returning prompt in this sample session is `user>`, although USER-Access may be configured to return another prompt. The prompt informs the user that USER-Access is waiting to accept a command.

A connection to any host in the network that is running USER-Access can be made using the LOGIN command. The LOGIN command below establishes a connection with an IBM MVS host named 'MVSXA' (in the Network Configuration Table). LOGIN prompts the user for login information such as remote username and password which it uses to establish a secure login to the remote host. The LOGIN output returned is based on the host and username to which the connection is made. The connection is completed when a USER-Access prompt appears. Notice that in this session USER-Access has been configured to prompt with the name of the remote host 'MVSXA'.

```

User> login
Hostname? mvsxa
Username? guest1
Password? _____
Qualifiers?
User: Connected to service 'USER' on host 'MVSXA'.
-----
GUEST1 LOGON IN PROGRESS AT 13:30:19 ON OCTOBER 3, 1989
NO BROADCAST MESSAGES
READY
-----
User: Logged in as user 'guest1'.
MVSXA>

```

If a connection fails, an error message is displayed. The error generally begins:

```
User: Failed to connect service 'USER' on host 'MVSXA' (UA-4105),
```

This is followed by either a NETEX message or a remote system error message. If the 'MVSXA' host is not in the Network Configuration Table (NCT), for example, an error similar to the following would appear:

```
User: Host 'MVSXA' does not exist in configuration (UA-804).
User: Host is not in the configuration (NETEX-3506).
```

If the username, password combination was invalid, an error such as the one below would be seen:

```
User: Failed to connect service 'USER' on host 'MVSXA' (UA-4105).
User: Remote: Login failure.
```

Of course, since all logins are made through the security system of the remote host, the error message actually seen by the user will depend on the host to which the connection is being made.

Following a successful login as above, a SHOW HOSTSS command can be used to display all remote host connections held by this USER-Access session. Each session can support up to ten host connections. The command below reveals just one remote host connection. The connection displayed is the one just established by LOGIN at the beginning of this session.

```

MVSXA> show hosts
User:
User: active --> (1)Host=MVSXA      User=GUEST1
User:
MVSXA>

```

Once a connection is established, a SHOW REMOTE command can be issued to return useful information about the connection and the remote USER-Access host. From the list below, for example, it can be seen that the remote host character code is **EBCDIC**, the default directory (or TSO Prefix), is **MVS**, and the USER-Access version number is 5.4.3.


```

ZOSY> show remote
User:
User: * BLOCKsize ..... 16384
User: * COPYRight ..... COPYRIGHT (c) 1999-2011 - Network Executive
Software, Inc. Mpls. MN
User:  DIReCTory ..... TEST1
User: * GATEWay .....
User:  HOMEdir ..... TEST1
User: * HOST ..... ZOSY
User: * HOSTCODE ..... EBCDIC
User: * HOSTENV ..... TSO FOREGROUND
User: * HOSTOS ..... z/OS 1.8
User: * HOSTTYPE ..... MVS
User: * PID ..... 0XF585808DF4B0
User:  PREFix ..... MVS:
User: * PRODUct ..... EFT213
User:  QUIet ..... off
User: * ROOTdir ..... EFT.EFT3.NTX.TEXT
User: * SERvice ..... EFT
User: * STATus .....
User: * TRANSlate ..... Network
User:  TSOPREfix ..... TEST1
User: * USERname ..... TEST1
User: * VERsion ..... 5.4.3
User:
User: * Informational qualifier (cannot be modified).
User:
ZOSY>

```

Similar information can also be displayed about the local Stratus host by issuing the **SHOW LOCAL** command. Note here that the local character code is ASCII7, the current local directory is %mkt2#m2_d01>nsc>sicom_nesi>user, and the local version of USER-Access is Release 4.1.

```

ZOSY> show local
User:
User:  DIReCTory ..... %mkt2#m2_d01>nsc>sicom_nesi>user
User: * GATEWay .....
User: * HOSTCODE ..... ASCII7
User: * HOSTTYPE ..... STRATUS
User:  INTeRactive ..... off
User: * NETwork ..... NETEX
User: * PID ..... 117539506
User:  PREFix ..... Stratus:
User: * PRODUct ..... UA893V
User:  QUIet ..... off
User: * STATus .....
User: * USERname ..... User_Name.RDHD
User: * VERsion ..... Release 4.1
User:
User: * Informational qualifier (cannot be modified).
User:
ZOSY>

```

Once a connection is established to a remote host, users can issue commands to that host using the **REMOTE** command. The example below issues the **REMOTE LISTCAT** command (an IBM MVS TSO command)

which says to return a directory listing of files that reside on the remote host in the current default directory. Notice that a host specific prefix appears in the left hand column indicating the results are being returned from the **MVS** host.

```
ZOSY> remote listcat
MVS: IN CATALOG:CAT.MVS3.TSO
MVS: GUEST1.BATCH.DATA
MVS: GUEST1.BATCH.JCL
MVS: GUEST1.MVS.PDS
MVS: GUEST1.SAMPLE.C
ZOSY>
```

A major feature of USER-Access is its implementation of a Host Independent Command set. Host independent commands allow a user to issue similar commands on all hosts around the network, without having to learn each host's native command set. The command in the example above can be issued again, but this time using the Host Independent Command **DIRECTORY**. USER-Access simply maps **DIRECTORY** to the IBM MVS **LISTCAT** command. Now network users need only learn one network-wide command set. This command set can be the USER-Access default one or one that the site defines. Below is a second pass at a remote directory listing, but this time using the Host Independent Command **REMOTE DIRECTORY**.

```
ZOSY> remote directory
MVS: IN CATALOG:CAT.MVS3.TSO
MVS: GUEST1.BATCH.DATA
MVS: GUEST1.BATCH.JCL
MVS: GUEST1.MVS.PDS
MVS: GUEST1.SAMPLE.C
ZOSY>
```

Local OpenVOS commands or local Host Independent Commands can also be executed from within USER-Access using the **LOCAL** command. Here, a local directory listing is given (using the Host Independent Command **DIRECTORY**), showing all files in the local user's current directory. The prefix in the left hand column now reflects the local host's type 'Stratus'.

```
ZOSY> local directory
Stratus:
Stratus:  %nsc#m2_d01>ua
Stratus:
Stratus:  Files: 5, Blocks: 14
Stratus:
Stratus:  w          5 fix-4096  89-09-23 12:34:51  demo.pm
Stratus:  w          1 seq       89-09-23 12:34:51  demo.ua
Stratus:  w          1 seq       89-08-16 11:14:43  pause.for
Stratus:  w          6 seq       89-09-10 10:24:49  script.ua
Stratus:  w          1 seq       89-08-16 11:34:07  tutorial.ua
Stratus:
Stratus:  Directories: 1
Stratus:
Stratus:  m          1 89-10-03 08:03:15  tests
Stratus:
Stratus:  Links: 1
Stratus:
Stratus:  89-10-03 10:03:28  user.pm -> %nsc#m2_d01>ua>user>user.pm
Stratus:
ZOSY>
```

To transfer a file from the local host to the remote host, the **SEND** command is used. The example below sends the file **DEMO.UA** from the current local directory **%nsc#m2_d01 > ua** on the Stratus host, to the current remote directory (or TSO Prefix) **GUEST1** on the IBM MVS host. Since all USER-Access commands

can be predefined with reasonable site defaults, the typical user would just type SEND followed by the source file name. The status line indicates the file has successfully been transferred, Notice that USER-Access uses the source file name to create a default destination file name when one isn't specified.

```
ZOSY> send demo.ua
User: Source                               Destination                               Size
User: -----                               -----                               -----
User: %nsc#m2_d01>ua>demo.ua              GUEST1.DEMO.UA                          1759
ZOSY>
```

With USER-Access it is also very easy to transfer a group of files using a single command. The example below sends all of the local files having an extension of 'UA' to the remote IBM MVS host. Per the user's request, the files are stored in a single IBM Partitioned Data Set (PDS), called STRATUS.PDS, as individual members DEMO, SCRIPT, and TUTORIAL. If any file transfer errors were encountered, they would be displayed in place of the status line below.

```
ZOSY> send *.ua stratus.pds(*)
User: Source                               Destination                               Size
User: -----                               -----                               -----
User: %nsc#m2_d01>ua>demo.ua              GUEST1.STRATUS.PDS(DEMO)                 1759
User: %nsc#m2_d01>ua>script.ua           GUEST1.STRATUS.PDS(SCRIPT)                21857
User: %nsc#m2_d01>ua>tutorial.ua         GUEST1.STRATUS.PDS(TUTORIAL)              268
ZOSY>
```

A quick REMOTE DIRECTORY will act as a second verification that the files have indeed been transferred. Note the new files DEMO.UA and STRATUS.PDS below.

```
ZOSY> remote directory
MVS: IN CATALDG:CAT.MVS3.TSO
MVS: GUEST1.BATCH.DATA
MVS: GUEST1.BATCH.JCL
MVS: GUEST1.DEMO.UA
MVS: GUEST1.MVS.PDS
MVS: GUEST1.SAMPLE.C
MVS: GUEST1.STRATUS.PDS
ZOSY>
```

File transfer is just as easy the other direction. To move a file from the remote host to the local host, use the RECEIVE command. The example below transfers the file SAMPLE.C from the IBM MVS system to the local Stratus host.

```
ZOSY> receive sample.c
User: Source                               Destination                               Size
User: -----                               -----                               -----
User: GUEST1.SAMPLE.C                      %nsc#m2_d01>ua>sample.c                   106
ZOSY>
```

This transfer can also be verified by viewing a LOCAL DIRECTORY listing.

```

ZOSY> local directory
Stratus:
Stratus:  %nsc#m2_d01>ua
Stratus:
Stratus:  Files: 5, Blocks: 15
Stratus:
Stratus:  w          5 fix-4096  89-09-23 12:34:51  demo.pm
Stratus:  w          1 seq       89-09-23 12:34:51  demo.ua
Stratus:  w          1 seq       89-08-16 11:14:43  pause.for
Stratus:  w          1 seq       89-10-03 13:33:33  sample.c
Stratus:  w          6 seq       89-09-10 10:24:49  script.ua
Stratus:  w          1 seq       89-08-16 11:34:07  tutorial.ua
Stratus:
Stratus:  Directories: 1
Stratus:
Stratus:  m          1 89-10-03 08:03:15  tests
Stratus:
Stratus:  Links: 1
Stratus:
Stratus:  89-10-03 10:03:28  user.pm -> %nsc#m2_d01>ua>user>user.pm
Stratus:
ZOSY>

```

To force a disconnection from all remote hosts (in this case the IBM MVS host), the EXIT command is used. EXIT insures a smooth shut down of network activities as well as local and remote files. EXIT also returns a USER-Access session status that can be interpreted by the local host. This status is especially useful when USER-Access is used within a batch job.

```

ZOSY> exit
ready 13:35:07

```

To keep this sample session short, no more commands or features of USER-Access will be shown. However, since only a small fraction of USER-Access has been described here, the user is encouraged to read the remaining sections for a full description of the benefits that can be realized using the product.

Stratus Local User's Guide

Introduction

This section is intended for users that would like an introduction to USER-Access and some of its features. This section explains how to invoke USER-Access from a Stratus terminal, what a USER-Access session looks like, logging in and transferring files to a remote host on the network, and executing commands on a remote host. Users are encouraged to refer to "Advanced Local User's Guide" on page 47 for a more in depth look into USER-Access. Users should also refer to the User's Guide for the remote host in which a connection will be made for additional information about that host's environment.

Invoking USER-Access on Stratus

USER-Access is invoked using the following general format:

```
user [input-file [argument1, argument2, ...]][-keyword value]
```

user	is the name of the command used to invoke USER-Access. It is possible that these commands may conflict with another Stratus command or abbreviation already set up at a particular site. If that is the case, USER-Access is the suggested alternative command that may be set up at installation time. If USER-Access is not invoked by either of these commands, contact the site administrator.
input-file	is an optional USER-Access input or script file containing USER-Access commands that may be read and executed. When USER-Access completes execution of the input file, the session terminates the Stratus system prompt is displayed.
argument1, argument2, ...	are optional arguments that may be passed as parameters to the input file. Multiword arguments should be enclosed in double quotation marks.
-keyword value	(optional) specifies optional command line keywords that may be given to affect operation of the USER-Access session. The following are valid keywords:
-GLObal	specifies the size in bytes of the global variable environment (displayed as the environment variable "USER_GLOBAL"). The default value is 3000 bytes which should be adequate unless a user session attempts to define a large number of global variables, in which case the GLOBAL switch can be used to increase the space available for global variables.
-HOMEdir	specifies the name of the user's "login" or "home" directory when USER-Access is invoked (displayed as the environment variable "USER_HOME"). Changing this keywords value redefines the location USER-Access uses to locate user startup files.
-OUTput	specifies the name of an output file that is to receive the output from this session.

- ROOTdir** specifies the name of the installed USER-Access root directory containing the site specific message, help, and startup files. This keyword is displayed as the environment variable “USER_ROOT”. There is generally no reason to modify this keyword.
- SEArch** specifies the search path USER-Access follows to locate local initiator startup files. SEARCH is described in more detail in “Local Stratus USER-Access Startup Files”.
- SERvice** specifies an alternative default CONNECT SERVICE name. This keyword is displayed as the environment variable “USER_SERVICE”. The default is “USER”.

USER-Access is invoked for interactive use by typing user at the Stratus system prompt:

```
ready 12:00:00
user
User>
```

The USER-Access prompt in the example above is User>, although USER-Access may be configured to prompt with a different string. The prompt means that USER-Access is ready to accept commands.

Note: There is a nine argument maximum to the input file on the USER-Access command line.

Local Stratus USER-Access Startup Files

When USER-Access is invoked, it attempts to read two startup files on the local host: a site startup file which is actually a partitioned data set member named SCLIENT located in the USER-Access root directory (SITE) data set, and a user startup file (sequential data set) catalogued under in the user’s Stratus prefix called prefix.CLIENT.UA. The site startup file is read first then the user startup file is read. Neither of the startup files is required.

The startup files consist of USER-Access commands. Typically a site administrator will create the site startup file to define basic aliases for general users. The user startup file provides more sophisticated users with a way to define custom aliases and qualifier defaults. User startup files make it possible to override defaults in the site startup file. For example, a simple startup file could contain the lines:

```
* My startup file (this is a comment line)
*
set alias ld local directory
set alias rd remote directory
set local prefix MYHOST:
```

This startup file creates two USER-Access aliases for displaying the local and remote directory listings, ld and rd respectively. It also sets the default USER-Access local prefix to be MYHOST:. After USER-Access is invoked, these new definitions will be read in, whether they are in the site startup file or the user startup file, and become available to the user as soon as the USER-Access input prompt appears.

It is possible to invoke USER-Access by declaring alternative startup files. This is done using the SEARCH qualifier on the command line when USER-Access is invoked. By default, SEARCH is defined as ‘(SITE) (USER)’. By implication, this reads partitioned data set (pds) member SCLIENT from the local USER-Access (SITE) partitioned data set and then *prefix.CLIENT.UA*, in that order. The order can be changed, other file names may be specified, or the special SEARCH keyword (NONE) can be used to override the default. Refer to “Stratus USER-Access SEARCH Keywords (SITE), (USER), and (NONE)” on page 92 for more information.

Remote USER-Access Startup Files

In addition to the local startup files, there are equivalent remote startup files that the USER-Access Responder on the remote host returns to the local Initiator following a successful connection. By default, both a site and user startup files are processed, but this can be overridden by the CONNECT command's SEARCH qualifier. Following a network connection, these startup files, if they exist, are sent back to the Initiator to be processed. They are not executed on the remote host. (For security reasons, the following commands may not be executed from a remote server startup file: CONNECT, DISCONNECT, LOCAL, RECEIVE, REMOTE, and SEND.) Any aliases defined in these files become available to the local user. This is important in that a USER-Access alias defined in the remote startup file will override an alias that has been previously defined in the session. Whether or not this is desirable depends upon the situation; care must be taken when defining aliases in a remote startup file.

For a Stratus host the server site startup file is a partitioned data set named SSERVER in the USER-Access root directory data set, and a user startup file (sequential data set) catalogued under the user's Stratus prefix called *prefix.SERVER.UA*. The site startup file is returned first, then the user startup file. Neither is required.

The exact name and location of the remote startup files depends on the remote host in which a connection is being made. Refer to the manual for the remote host for more information.

Getting Started

Once the prompt appears, it is time to begin giving commands to USER-Access. This section will present some basic concepts that are an important foundation for understanding the details of USER-Access.

USER-Access Commands and Command Qualifiers

A USER-Access command can be invoked anytime the command line prompt appears. Commands may be fully spelled out or abbreviated. The minimum spelling of any command is the first 'n' capital letters of the command name. Abbreviations for each command are shown in "Command Descriptions" on page 115.

Several of the USER-Access commands have qualifiers or keywords associated with them. A command's qualifiers can affect how a command responds to a user, the performance of a command, and the flexibility of a command. Most of the qualifiers have default values already associated with them. The novice user does not need to be concerned with overriding or redefining these values. The sophisticated user can use the qualifiers to modify commands, often making the commands more powerful. There are two methods for changing the values of qualifiers:

1. A qualifier can be redefined to assume a new default value by means of the SET command.
2. The current value of a qualifier can be overridden by specifying a new value on the command line. This is accomplished by using the special character dash '-' followed by the qualifier name and its new value.

Command qualifiers are similar to USER-Access commands in that they may be abbreviated. The minimum spelling of any qualifier is the first 'n' capital letters of the qualifier name. Abbreviations for each command qualifier are shown in "Command Descriptions" on page 115. For instance, the minimum spelling of the qualifier CREate is CRE.

Displaying the Valid Qualifiers for a Command

A list of valid qualifiers for a particular USER-Access command can be obtained with the SHOW QUALIFIER command. The list also includes a brief description of each qualifier. For example, to display the list of valid qualifiers for the INPUT command, type:

```

User> show qualifier input
User:
User:  CONTinue .... continue on error (on/off)
User:  ECHO ..... echo input to terminal (on/off)
User:  PROMPT2 ..... secondary prompt for input continuation
User:  PROMpt ..... prompt string for USER input
User:  SEARch ..... search path for default INPUT commands
User:  VERify ..... verify string/alias substitution (on/off)
User:

```

Displaying the Current Value of a Qualifier

The SHOW command is used to obtain a listing of the current values for a command's qualifiers. For example, a listing of the SEND qualifier values is displayed by entering:

```

User> show send
User:
User:  CRC ..... off
User:  CREate ..... new
User:  * DIRectory:LOCAl ... locdir_value
User:  * DIRectory:REMote .. remdir_value
User:  FLOW ..... off
User:  MAXRECORD ..... value
User:  MODe ..... character
User:  PARTialrecord ..... on
User:  QUIet ..... off
User:
User:  * Informational qualifier (cannot be modified).

```

The qualifier name appears in the left hand column and its value appears in the right hand column. In this example, the value of qualifier CREATE is currently set to 'new'. Qualifier QUIET is turned 'off'. Notice that some qualifiers are flagged as "informational qualifiers". These are shown along with the SEND qualifiers but are not controllable in the same way. They appear because they provide information important to the command and the one using it. Qualifiers flagged as informational cannot be modified. (DIRectory:LOCAl and DIRectory:REMote, shown above, may be modified using SET LOCAL DIRECTORY and SET REMOTE DIRECTORY respectively. The SEND and RECEIVE commands list them as informational qualifiers since they are used to direct file lookup for file transfers.)

An individual qualifier's value can be examined by using the SHOW command followed by the command name and qualifier name. For instance, the current value of the INPUT PROMPT qualifier can be shown by entering:

```

User> show input prompt
User: PROMpt ..... User>

```

Setting a Command Qualifier

Use the SET command to redefine the value of a qualifier for a command for the duration of the USER-Access session or until it is changed again using the SET command. For example, to change the default RECEIVE file transfer mode to STREAM, modify the MODE qualifier of the RECEIVE command:

```
User> set receive mode stream
```

The RECEIVE file transfer mode now will default to STREAM until the qualifier MODE is redefined. The change can be verified with the command:

```

User> show receive mode
User: MODe ..... stream

```


Some command qualifiers, such as INPUT qualifiers CONTINUE, ECHO, and VERIFY, are Boolean qualifiers: their values are either ON or OFF. To set a Boolean command qualifier to ON, enter:

```
User> set command qualifier on
```

or

```
User> set command qualifier
```

For the INPUT qualifier ECHO, this would be:

```
User> set input echo on
```

or

```
User> set input echo
```

For Boolean qualifiers, a missing value is interpreted by USER-Access as ON.

Besides string and Boolean qualifiers, there are also Integer qualifiers. These qualifiers, such as BLOCKSIZE, LINES, and TIMEOUT, accept only integer values and often have numeric range checks associated with them. Integer qualifier values may be appended with a 'K' (210) or 'M' (220) multiplier. For example, to set the CONNECT BLOCKSIZE qualifier to 16 kilobytes, the following may be entered:

```
User> set connect blocksize 16k
```

Overriding a Command Qualifier

The qualifiers that can be defined with the SET command (all non-informational qualifiers), can also be overridden on the command line. For example, if the current RECEIVE file transfer mode is STREAM, it can be overridden for a single transfer by entering:

```
User> receive -mode character sourcefile
```

This command does not change the default value of the MODE qualifier, it simply overrides the default value for the duration of the command. Therefore, the file 'sourcefile' above would be transferred in CHARACTER mode while the default value of RECEIVE qualifier MODE would remain STREAM. This can be verified with the command:

```
User> show receive mode  
User: MODE ..... stream
```

When forcing a Boolean qualifier to ON from the command line, the value ON is optional. For example, the commands shown below are equivalent.

```
User> send -quiet on sourcefile  
User> send -quiet sourcefile
```

USER-Access interprets the missing Boolean value to be ON, even if the default value is OFF.

Online Help

Built into USER-Access is an online help facility that makes it easy for a user to obtain help on a particular command or topic. The help facility also returns useful information on command qualifiers, qualifier defaults, and command examples. To obtain a general USER-Access help display, use the HELP command as follows:

```
User> help
```

The general, or top-level help display will include additional topics in which help can be obtained. For instance, one of the help sub-topics will be the USER-Access command LOCAL. To get additional help on the LOCAL command, one would type:

```
User> help local
```

To get help on qualifiers for the LOCAL command, one would type:

```
User> help local qualifiers
```

It is important to note that some help information resides on remote hosts. Therefore, a remote connection is required in some cases (such as 'HELP SEND QUALIFIERS').

Refer to the HELP command in "Command Descriptions" on page 115 for more details.

Controlling USER-Access Input and Output

The USER-Access commands INPUT and OUTPUT, along with their respective qualifiers, control a majority of the user-oriented input and output within USER-Access. By setting various qualifiers, users can change the USER-Access prompt, tell USER-Access to continue processing even if an error occurs, cause output to be held after each page, save the output to a local file, etc. This section very briefly discusses some of the things that can be done to control USER-Access I/O.

By typing SHOW INPUT, the user can get a list of all INPUT qualifiers along with their current values:

```
User> show input
User:
User:  CONTInue ..... off
User:  ECHO ..... off
User:  PROMPT2 ..... More>>
User:  PROMpt ..... User>
User:  SEARch .....
User:  VERify ..... off
User:
```

Each of these qualifiers is explained in detail in "Command Descriptions" on page 115 under the INPUT command, along with examples of its use. Very simply, the SET command is used to modify any of the qualifiers. For instance, to change the USER-Access prompt from User to MY-PROMPT: type the following:

```
User> set input prompt "MY-PROMPT: "
MY-PROMPT:
```

Notice that the prompt for the next command has now changed to "MY-PROMPT:". To tell USER-Access to continue processing within an input script or alias (discussed later) even after an error results, turn on the CONTINUE qualifier by entering:

```
User> set input continue on
```

Users can affect the output as it is returned from USER-Access by modifying OUTPUT qualifiers. To look at the available qualifiers for the OUTPUT command, type SHOW OUTPUT command:

```
User> show output
User:
User:  COLumns ..... 80
User:  * DESTInation .....
User:  FORmat ..... {msg("text")}
({msg("facility")}{msg("code")}).
User:  HOLD ..... off
User:  LINes ..... 24
User:  PREFix ..... User:
User:  QUIet ..... off
User:  TRUNcate ..... off
User:
User:  * Informational qualifier (cannot be modified).
```

User:

Each of these qualifiers is explained in detail in “Command Descriptions” on page 115 under the OUTPUT command, along with examples of their use. As with the INPUT qualifiers, the SET command can be used to modify any of the OUTPUT qualifiers. For example, to tell USER-Access to pause every 24 lines (the current value of the LINES qualifier), turn on the HOLD qualifier with the following command:

```
User> set output hold on
```

This will prevent general USER-Access output from scrolling off the screen. To modify the number of lines per screen to twenty, change the LINES qualifier:

```
User> set output lines 20
```

The OUTPUT command itself can be used to capture the results of a USER-Access session to a file. This is done by typing OUTPUT followed by a file name. In addition, the user’s input can be captured by turning on the INPUT ECHO qualifier:

```
User> set input echo on
User> output tmpfile
```

Following this command sequence, all input and output for this session is directed to the file named *tmpfile*. If the ECHO qualifier was not turned on, only the command results (output) would be captured. More information concerning INPUT and OUTPUT can be found in the “Advanced Local User’s Guide” on page 47 of this manual. This facility is particularly useful as a means of providing information to Network Executive Software’s technical support personnel regarding questions and problems.

USER-Access Error Messages

USER-Access provides a friendly user interface across many different host types. This includes error messages that are easy to understand. Error messages returned by USER-Access consist of at least a USER-Access level error message followed by an optional host specific error message. All error messages also have an associated error code that can be used to locate additional information in the error message appendices.

An example of a simple “Invalid command” error follows:

```
User> xxxxxx
User: Invalid command `xxxxxx' (UA-4708).
```

The error text is straightforward. The error code (UA-4708) indicates the error is a general USER-Access error with error number 4708.

The next example demonstrates an error resulting from a SEND command that contains a general USER-Access error followed by a host specific USER-Access error and finally an operating system specific error:

```
User> send badfile
User: Failure during CHARACTER mode send (UA-5001).
User: Failed to access file 'badfile' (UA123-8302).
User: OS - file not found (OS-18012).
```

The first error code (UA-5001) indicates that this is a general USER-Access error (UA) with an error number of 5001. The second error code (UA123-8302) says the error is from USER-Access (UA), but generated by the USER-Access product number 123 (or more exactly H123). The actual error number is 8302. The last error code (OS-18012) indicates the error is generated by the operating system (OS or whatever the operating system name might be), with the operating system error number of 18012. The USER-Access error messages are listed in “Appendix A. USER-Access Error Messages for Stratus” on page 169 and in similar appendices in other USER-Access manuals. The general USER-Access errors can be found in any manual. The product specific errors are in the manual for the product indicated by the product number (e.g., UA123 is product H123). Refer to the appropriate manuals for the host operating system for any operating system messages.

It is important to note that a site has the ability to change the error message format and it may not exactly match the examples above. There are, however, three main pieces of information for each message: the message text, the facility generating the message, and the error number. This information should be easy to decipher. If not, see the site administrator.

Aliasing

Much of the versatility USER-Access offers for users is based on a very powerful script-processing or alias capability. Users of the product benefit from aliasing by having special commands, or aliases, defined for them. While a detailed description of the facility is provided in “Advanced Local User’s Guide” on page 47, this brief discussion is provided to give a general familiarity of aliasing without getting lost in detail.

Aliasing provides a means of creating a custom command set for a user or group of users. An alias is nothing more than a new name for a USER-Access command or set of commands. Aliases are useful for creating “shorthand” commands for complex or frequently used USER-Access command sequences. The simplest aliases are one for one translations of an alias name and a USER-Access command. For example, if the user is accustomed to typing a question mark to obtain help in a given application, an alias can be defined very easily using the SET ALIAS command to map ‘?’ to HELP. The new alias may then be viewed with the SHOW ALIAS command.

```
User> set alias ? help
User> show alias ?
User: ? ..... HELP
```

Now, instead of typing HELP to obtain help information, the user can just type ‘?’ at the USER-Access prompt. The commands are considered equivalent by USER-Access. Below is the definition of a much more complicated alias called EDIT which allows a user to use a familiar local editor to edit a remote file.

```
User> set alias EDit { } -
More>>         receive -mode character {1} edit.tmp !
More>>         local -interactive myeditor edit.tmp !
More>>         send -mode character -create replace edit.tmp {1} !
More>>         local delete edit.tmp
```

The basic procedure of the EDIT alias is to transfer the remote file to the local host (RECEIVE), edit the temporary file using the local editor (LOCAL -INTERACTIVE MYEDITOR), send the file back to the remote host when the edit is complete (SEND), and finally delete the temporary file (LOCAL DELETE). The exact syntax and special characters used to define the alias are explained in detail in “Developing USER-Access Scripts Using Input Files and Aliases” on page 75.

To use the alias, the user simply invokes it from the command line like any other USER-Access command. For example, to edit an existing file on the remote host called MYFILE, you type:

```
User> edit myfile
```

USER-Access takes care of the rest. Even though several USER-Access commands are required to actually edit a remote file, the user sees it as a simple EDIT command. This is the real advantage of aliasing.

To display the definition of the EDIT alias, the SHOW ALIAS command is used:

```
User> show alias edit
User: Edit ..... receive -mode character {1} edit.tmp
User:         local -interactive myeditor edit.tmp
User:         send -mode character -create replace edit.tmp {1}
User:         local delete edit.tmp
```

Aliases created within an interactive session are lost when the session is terminated. To create aliases that can be used from session to session, they must be defined within a USER-Access input or script file, or within a

site or user startup file which are read automatically when USER-Access is invoked. Refer to “Developing USER-Access Scripts Using Input Files and Aliases” on page 75 for a detailed description of aliasing.

Terminating a USER-Access Session

To end an interactive USER-Access session type EXIT:

```
User> exit
```

EXIT will disconnect all connections to remote hosts and terminate the current USER-Access session. Any local or remote files that had been opened will be closed. The QUIT command also may be used to terminate an interactive session. Refer to “Command Descriptions” on page 115 for more details on EXIT and QUIT.

Establishing a Connection to a Remote Host

In order to transfer files or execute commands on another host, a network connection must be established. This connection provides a link between the USER-Access Initiator on the local host and the USER-Access Responder on the remote host. There are three ways to make a connection to a remote host; the CONNECT command or the LOGIN and LOGON aliases.

Using CONNECT to Establish a Connection

The CONNECT command allows a user to login to a remote host. The basic format of the command is:

Command	Parameters
CONnect	[qualifiers] host userid password [arguments]

Where:

- qualifiers** This represents optional CONNECT qualifiers that may be added to the command line to override the default- values. The CONNECT qualifiers control such things as NETEX block-size negotiation, the service name in which to connect, and whether or not to display login information.is the name of a remote host as defined in the local network.
- host** is the name of a remote host as defined in the local network.
- userid** is the user name or ID describing a valid user account on that host.
- password** is the associated password needed to login to userid.
- arguments** indicates additional arguments that may be required by the remote host at login time.

Below is an example CONNECT where the host name is 'bluesky', the userid is 'guest', and the password is 'netex':

```
User> connect bluesky guest netex
User: Connected to Service Initiator on host 'BLUESKY'.
=====
Welcome to Operating System - Version 5.0

Today is November 11, 2011 - The system will be down
For testing tonight between 19:00 and 22:00.
Your System administrator
=====
User: Logged in as user 'guest'.
User: Connected to service 'USER015' on host 'BLUESKY'.
```

Following a successful CONNECT, USER-Access returns several informative messages, the exact syntax of which depends upon the host to which a connection is being made. The first message above indicates that an initial network connection was established to the USER-Access Responder (Service Initiator or service 'EFT'). Following that message are several lines of information surrounded by equal signs (====). The information between the equal signs is returned by the remote operating system at login time. This information is not necessarily important to USER-Access but may be to the user logging in. Next is a USER-Access message indicating that a successful login has completed. Finally a message may appear that informs the user of the name of the network service handling the connection.

Besides the additional parameters that can be passed directly to the remote login procedure, the CONNECT command also has several qualifiers associated with it. The use of most of these qualifiers is a function of the remote host. Refer to the User's Guide for the remote host for more information. "Command Descriptions" on page 115 describing the CONNECT command will also assist in the use of this command and its qualifiers.

Since most users would rather be prompted for input and would rather not see their passwords echoed back to the terminal (if possible), it is suggested that the LOGIN and LOGON aliases be used when establishing a remote host connection. This alias is documented in the next section.

Using LOGIN to Establish a Connection

The suggested way for establishing a remote connection is to use the LOGIN or LOGON alias. LOGIN and LOGON are similar to CONNECT but has the advantage of being interactive. Below is a repeat of the example from the previous section but using LOGIN instead of CONNECT:

```
User> login
Hostname? bluesky
Username? guest
Password? _____
Qualifiers?
User: Connected to Service Initiator on host 'BLUESKY'.
=====
Welcome to Operating System - Version 5.0

Today is November 11, 2011 - The system will be down
For testing tonight between 19:00 and 22:00.
Your System administrator
=====
User: Logged in as user 'guest'.
User: Connected to service 'USER015' on host 'BLUESKY'.
```

Notice that LOGIN prompts the user for appropriate login information and that the password was not printed to the terminal. (Whenever possible USER-Access supports NO-ECHO mode to improve security; not all systems provide this mode.) This interface is much more friendly than using CONNECT and can be tailored to the needs of a given site by the system administrator. Following the prompts, the connect proceeds as expected.

Note: Since LOGIN is an alias that can be modified by the site administrator, it may operate differently than the example. However, the overall process should remain similar.

Exchanging Host Information on Connect

To the user, the connect/login process appears fairly straightforward, but to USER-Access, much must be done in order for two hosts to communicate. The issues concerning CONNECT (LOGIN) qualifiers and login are addressed in the Remote User's Guide section of the manual for the host to which the connection is being made. Contained in this section is a general discussion on the information passed by USER-Access that is available to the user. This information may be useful in making decisions once a connection has been established.

Once a successful login has been assured, the USER-Access Responder (the remote server) sends information about itself to the Initiator (the local client) and vice versa. The information, which describes both the remote and local environments, is exchanged in order for the two sides to establish how compatible they are and what functions can be supported. The SHOW command is used to display this information. For instance, to display information describing the local environment, type SHOW LOCAL as:

```

TNPSTRAT> show local
User:
User:  DIRectory ..... Local_Directory_Value
User: * GATEway .....
User: * HOSTCODE ..... Character_Code
User: * HOSTTYPE ..... OS_Type
User:  INTeractive ..... off
User: * NETwork ..... NETEX
User: * PID ..... Process_ID
User:  PREFix ..... OS_Prefix
User: * PRODuct ..... Product_Number
User:  QUIet ..... off
User: * STATus ..... Local_Status
User: * USERname ..... User's_Name
User: * VERsion ..... UA_Version_Number
User:
User: * Informational qualifier (cannot be modified).
User:
TNPSTRAT>

```

The qualifiers that are preceded by an asterisk (HOSTCODE, PID, etc.) reflect environmental data describing the local host and cannot be changed by the user. The remaining qualifiers (DIRectory, PREFix, etc.) that appear are directly tied to the LOCAL command and may be modified to affect that command's execution. (Note that the display above is only a sample of the information that might actually be seen for a particular host).

To display the remote environment's information, use the SHOW REMOTE command:

```

TNPSTRAT> show remote
User:
User: * BLOCKsize ..... Negotiated_Blocksize
User:  DIRectory ..... Remote_Directory_Value
User: * GATEway .....
User: * HOST ..... Host_Name
User: * HOSTCODE ..... Native_Character_Code
User: * HOSTTYPE ..... OS_Type
User: * PID ..... Process_ID
User:  PREFix ..... OS_Prefix
User: * PRODuct ..... Product_Number
User:  QUIet ..... off
User: * SERvice ..... Offer_Name
User: * STATus ..... Local_Status
User: * TRANSlate ..... Current_Translation
User: * USERname ..... User's_Name
User: * VERsion ..... UA_Version_Number
User:
User: * Informational qualifier (cannot be modified).
User:
TNPSTRAT>

```

Again the qualifiers marked by an asterisk describe the remote environment (HOST, PID, etc.) as well as information important to the connection itself (BLOCKsize, TRANSlate, etc.). The remaining qualifiers (DIRectory, QUIet, etc.) are directly associated with the REMOTE command and affect its execution.

Establishing Multiple Host Connections

A USER-Access session may have up to ten host connections at any given time. Although ten may be unrealistic in most applications, it may be desirable from time to time to make a second host connection at the same time another connection is in place. For example, assume the user of the session below has already established a connection from the local host to a remote host named BLUESKY. This first connection can be verified by invoking the SHOW HOSTS command:

```
User> show hosts
User: active ----> (1) Host=BLUESKY User=guest
```

SHOW HOSTS gives a list of all existing connections for the present session. The current “active” connection is flagged. The active connection is the one, if any, that reflects the current remote host. To establish a second connection the LOGIN alias is used as explained in a previous section. For example, to connect to a host named REDSKY, the following command sequence is used:

```
User> login
Hostname? redsky
Username? newuser
Password? _____
Qualifiers?
User: Connected to Service 'USER' on host 'REDSKY'.
=====
          ***** Welcome to Network Host REDSKY *****
                    04 - 02 - 89
=====
User: Logged in as user 'newuser'.
```

The SHOW HOSTS command can be used again to display the list of connections held by this session:

```
User> show hosts
User:                (1) Host=BLUESKY User=guest
User: active ----> (2) Host=REDSKY  User=newuser
```

Notice that REDSKY is now flagged as the active host. This means that any file transfer or remote command execution will be directed to it instead of host BLUESKY. The SHOW REMOTE command also will display the remote environment for host REDSKY since it is now active. The connection to host BLUESKY remains but is in an idle state. To make it the active connection, the SET HOST command is used as:

```
User> set host bluesky
```

or

```
User> set host 1
```

Now a look at the host display will show that BLUESKY is the active host:

```
User> show hosts
User: active ----> (1) Host=BLUESKY User=guest
User:                (2) Host=REDSKY  User=newuser
```

Having multiple host connections can be useful for managing system activities on a number of hosts from a single point. For instance, a user on one host can send messages to a number of other hosts. Or a user can start up jobs on several other hosts all from a single terminal on the network.

Disconnecting from a Host

To terminate an existing connection, the DISCONNECT command is used. Assume two connections are currently established to hosts BLUESKY and REDSKY respectively, where BLUESKY is the active connection. The following will terminate this connection:

```
User> disconnect
User: Disconnected from host BLUESKY.
```

To verify the connection has been broken, use the SHOW HOSTS command:

```
User> show hosts
User:                (2) Host=REDSKY   User=newuser
```

Following a disconnect, there is no active host. In order to make an existing idle connection active, use the SET HOST command. The following command will make the connection to REDSKY active:

```
User> set host redsky
```

SHOW HOSTS will now indicate the change:

```
User> show hosts
User: active ----> (2) Host=REDSKY   User=newuser
```

An alternative way to disconnect from an active host is to exit the USER-Access session. The EXIT command causes all connections to be disconnected prior to terminating the session.

Transferring Files as a Local User

The file transfer capabilities of USER-Access are provided by two commands, SEND and RECEIVE. The SEND command provides file transfer from a user's local host to the current remote host. The RECEIVE command transfers files from the remote host back to the local host. Prior to transferring files, a network connection must exist.

Sending Files to a Remote Host

The basic format of the SEND command is:

Command	Parameters
SEND	src_spec [dest_spec] [qualifiers]

Where:

- src_spec** is the file specification of the local file to be transferred to the remote host.
- dest_spec** is the file specification of the remote file which is to be created or replaced by the transfer. This parameter is optional. If it is omitted, USER-Access will use src_spec to create the destination file specification based on the remote host.
- qualifiers** represents optional SEND qualifiers that may be added to the command line to override the default values. The SEND qualifiers control such things as file creation, mode of transfer, and record orientation, and are defined by the remote host.

Once a connection to a remote host has been established, the user may begin transferring files. This is generally as easy as typing SEND followed by a local file name:

```
User> send src_spec
```

where **src_spec** is the name of an existing file on the local host. USER-Access takes care of mapping the local file name to a valid remote file specification in all but a few instances. If USER-Access cannot successfully handle the mapping, for example if the source file name contains unusual characters that the remote host just cannot tolerate, then the user must include the destination file name on the command line. Specifying the destination name is also useful for changing the name of a file from one host to another. The example below transfers file *src_spec* and renames it *new_file* on the remote host:

```
User> send src_spec new_file
```

The SEND command also supports wildcarding on both the source and destination file specifications. This information along with all of the host specific information concerning file transfers, including examples, is explained in the file handling section of the appropriate manual. Source file specifications, source wildcarding, etc., can be found in “File Handling Under Stratus USER-Access” on page 39. Destination file specifications, destination wildcarding, and qualifiers that affect the SEND command, can be found in the same section of the manual for the host to which files are being transferred.

Receiving Files from a Remote Host

The basic format of the RECEIVE command is:

Command	Parameters
RECEive	src_spec [dest_spec] [qualifiers]

Where:

- src_spec** is the file specification of the remote file to be transferred to the local host.
- dest_spec** is the optional specification of the local file which is to be created or replaced by the transfer. If it is omitted, USER-Access will use *src_spec* to create the destination file specification on the local host.
- qualifiers** represents optional RECEIVE qualifiers that may be added to the command line to override the default values. The RECEIVE qualifiers are defined by the local host. As do the SEND qualifiers, the RECEIVE qualifiers control such things as file creation, mode of transfer, and record orientation.

Files can be received from a remote host as soon as a connection has been established. Receiving a file is as easy as typing RECEIVE followed by a remote file name:

```
User> receive src_spec
```

where *src_spec* is the name of a file that currently resides on the re-mote host. In the same way as it handles SEND, USER-Access maps the remote file name to a valid local file name in all but a few instances which are generally due to character or length conflicts. If the file name mapping cannot be automated, or if the user simply wishes to rename the file as it is received, the local file name must be included as a second parameter on the command line, as shown:

```
User> receive remote_file local_file
```

The example above transfers file *remote_file* from the remote host and renames it *local_file* on the local host.

The RECEIVE command supports wildcarding on both the source and destination file specifications. This information along with all of the host specific information concerning file transfers, is explained in the file handling section of the appropriate manual. Source file specifications, source wildcarding, etc., can be found in file handling in the manual for the remote host. Destination file specifications, destination wildcarding, and qualifiers that affect the RECEIVE command can be found in “File Handling Under Stratus USER-Access” on page 39 of this manual.

Send and Receive Qualifiers

USER-Access was designed to make file transfer very easy for all types of users. Much of the simplicity comes through the use of default qualifier values. Although SEND and RECEIVE have several qualifiers associated with them, defaults can be set up to operate most of the time for most users. Therefore, the majority of users seldom need to modify the qualifier values. On the other hand, changing the value of a SEND or RECEIVE qualifier is simple.

Only simple validation of many of these qualifiers is performed when the set command is issued. Detailed validation is performed when the send or receive is issued.

To show the available SEND or RECEIVE qualifiers after establishing a remote connection, use the SHOW QUALIFIERS command. For example, to display the list of valid qualifiers for SEND, type the following:

```
User> show qualifiers send
User:
User:  CRC ..... file transfer checksum (on/off)
User:  CREate ..... file create options
User:  MAXRECORD ..... maximum RECORD mode size
User:  MODE ..... file transfer mode
User:  QUIet ..... inhibit file transfer display (on/off)
User:
```

The output above reflects a sample of the many qualifiers that might be seen. The actual qualifiers for SEND depend on the remote host since that is where file creation takes place. The RECEIVE qualifiers are directly associated with the local host for the same reason. If a new connection is made to a different host, the qualifiers may change significantly.

To view the current values for the SEND or RECEIVE qualifiers, use the SHOW command. For example, SHOW SEND displays the list of SEND qualifiers along with their current values:

```
User> show send
User:
User:  CRC ..... off
User:  CREate ..... new
User:  * DIRECTORY:LOCAL ...local_dir_value
User:  * DIRECTORY:REMOTE ..remote_dir_value
User:  MAXRECORD ..... 10000
User:  MODE ..... character
User:  QUIet ..... off
User:
User:  * Informational qualifier (cannot be modified).
```

Notice that a DIRECTORY entry appears for both the local and remote host. This value determines where the file will come from and where it will be sent if the respective file specifications are not given. (These qualifiers may be modified by using SET LOCAL DIRECTORY and SET REMOTE DIRECTORY.) The remaining qualifiers (the non-informational qualifiers) may be modified using the SET command. For example, to change the RECEIVE command's default file option CREATE from NEW to REPLACE, use the following:

```
User> set receive create replace
```

Or, to override the current value for a single file transfer, modify it on the SEND or RECEIVE command line. For example:

```
User> receive sourcefile -create replace
```

For a complete list of valid RECEIVE qualifiers, refer to the "File Handling Under Stratus USER-Access" section on page 39 of this manual. This section will also address detailed information about transferring files to this host, wildcard support, transfer modes, and much more. Refer also to the RECEIVE command in "Command Descriptions" on page 115 of this manual.

The qualifiers for the SEND command on the other hand, are detailed in the file handling and command description sections of the manual for the remote host to which file transfers will be made. That manual will also address information concerning host file specifications, wildcard support, file types supported, etc.

Executing Remote Host Commands

USER-Access users can issue host commands on the remote host and view the results. Host commands can take the form of a native host command or an alias that translates to a host-specific command. Remote commands are issued from a USER-Access session via the REMOTE command. A network connection to a remote host must exist prior to issuing REMOTE. The command line format is:

Command	Parameters
REMOTE	[qualifiers] command

Where:

REMOTE is the keyword for this command.

qualifiers represents optional USER-Access REMOTE qualifiers that may be added to the command line to override the default values. These qualifiers and their default values are defined by the remote host. Qualifiers must appear before the remote command.

command either a valid command on the remote host, an alias command defined using SET REMOTE ALIAS, or one of the predefined host independent commands.

USER-Access performs translation on any alias prior to passing the command string to the remote host. By default, the results of a REMOTE command get transferred back across the network and displayed at the local user's terminal.

For example, assume the remote host supports a command called DISPLAY TIME that returns the current time of day. A user could execute this command from a USER-Access session by typing the following:

```
User> remote display time
SYSTEM-A:
SYSTEM-A: The current time is: 12:12:01 pm
SYSTEM-A:
```

The results are displayed in the remote host's format with the exception of the optional host prefix that precedes each line of output (SYSTEM-A:). This prefix can be modified to the user's liking with the SET REMOTE PREFIX command.

Since users may be unfamiliar with the command syntax of a remote host, USER-Access defines a set of commands (implemented as aliases) that exist on all hosts¹. These commands, referred to as host-independent commands, allow a user to execute commands on many different systems with a single, simple command set. To look at the list of host independent commands defined for the current remote host, issue the SHOW REMOTE ALIAS command:

```
User> show remote alias
User:
User: COPY ..... Copy_a_file
User: DElete ..... Delete_a_file
User: DIRectory ..... List_files
```

¹ Some of these commands may not be supported on all hosts.

```

User: RENAME ..... Rename_a_file
User: TYPE ..... Type_contents_of_a_file
User: WHO ..... Who_is_on_the_system
User:

```

The actual output seen by the user will list all of the remote aliases (including host independent commands) in the left column and the host command translations in the right column. Users can issue host-independent commands as if they were commands native to the remote host. USER-Access handles the translation. For example, to obtain a list of files that reside on the remote host, the host-independent command DIRECTORY could be used:

```
User> remote dir
```

The actual native command for the remote host could be given. Assuming the native command for listing files on the remote host is LISTFILES, an alternative to the above would be:

```
User> remote listfiles
```

The commands would give identical results since the host independent command DIRECTORY would be mapped to the native command LISTFILES for this host.

Refer to the remote user’s guide in the manual of the remote host for a list of host independent commands defined for that system, as well as a discussion on executing commands on that host. Also see the command description section of the same manual for the list of valid REMOTE qualifiers and an example of their use.

Executing Local OpenVOS Commands

Users can issue host commands on the local host and view the results. Host commands can take the form of a valid Stratus command or an alias that translates to a valid TSO command. Local commands are issued from a USER-Access session via the LOCAL command. The format of the LOCAL command is:

Command	Parameters
LOCAL	[qualifiers] [command]

Where:

qualifiers represents optional USER-Access LOCAL qualifiers that may be added to the command line to override the default values. Qualifiers must appear before the local command.

command can be a valid Stratus command or CLIST, an alias command defined using SET LOCAL ALIAS, or one of the predefined host independent commands (e.g. DIRECTORY, TYPE, WHO, etc.). USER-Access performs translation on any alias prior to passing the command string to OpenVOS. The default is none, which means to drop into an interactive Stratus sub-process.

By default, the results of a local command get displayed at the user’s terminal.

The figure below is an example of the LOCAL command being used within Stratus USER-Access to obtain a list of what is happening on the system. The OpenVOS command is “list_users -full -kernel -system”.

```

User> local list_users -full -kernel -system
Stratus:
Stratus:      LOGIN TIME          CPU    PF    USER
Stratus:  80-01-01 00:00:00      14:51.7  22    Cache_Manager_Post.System (Cache_Manager_Post)
Stratus:  80-01-01 00:00:00      2:45:52.1  16    Cache_Manager_Timer.System (Cache_Manager_Timer)
Stratus:  80-01-01 00:00:00       3:25.0  81    Diagnostic_Utility.System (Diagnostic_Utility)

```

```

Stratus: 89-09-01 14:12:19 10:05.4 2943 smith.SysAdmin
Stratus: 89-09-01 14:12:04 33:42.8 4308 * smith.SysAdmin
Stratus: 80-01-01 00:00:00 9.3 13 Kernel_Utility.System (Kernel_Utility)
Stratus: 80-01-01 00:00:00 45:21.7 19 Maintenance_Utility.System (Maintenance_Utility)
Stratus: 89-08-22 13:46:02 12:26.0 44 Overseer.System (BatchOverseer)
Stratus: 89-08-22 13:46:20 7.2 34 Overseer.System (insite_monitor)
Stratus: 89-08-22 13:45:45 10:04.0 55 Overseer.System (LinkServer1)
Stratus: 89-08-22 13:45:47 16:17.6 63 Overseer.System (LinkServer2)
Stratus: 89-08-22 13:46:14 39:29.8 278 Overseer.System (MaintServer)
Stratus: 89-08-22 13:45:53 4:52.0 20 Overseer.System (NetworkWatchdog)
Stratus: 89-08-22 13:45:59 41:07.9 58 Overseer.System (TheOverseer)
Stratus: 89-09-02 08:53:48 38.8 180 jones.SysAdmin (netex)
User>

```

The prefix `Stratus:` indicates that the results are being returned from the Stratus host. Using `LOCAL` from within a `USER-Access` session, it is also possible to invoke a compiler, broadcast user messages, run command macros, etc. For example, to send a message to user `guest`, the command would be:

```
User> local send_message guest 'Please Start Noon Job!'
```

To execute a local command under `OpenVOS`, `USER-Access` creates a new process then issues the command under it. Any local commands issued affect the environment of the new process, not `USER-Access`. Therefore, commands such as `LOCAL CHANGE CURRENT DIR` affect future local commands. If such commands are issued under the new environment, `USER-Access` and the new process are no longer synchronized (for example, the local current directories will differ). They can be resynchronized with the `SET LOCAL` command.

```
User> set local directory %nsc#m1_d01>system
```

A display of the local directory will verify the change:

```
User> show local directory
User: DIRectory ..... %nsc#m1_d01>system
```

The new directory value will be used as the default directory for all subsequent `LOCAL` commands since `USER-Access` makes this change to the parent process, not a sub-process. Therefore, the `Stratus` command `DISPLAY_CURRENT_DIRECTORY` results in the following value:

```
User> local display_current_directory
Stratus: %nsc#m1_d01>system
```

Some `OpenVOS` commands require interaction from the user. If that is the case, the `LOCAL` qualifier `INTERACTIVE` must be set when the command is issued. For example, to run a program called `TEST` that prompts for a file name, the `INTERACTIVE` flag would be set as shown in the following example.

```
User> local -interactive test
Enter file name to test? user.tmp
*****
*** the file 'USER.TMP' exists ***
*****
```

The `INTERACTIVE` qualifier tells `USER-Access` to treat the terminal as default input to test. Normally input is discarded.

The `LOCAL` command also gives the user the ability to enter an interactive `Stratus` sub-process, keeping the `USER-Access` session in the background. This local interactive mode can be invoked by leaving the command off of the `LOCAL` command line. The value of the `INTERACTIVE` qualifier is also ignored.

```
User> local
<Type LOGOUT to return to USER-Access>
```

```
ready 12:00:00
```

At this point the user is simply running a Stratus subprocess. Any valid Stratus command can be issued just as if USER-Access had never been invoked. To return back to the USER-Access session, the user must logout of the Stratus subprocess by typing LOGOUT:

```
ready 12:00:00
logout
```

```
User>
```

LOGOUT returns the user back to the USER-Access session, where all remote connections, alias definitions, and the like have been retained. Local interactive mode makes it easy for a user to bring up USER-Access, establish a remote connection, and then return to Stratus command level for further command activity. When a file or remote job is requested, the user simply returns to USER-Access where the remote host is actively waiting. For more information on the LOCAL command and its qualifiers, refer to “Command Descriptions” on page 115 of this manual.

Issuing Local OpenVOS Host-Independent Commands

As on the remote host, the local USER-Access user has the option of executing native host commands, host independent commands, or user defined aliases. The host independent commands allow a user to execute commands on many different systems with a single command set. To display the list of host independent commands defined for Stratus USER-Access, issue the SHOW LOCAL ALIAS command as shown in the following example:

```
User> show local alias
User:
User: CANcel ..... cancel_batch_requests -no_ask
User: COPY ..... copy_file
User: DElete ..... delete_file -no_ask
User: DIFference ..... compare_files
User: DIRectory ..... list -all -full -header
User: HELP ..... help
User: PRInt ..... print
User: QUEue ..... list_print_requests -all -long
User: REName ..... rename
User: STATus ..... list_users -full -kernel -system
User: SUBmit ..... batch
User: TYPE ..... display -no_header
User: WHO ..... list_users -full
User:
User>
```

The host independent commands are in the left column and the Stratus command translations are in the right column. Users can issue host independent commands as if they were commands native to Stratus. USER-Access takes care of the translation.

The following is an example of a LOCAL command that invokes a host independent command called WHO. WHO translates to the Stratus command “list_users -full” which displays a list of users currently logged onto the system:

```
User> local who
Stratus:
Stratus:          LOGIN TIME          CPU  PF      USER
Stratus:    89-08-30 09:53:46    12:53.9  412    smith.development
```



```

Stratus:      89-08-30 09:54:31      11:33.2  301      smith.development (test.pm)
Stratus:      89-08-38 09:54:58      18:37.7  343      smith.development (tp_manager.pm)
Stratus:      89-09-09 13:12:12          52.9   153      white.SysAdmin
Stratus:      89-09-09 14:54:22          9.4    134      * jones.SysAdmin
Stratus:      89-09-01 10:38:51          3.2    146      jones.SysAdmin (xyzd)
Stratus:      89-09-01 10:38:45          6.3    259      jones.SysAdmin (uademo)
Stratus:
User>

```

Notice that the output from WHO is equivalent to the output from `list_users -full`.

```

User> local list_users -full
Stratus:
Stratus:      LOGIN TIME          CPU   PF      USER
Stratus:      89-08-30 09:53:46      12:53.9  412    smith.development
Stratus:      89-08-30 09:54:31      11:33.2  301    smith.development (test.pm)
Stratus:      89-08-38 09:54:58      18:37.7  343    smith.development (tp_manager.pm)
Stratus:      89-09-09 13:12:12          52.9   153    white.SysAdmin
Stratus:      89-09-09 14:54:22          9.4    134    * jones.SysAdmin
Stratus:      89-09-01 10:38:51          3.2    146    jones.SysAdmin (xyzd)
Stratus:      89-09-01 10:38:45          6.3    259    jones.SysAdmin (uademo)
Stratus:
User>

```

Local Stratus USER-Access users can also create their own local aliases using the `SET LOCAL ALIAS` command. For example, to create a local alias called DEV that gives a listing of all Stratus devices (i.e., `list_devices -type all`), issue the following command:

```
User> set local alias dev list_devices -type all
```

Now the `SHOW LOCAL ALIAS` command can be used to display the new alias:

```
User> show local alias dev
User: DEV .....list_devices -type all
```

This new alias is equivalent to the Stratus command `list devices` and is stored along with the local host independent commands. Users can create as many local aliases as desired. To make them available for use in all USER-Access sessions, edit them into a local USER-Access startup file.

Refer to “Command Descriptions” on page 115 of this manual for the list of valid LOCAL qualifiers and an example of their use. Also see the “Advanced Local User’s Guide” for further discussion on host aliases and aliases in general.

Editing Remote Files with an Editor

Using a predefined alias called `EDIT`, users can invoke their favorite Stratus editor to edit files that reside on the remote host in which they are connected. The `EDIT` alias is typically defined in the USER-Access site startup file but can easily be redefined and customized in the user’s startup file. The following is a sample `EDIT` alias that invokes the Stratus ISPF editor.

```

User: show alias edit
User: EDit .. receive -create replace -mode character {1} edit.tmp
User:      local -interactive edit edit.tmp
User:      ask -prompt "Update remote (Y/N)? " -default "Y" yn
User:      {cmp("Yes",yn,"send -create replace edit.tmp {1}")}
User:      local delete edit.tmp

```

To invoke `EDIT`, the user simply types `EDIT` followed by the name of an existing file on the remote host:

```
User> edit rfile
```

In the example, the remote file `rfile` would automatically be transferred to the Stratus system and the ISPF editor would be invoked. The user would then edit the file in the normal way. Once the edit session is over, the user has the option of overwriting the remote file or not. Finally, the local temporary file is deleted. This EDIT alias can be greatly enhanced to address file protection, loss of remote connection, etc. It is up to the site to determine exactly how EDIT should function in each environment.

Refer to the “Advanced Local User’s Guide” on page 47 for more information on developing multicommand aliases.

Interrupting a Command within Stratus USER-Access

To interrupt or terminate a command from executing once it has started, hit the key sequence `< CTRL> < C>`. This is accomplished by holding down the CTRL key (Control) and pressing the C key. This key sequence will cause any USER-Access, local, or remote host command to terminate within a few seconds. Often it is desirable to interrupt a command if the output becomes too lengthy (e.g. a directory listing), or if the operation is no longer wanted (e.g. sending a group of files). Interrupting a command with a single `< CTRL > < C >` will result in the USER-Access prompt being displayed unless an alias or input script is handling interrupts specially. (This is discussed in more detail in the “Advanced Local User’s Guide” on page 47.)

To terminate a USER-Access session that appears to have hung, enter the `< CTRL> < C>` sequence three times in a row. Three consecutive interrupts causes USER-Access to cleanup and exit.

Stratus Remote User's Guide

This section is intended for users who are currently running the USER-Access Initiator from any local host, regardless of operating system, and would like to establish a network connection into a Stratus host. The information provided here and in “File Handling Under Stratus USER-Access” on page 39, should be enough to help a non-Stratus user start being productive in a very short period of time. Stratus users should also reference these sections to become comfortable with how USER-Access operates in the Stratus environment.

Connecting into a Stratus OpenVOS Host

A USER-Access network connection is established into a Stratus environment by means of a process known as the Service Initiator. The task of the Service Initiator is to process Stratus login attempts from remote USER-Access users (Initiators) and, if successful, startup a server process (Responder) that will then communicate with the remote user for the duration of the USER-Access session.

The server process started by the Service Initiator is a logged in, interactive process running under the given username/password as supplied by the remote user. All remote login requests are validated to insure that proper security is maintained. Once the Service Initiator has started the server process, it severs ties with both the remote user and the server process, thus becoming available to service additional remote login requests.

CONNECT Qualifiers Used by Stratus USER-Access

The CONNECT command (which is used by the LOGIN alias) has several qualifiers associated with it, of which some are only used by certain systems to assist in the login process. There are no CONNECT qualifiers of special importance to Stratus.

The following CONNECT qualifiers are ignored by Stratus USER-Access at connect/login time:

- ACCount
- APPLication
- COMmand
- PROFile
- PROJect
- SCRipt
- SECOndary
- SITE

The CONNECT command as described in “Command Descriptions” of the local USER-Access guide lists all applicable CONNECT qualifiers.

Remote Stratus USER-Access Startup Files

After establishing a successful connection into a remote Stratus host, USER-Access executes the startup files as described by the CONNECT SEARCH qualifier on the Initiator. The normal default definition of this qualifier is:

```
User> show connect search
User: SEArch ..... (SITE) (USER)
```

Possible values for CONNECT SEARCH are the keywords (NONE), (SITE), (USER), or any valid remote file specification containing USER-Access commands. The definitions for the special keywords as they relate to USER-Access for Stratus are given below.

- (NONE)** do not process any responder (or server) startup files.
- (SITE)** implies a USER-Access site startup file (partitioned data set member) called SSERVER located in the remote Stratus site partitioned data set. This is explained in more detail in “Stratus USER-Access SEARCH Keywords (SITE), (USER), and (NONE)” on page 92.
- (USER)** implies a USER-Access user-level startup file (sequential dataset) called SERVER.UA that is located in the user’s login Stratus prefix. This is explained in more detail in “Stratus USER-Access SEARCH Keywords (SITE), (USER), and (NONE)” on page 92.

No server startup files are required. If any do exist, USER-Access sends their contents (a sequence of USER-Access commands) back to the local Initiator where they are then executed in the order described by the SEARCH qualifier. The following commands may not be executed from a server startup file:

- CONNECT
- DISCONNECT
- LOCAL
- RECEIVE
- REMOTE
- SEND

Transferring Files to a Stratus Host

The file transfer capabilities of USER-Access are provided by two commands, SEND and RECEIVE. The SEND command provides file transfer from the local host to a remote Stratus OpenVOS host. The RECEIVE command transfers files from a remote Stratus host back to the local host.

The SEND and RECEIVE commands function the same regardless of the host from which they are executed. However, the command qualifiers to SEND and RECEIVE differ depending on the hosts involved. The qualifiers affect how files are stored, transferred, named, etc. For details on the SEND and RECEIVE qualifiers that exist for file transfers to and from a Stratus system, refer to “File Handling Under Stratus USER-Access” on page 39.

Executing Remote OpenVOS Commands

The REMOTE command gives users the ability to execute host commands on a remote Stratus system and view the results. Host commands can be either a native OpenVOS command or a USER-Access remote alias (or host-independent command) having a OpenVOS command translation.

The following is an example of the REMOTE command being used within Stratus USER-Access to check the status of a default print queue. The Stratus command is list_print_requests.

```

User> remote list_print_requests
Stratus:
Stratus:      Name          #    Time    Copies  Size    Options
Stratus: -----
Stratus: list_users.log    04  10:04     01    02    * b_jones.SysAdmin on %nsc#p1.0
Stratus: memo.doc          06  10:05     01    01    normand_st_onge.inter
Stratus: uaccess.list      07  10:04     01    25    b_jones.SysAdmin
Stratus:
User>

```

The prefix **Stratus:** indicates that the results are being returned from the Stratus host. This prefix can be modified or turned off using the SET REMOTE PREFIX command.

Using REMOTE to execute commands on a Stratus host, it is possible to do tasks such as submit a batch job, invoke a compiler, broadcast user messages, delete a file, etc. Any noninteractive, nonscreen oriented OpenVOS command can be issued. Interactive commands that require users to respond to prompts or full screen oriented applications cannot be run through USER-Access using the REMOTE command. It should be noted, however, that many interactive tasks can still be performed by providing an input file containing the requested information.

To execute a remote command under OpenVOS, USER-Access creates a new process then issues the command under it. Any remote commands issued affect the environment of the new process, not USER-Access. Therefore, commands such as REMOTE CHANGE_CURRENT_DIR affect future local commands. If such commands are issued under the new environment, USER-Access and the new process are no longer synchronized (for example, the remote current directories will differ); They can be resynchronized with the SET REMOTE command.

The most likely item for a user to modify is the remote directory default. USER-Access makes this possible with the SET REMOTE DIRECTORY command:

```
User> set remote directory %nsc#m1_d03>system>SysAdmin
```

A display of the remote directory will verify the change:

```
User> show remote directory
User: DIrectory ..... %nsc#m1_d03>system>SysAdmin
```

This new directory value will be used as the default directory for all subsequent REMOTE commands since USER-Access makes this change to the parent process, not a subprocess. Therefore, the Stratus command display_current_dir results in the following value:

```
User> remote display_current_dir
Stratus: %nsc#m1_d03>system>SysAdmin
```

For more information on the REMOTE command, refer to “Command Descriptions” on page 115 of this manual. For more information on the OpenVOS command set, refer to the OpenVOS documentation.

Issuing Remote OpenVOS Host Independent Commands

Although executing Stratus commands from a remote system may be very useful, many times remote users are not familiar with the Stratus command set. Therefore, USER-Access makes a set of Host Independent Commands available for all users around the network to use, without requiring them to learn each host’s command set. These commands are implemented as remote aliases. To display the list of host independent commands defined for Stratus USER-Access, issue the SHOW REMOTE ALIAS command:

```

User> show remote alias
User:
User: CAnCel ..... cancel_batch_requests -no_ask
User: COpy ..... copy_file
User: DElete ..... delete_file -no_ask
User: DIfference ..... compare_files
User: DIRectory ..... list -all -full -header
User: HELp ..... help
User: PRint ..... print
User: QUEue ..... list_print_requests -all -long
User: REName ..... rename
User: STATus ..... list_users -full -kernel -system
User: SUBmit ..... batch
User: TYpe ..... display -no_header
User: WHO ..... list_users -full
User:
User>

```

The host-independent commands are in the left column and the OpenVOS command translations are in the right column. Users can issue host independent commands as if they were commands native to Stratus. USER-Access takes care of the translation.

Below is a list of all standard Stratus host independent commands along with a description of how they are used. From a local USER-Access Initiator, any of these commands can be invoked on a remote Stratus system by means of the REMOTE command.

- CAnCel** cancels a specified job entry in the default Stratus print queue. A specific queue name may be specified, Use the QUEUE alias to display the queue and determine the job entry number. The format is:
- ```
 CANCEL entry [-QUEUE name]
```
- COpy**             copy a Stratus file to another file name or OpenVOS directory. The format is:
- ```
          COPY file_spec1 file_spec2
```
- DElete** delete a file or set of files on the Stratus host. The wildcard character '*' may be used in the file specification in order to delete multiple files. The format of DELETE is:
- ```
 DELETE file_spec
```
- DIfference**       compare two Stratus files and list their differences. The format is:
- ```
          DIFFERENCE file_spec1 file_spec2
```
- DIRectory** display a listing of all the files in the given Stratus directory. The wildcard character '*' can be used to list only select files if desired. The format is:
- ```
 DIRECTORY [directory_name I file_spec]
```
- HELp**             obtain help on an OpenVOS topic. The format is:
- ```
          HELP [OpenVOS topic]
```
- PRint** print an OpenVOS file or set of files to a OpenVOS printer. A specific queue name may be specified. The format is:
- ```
 PRINT file spec [-QUEUE name]
```
- QUEue**            display the current entries in the default Stratus print queue. A queue name can be given as -QUEUE name. The format is:
- ```
          QUEUE [-QUEUE name]
```

REName change the name of the specified Stratus file to the new name specified as the second parameter. The format is:

```
RENAME old_name new_name
```

STatus display a listing of the current activity on the Stratus host. No parameters are required. The format is:

```
STATUS
```

SUBmit submit an existing Stratus job file to the default batch queue. A specific queue name can be given as `-QUEUE name`. The format is:

```
SUBMIT [-QUEUE name]job_file
```

TYPE type out the contents of a Stratus file. The format is:

```
TYPE file_spec
```

WHO display a listing of users that are currently logged on the Stratus host. No parameters are required. The format is:

```
WHO
```

The following is an example of a `REMOTE` command that invokes a host-independent Command called `WHO`. `WHO` translates to the Stratus command “`list_users -full`” which displays a list of users currently logged onto the system, as shown in the following example.

```
User> remote who
Stratus:
Stratus:      LOGIN TIME          CPU   PF      USER
Stratus:  89-11-01 14:30:29        6.5   67     a_west.SysAdmin
Stratus:  89-10-25 07:48:17         3.4   60     a_west.SysAdmin (login_svr)
Stratus:  89-11-01 14:12:31        1:15.5 120    g_smith.SysAdmin
Stratus:  89-11-01 11:44:16       16:09.4 3121   * normand_st.onge.inter
Stratus:  89-10-26 12:28:04         2:30.3 618    r_jones.SysAdmin
Stratus:  89-10-26 11:45:15         2:01.6 350    r_jones.SysAdmin (xyzd)
Stratus:  89-11-01 09:12:07       38:59.5 4943   t_johnson.SysAdmin
Stratus:  89-10-31 15:04:27       4:17:32.7 225    t_johnson.SysAdmin (netex)
Stratus:
User>
```

The output from `WHO` is equivalent to the output that would be seen from the Stratus command “`list_users -full`”.

Refer back to the previous section describing the command `list_print_requests`. This command can also be executed from a remote host using the host-independent command `QUEUE`.

```
User> remote queue
Stratus:
Stratus:      Name          #   Time   Copies  Size  Options
Stratus:  -----
Stratus:  memo.doc           04  14:16   *    03    01  normand_st_onge.inter on %nsc#pl.0
Stratus:  uaccess.log        05  14:17         01    85  g_smith.SysAdmin
Stratus:  daily_log.bck      06  14:17         01    02  r_jones.SysAdmin
Stratus:
User>
```

Users can also create their own remote aliases using the SET REMOTE ALIAS command. For example, to create a remote alias called LISTFILES that gives a Stratus file listing (i.e., *list -all -full -header*), issue the following command:

```
User> set remote alias LIStfiles list -all -full -header
```

Now the SHOW REMOTE ALIAS command can be used to display the new alias:

```
User> show remote alias listfiles  
User: LIStfiles ..... list -all -full -header
```

This new alias is equivalent to the DIRECTORY host independent command and is stored in the same fashion. Users can create as many remote aliases as desired. To make them available for use in all USER-Access sessions, edit them into a remote USER-Access startup file.

Refer to “Advanced Local User’s Guide” on page 47 for further discussion on host aliases and aliases in general.

File Handling Under Stratus USER-Access

This section is intended to address Stratus file handling issues as they relate to the USER-Access file transfer commands SEND and RECEIVE. Both local and remote Stratus USER-Access users should use this section as a reference for transferring files to and from a Stratus host. Prior to reading this section, it is important to understand the following terminology:

- Source Host** Refers to the host in which the source file (of either a SEND or RECEIVE), resides. The source file is the existing file which is being transferred to the destination host.
- Destination Host** Refers to the host in which the destination file (of either a SEND or RECEIVE), will be created or overwritten. The destination file is the new or replaced file that results following a file transfer.

The distinction between Source Host and Destination Host is important since both the SEND and RECEIVE commands always transfer files from the Source Host to the Destination Host. SEND and RECEIVE command qualifiers are usually tied directly to the Destination Host since that is where files get created.

The following section describes the qualifiers for the SEND and RECEIVE commands that exist for file transfers when a Stratus system is the Destination Host.

Stratus File Transfer Qualifiers and Default Values

Below is a list of the SEND and RECEIVE command qualifiers that are available for file transfers when a Stratus system is the Destination Host. That is, when the local host is a Stratus system, the qualifiers listed below pertain to the RECEIVE command (local Stratus is the Destination Host). When the remote host is a Stratus system, the qualifiers listed below are valid for the SEND command (remote Stratus is the Destination Host).

- CRC** (BOOLEAN) indicates whether or not USER-Access should perform a Cyclic Redundancy Check (CRC) as part of the file transfer. When CRC is enabled, a 32-bit CRC is calculated by the sender along with a block sequence number. These are verified by the receiver. The default for this value is OFF.
- CREate** (STRING) indicates to the destination Stratus host how to create the new file. Each host has its own CREATE options and defines a default that represents the “normal” thing to do when creating new files. On Stratus, the “normal” thing to do is create a new file. Therefore, the default CREATE option on Stratus is NEW. Other options are REPLACE, APPEND, DELETE, and BACKUP. REPLACE overwrites an existing file of the same name; if the file exists, the qualifiers ORGANIZATION and RECLENGTH are ignored. APPEND appends the source file to the destination file if it exists, or creates a new file if it does not. DELETE deletes the file if it exists. BACKUP renames any existing file, adding a “.bak” extension before creating the new file.
- FLOW** (BOOLEAN) indicates whether or not USER-Access should enable file transfer flow control. When FLOW is on, every NETEX block to be transferred must be requested by the receiving host. The sender sends a block only when the receiver is ready for one. FLOW exists to prevent unusual NETEX read timeouts during file transfers that can be caused, for example, by an interactive/selective restore from an archive file. (Waiting for an operator to load a tape is another example of when FLOW may be required). Because each block must be requested by the USER-Access receiver, a significant penalty in performance is paid when FLOW is enabled. The default is OFF.

- MAXRECORD** (INTEGER) the maximum allowed record size when transferring files in CHARACTER or RECORD mode. If an attempt is made to transfer a file in CHARACTER or RECORD mode that has records larger than MAXRECORD, the transfer will terminate with an appropriate error message. The default is 10000
- MODE** (STRING) the current file transfer mode. USER-Access for Stratus supports the following modes: CHARACTER, STREAM, RECORD, BACKUP, RESTORE, V1char, and COPY. The value of MODE is used internally by USER-Access to decide how to open and create files being transferred. A mode must be supported by both hosts in order to have a successful transfer. By default, USER-Access creates new files as sequential files, unless STREAM mode transfer has been used, in which case the file is created as a stream file. For more information, refer to “Transfer Modes Supported Under Stratus USER-Access” on page 45. The default is CHARACTER.
- ORGANIZATION** (STRING) the file organization that Stratus will create when doing either a local RECEIVE or a remote SEND. Four file organizations are supported: STREAM, SEQUENTIAL, RELATIVE, and FIXED. The default is SEQUENTIAL except for STREAM mode, for which the default is STREAM.
- PARTIALRECORD** (BOOLEAN) with PARTIALRECORD enabled, records of length greater than the NETEX block size can be transferred. The default is ON.
- QUIET** (BOOLEAN) tells USER-Access whether or not to display informational type messages on file transfer. This value should be set to either ON or OFF. The default is OFF.
- RECLENGTH** (INTEGER) the number of bytes contained in one record. This qualifier is required when creating a fixed or relative file. For relative files, it specifies the maximum record length for that file. RECLENGTH is ignored when creating STREAM or SEQUENTIAL files. There is no default.
- SPACE** (INTEGER) the number of bytes which USER-Access should allocate to the destination file prior to the file transfer. A ‘K’ or an ‘M’ may be appended to the number to represent kilobytes or megabytes, respectively. If SPACE is specified as either a 0 or a value with a leading asterisk (*), USER-Access uses the size of the source file to determine how much space should be allocated to the destination file. Specifying a positive SPACE value without a leading asterisk tells USER-Access to use the larger of the SPACE values or the size of the source file. Generally SPACE is used only for special applications or when the source host cannot determine the size of the source file being transferred. The default is 0.

Definition of DIRECTORY Under Stratus USER-Access

When USER-Access is invoked from either the Initiator or Responder side, the session begins with a default definition of the DIRECTORY qualifier for both LOCAL and REMOTE. The definition of this qualifier under Stratus is the user's current directory. This DIRECTORY value becomes the default for all file transfers when no source or destination pathname is specified. It is also the default for all LOCAL and REMOTE commands issued through USER-Access.

To change the default value of DIRECTORY, the USER-Access commands SET LOCAL DIRECTORY and SET REMOTE DIRECTORY are used. For example, if the remote host is a Stratus system and the remote directory is currently defined as %nsc#m2_d05>old_directory, a new value of %nsc#m2_d04>new_directory can be established as:

```
User> set remote directory %nsc#m2_d04>new_directory
```

To display the new value of the remote directory, the SHOW command is used:

```
User> show remote directory
User: DIRectory ..... %nsc#m2_d04>new_directory
```

All subsequent file transfers and remote commands will then use this new value as a default when no device or directory is specified.

Stratus File Name Specifications

Below is a very brief discussion of file specification syntax for the Stratus. This is provided as an aid to the occasional Stratus user who is interested in transferring files to or from a Stratus OpenVOS system yet is unsure of file specification syntax. Each OpenVOS file name component is case-sensitive.

A file specification under Stratus OpenVOS has the form:

```
%SYS#MSTR>DIR>.>FILE
```

Where:

- SYS** the system name of the Stratus site preceded by a percent sign (%).
- MSTR** the name of the master directory containing the file preceded by a number sign (#).
- DIR** the names of the directories in the path of the file in order, each preceded by a greater than sign (>). **DIR** may contain up to 9 directories with each directory up to 32 characters in length.
- FILE** the name of the file preceded by a greater-than sign (>). The file name may consist of alphanumeric and the ASCII characters:

```
@ [ \ ] ~ ' { | } ~ " $ + , - . / :
```

The file name cannot begin with a hyphen. The file name can consist of up to 32 characters.

A typical USER-Access user would set the default directory on the Stratus then SEND files from it to a remote host or RECEIVE files into it. In this way, one only needs to specify the file name of a file specification. However, a full pathname may also be used.

A link name can also be used where a file specification is called for so long as its definition is a valid file specification.

Stratus File Specification Examples

Some example Stratus OpenVOS file specifications are listed below to help non-Stratus users get a better understanding of their appearance.

```
%nsc#m2_d03>jones>documents
#m2_d01>system>doc>release_notes
development>g_smith>workfile
>distribution>tape_log.89-10-30      (file relative to current disk)
<accounting>balance_sheet.89.Q1     (file relative to current directory)
project.list
```

When the System, Master Directory or Directory portion is missing, the current location is used as the default.

File Transfer Examples from a Local Stratus Host

Example 1

To send file forecast.89 from the current default local DIRECTORY (%nsc#m4_d08>mrktg), to a remote host, the following command would be issued:

```
User> send forecast.89
User: Source                Destination                Size
User: -----
User: %nsc#m4_d08>mrktg>forecast.89          forecast.89          54900
```

The resulting destination file specification depends on the remote host in which the connection is made. If no destination name is specified, the source name is used to construct the destination name, and the file is stored in the current default remote directory. The SIZE indicated in the display represents an approximation of the number of bytes from the source file transferred.

Example 2

To send the executable file test1.pm from #m8_d03> user to another Stratus host with the new name of test1.sav, issue the following:

```
User> send -mode record -reclen 4096 -org fixed test1.pm test1.sav
User: Source                Destination                Size
User: -----
User: %nsc#m8_d03>user>test1.pm              %xyz#m2_d01>arch>test1.sav          69632
```

Notice that the RECLENGTH and ORGANIZATION qualifiers were used to recreate the file on the remote Stratus host as a fixed file with 4096 byte records.

Example 3

To receive the file doc.txt from the remote host, and store it with the same name on the local Stratus host, the following command would be issued:

```
User> receive doc.txt
User: Source                Destination                Size
User: -----
User: doc.txt                %nsc#m8_d03>mrktg>doc.txt          43002
```

The new doc.txt file was created in the default local DIRECTORY (%nsc#m8_d03>mrktg) since no file specification was given.

File Transfer Examples to a Remote Stratus Host

Example 1

To send a local file called MYFILE.TMP from a non-Stratus system to a Stratus host, forcing a CRC on the transfer, the following command is issued:

```
User> send myfile.tmp -crc -org sequential
User: Source                Destination                Size
User: -----
User: myfile.tmp            %nsc#d03>user>jones>myfile.tmp          2300
```

Notice that the ORGANIZATION qualifier was also used to create a SEQUENTIAL file on the Stratus host.

Example 2

To receive a file called START_UP.CM from a remote Stratus host, issue the command:

```
User> receive start_up.cm
User: Source                               Destination                               Size
User: -----                               -----                               -----
User: %nsc#d03>user>me>start_up.cm        start_up.cm                             77200
```

Source Wildcard Support for Stratus File Transfers

Wildcarding is valid on the source file specification for both the SEND and RECEIVE commands. The only wild card supported on Status is the asterisk (*); the question mark (?) is not supported. The asterisk (*) matches zero or more characters. For example, *def matches the strings abcdef, cdef, def.

An example use of source wildcarding for Stratus USER-Access appears below. The example demonstrates sending all the files in the default directory that start with the letter B and have any extension:

```
User> send b*.*
User: Source                               Destination                               Size
User: -----                               -----                               -----
User: %nsc#d17>black.ftn                   BLACK.FTN                                82293
User: %nsc#d17>blue.sam                   BLUE.SAM                                 167461
User: %nsc#d17>brown.doc                  BROWN.DOC                                4365
```

Destination Wildcard Support for Stratus File Transfers

Destination wildcarding is also available on Stratus USER-Access. Destination wildcarding makes it possible to transfer a set of files from one system to a Stratus host, modifying the file or data set names as part of the process. The single character '*' is used to make this happen.

When an '*' is seen as part of the destination file specification, USER-Access replaces it with either the "file name" portion of the source file specification or the "file extension" portion of the source file specification, depending on its position in the destination file specification. For example, to send all of the files with an extension of FTN from some local host to a Stratus host, renaming the files with an extension of FOR, destination wildcarding would be used as:

```
User>.send *.ftn *.for
User: SOURCE                               DESTINATION                               SIZE
User: -----                               -----                               -----
User: ABCDEF.FTN                          %nsc#m1>abcdef.for                       33114
User: SAMPLE.FTN                          %nsc#m1>sample.for                       67261
User: TEST.FTN                             %nsc#m1>test.for                         4277
```

In this example, USER-Access replaced the '*' in the destination file specification with file names (ABCDEF, SAMPLE, TEST) from the source file specifications. The file extension was also renamed from FTN to FOR. It is also possible to append characters around the '*', For instance the destination file specification could have appeared as:

```
User> send *.ftn x*x.ftn
User: SOURCE                               DESTINATION                               SIZE
User: -----                               -----                               -----
User: ABCDEF.FTN                          %nsc#xabcdefx.ftn                       33114
User: SAMPLE.FTN                          %nsc#xsamplerox.ftn                     67261
```

In this example, destination wildcarding was used to modify both the file name and file extension portion of the destination file.

Accessing a Stratus Tape Device

Stratus USER-Access supports direct I/O to a tape device. USER-Access commands that allow Stratus file specifications also allow Stratus port names. Prior to accessing a file, USER-Access checks for the existence of a port with the specified name. If a port exists, the attached object is accessed. Otherwise the specification is assumed to be a file name. By attaching a port to a tape device, prior to invoking USER-Access, data can be transferred to or from tape.

A tape device can be accessed by first attaching a port to the device:

```
attach_port
```

```
----- attach_port -----
port_name:    t_port
path name:    #mt1.0
-hold_open:   no
```

Next use the `set_tape_defaults` command to change any tape defaults. In most cases the system defaults suffice:

```
set_tape_defaults
```

```
----- set_tape_defaults -----
tape_port_name:  t_port
-tape_format:    ansi
-density:
-volume_id:
-owner_id:
-message:
-access_rights:
-file_id:
-file_number:
-translation:
-disposition:
-open_type:
-file_format:
-block_length:
-record_length:
-blocking_factor:
-reel_retension:
```

Before invoking USER-Access mount the tape with the `mount_tape` command:

```
mount_tape
```

```
----- mount_tape -----
tape_port_name:  t_port
-volume_id:
-owner_id:
-message:
-access_rights:
-create_volume:
-reel_retension:
```

The tape device can now be accessed through the port name attached above. Assume the USER-Access CLIENT has been invoked and a connection has been established with a remote host. The file source_file can be transferred directly to tape by specifying the port name t_port as the destination specification:

```
User> receive source_file t_port
```

Similarly, a file can be transferred directly from tape by specifying the port name t_port as the source specification:

```
User> send t_port destination_file
```

The port name t_port is not a valid specification unless the port has been attached and the tape mounted prior to invoking USER-Access.

If the USER-Access SERVER requires access to a tape device, it is necessary to run a standalone USER-Access SERVER (refer to section entitled “Running a Stratus Stand-Alone USER-Access Server”). Prior to invoking the SERVER, a port must be attached to the tape device and the tape device must be mounted. This can be done from the STRATUS command line or within a STRATUS command macro. Below is an example of a command macro that sets up the tape device prior to invoking a stand-alone USER-Access SERVER:

```
UATAPE.CM - USER-Access stand-alone SERVER accessing tape
```

```
!attach_port t_port #mt1.0
!set_tape_defaults -tape_format ibm -access_rights readonly
!mount_tape t_port -access_rights readonly -volume_id M1D01
!uaserver -service TSERV
```

Transfer Modes Supported Under Stratus USER-Access

Stratus USER-Access supports seven modes of file transfer: BACKUP, CHARACTER, COPY, RECORD, RESTORE, STREAM, and VICHAR. A user selects the file transfer mode by setting the SEND or RECEIVE qualifier MODE. The mode must be supported by both hosts for a successful file transfer. The MODE qualifier defines the form in which data will be transferred between two actively connected hosts. Keep in mind that the internal representation of data within a file varies from host to host even though most hosts define the same modes of transfer. Each mode is described in further detail below as it relates to Stratus.

Note: A more detailed description of each transfer mode is given in “Advanced Stratus Transfer Modes” on page 97.

BACKUP mode is designed to allow Stratus files to be backed up to some other host and then restored, with full characteristics, at some later time. A special header is built around the resulting file in order to properly restore the file and its original characteristics. BACKUP mode is especially instrumental to USER-Access for supporting Central Archiving. Central Archiving allows Stratus files, directories, or disks to be backed up across the network, in a single container me, on to another host's disk or tape drive. Stratus Central Archiving uses the standard Stratus Save/Restore utility to read or write the data. The resulting file will generally not be intelligible on the destination host, but can be restored to a Stratus system as it originally existed.

CHARACTER mode file transfers are generally designed for moving text files from one host to another. This mode performs automatic code conversion across the network and assumes the data being transferred contains only text data. When a Stratus is the destination host, sequential files are created by default.

COPY mode is designed for peer-to-peer file transfers. In conjunction with Stratus' SAVE/RESTORE utility, archived data can be transferred and restored on a destination Stratus with one file transfer.

RECORD mode transfers are designed for moving record oriented binary data. As in STREAM mode, no code conversion is performed on the data. In RECORD mode, the qualifier MAXRECORD determines the maximum allowable record that can be read from or written to a file. The qualifier RECLENGTH is required

to determine record size of the created file. When a Stratus is the destination host, sequential files are created by default.

RESTORE mode is used to restore a file previously transferred in **BACKUP** mode. **RESTORE** mode expects to find a backup header built around the file it is attempting to restore. Central Archiving also uses **RESTORE** mode to list or restore a previously saved container file. Refer to the discussion on **BACKUP** mode above.

STREAM mode file transfers are generally designed for moving files that contain block oriented binary data. **STREAM** mode files are transferred as an unstructured stream of bytes, without record orientation, although the data may contain record headers. No code conversion is performed on the data for **STREAM** mode transfers. When a Stratus is the destination host, stream files are created by default.

VICAR mode is provided for compatibility with Release 1 versions of **USER-Access**. It is equivalent to **CHARACTER** mode under Release 1.

Advanced Local User's Guide

Introduction

This section is intended for users who already have a good working knowledge of USER-Access and would like to learn more details about the product. Site administrators responsible for USER-Access as well as those users developing USER-Access scripts and aliases will benefit most from this section.

The majority of this section discusses how to develop a custom USER-Access interface through the use of string functions, input scripts, and aliases. The remainder of the section discusses advanced topics such as user-definable help files and USER-Access batch jobs.

Special Characters

Several characters have special meaning to USER-Access when it is parsing a command line. The position of the character within a line is a determining factor on how USER-Access will interpret it. The characters are:

- * The asterisk is treated as a comment character if it appears as the first character on the command line. That is, USER-Access ignores the line. An alternate comment character is the pound sign character ('#' in the United States). Comments are generally used within USER-Access alias definitions and input files to make them more readable for the user. The following are example USER-Access comment lines:

```
User> * This is a comment and is ignored
```

- # The pound sign is identical to the '*' character as described above.

```
User> # The pound sign is treated as a comment too
```

- The dash character has two meanings within USER-Access. First, if it appears as the last character of a command line, it tells USER-Access to continue the command on the next line. USER-Access then prompts for more input. For example:

```
User> set alias example -  
More>> text Example of continuing a command on next line.
```

The second use of the dash character is to specify a qualifier to a USER-Access command. A qualifier must follow the dash without any spaces between the two. For example, to turn on quiet mode on the SEND command, the user would specify the QUIET qualifier as below:

```
User> send -quiet source_file destination_file
```

To tell USER-Access to take the dash literally on a command line, escape it by typing two dashes in a row (i.e., '--').

- ! The exclamation point is used by USER-Access as the escape character for special command line processing. Depending upon its position within a command line, it is interpreted several different ways. First, the exclamation point is used to create multicommand aliases when it appears as the last character on the command line (with no trailing spaces). For example, to create a two command alias called NAME, the exclamation point is used as follows:

```
User> set alias NAME { } ask -prompt "Enter Name: " name !  
More>> text Hello {name}.
```

Second, the exclamation point is used to escape the ‘{’ and ‘}’ characters. An exclamation point appearing immediately before either of these characters tells USER-Access to take them literally and skip any string processing that would normally be done. For example:

```
User> text Leave the braces !{here!}.
```

Finally, the exclamation point is used to tell USER-Access not to do any alias processing on a given command. Since an alias may have the same name as a USER-Access command, an exclamation point immediately preceding a command tells USER-Access to use the command, not the alias. The same holds true for local and remote command aliases. If an exclamation point appears immediately before a command preceded by LOCAL or REMOTE, USER-Access uses the command as it appears without processing it as an alias. Each of the following lines in the example below tells USER-Access to use the command even if an alias by the same name has been defined:

```
User> !text ignore alias processing on text
User> local !dir
User> remote !who.
```

{ This character marks the beginning of string substitution. It is used along with the ‘}’ character to delimit a positional parameter, a string variable, or a string function. For example, to print out the value of string variable NAME, the following command could be issued:

```
User> text Your name is {name}.
```

To tell USER-Access to take either the ‘{’ or ‘}’ literally, use the exclamation point:

```
User> text Print the line with braces !{name!}.
```

To tell USER-Access to turn off string substitution, the sequence ‘{}’ is used:

```
User> text {} Turn off string substitution {name}.
```

} This character marks the end of string substitution. See the explanation of ‘{’ above.

“ The double quote character allows the user to create a string that contains embedded blanks:

```
User> set input prompt "NEW PROMPT> "
```

To escape the double quote character, type two in a row (i.e., “”).

USER-Access String Substitution

USER-Access string substitution gives users the ability to write complex aliases and input scripts. String substitution can take place anywhere within a USER-Access command line. The syntax is:

```
{string}
```

where string is either a string literal, string variable (including positional parameters), or a string function. String substitution involves the replacement of {string} by its computed value. The result, or replaced value, of string substitution is always a string.

A string literal refers to any quoted string. The following are examples of a string literal:

```
" "
"Box"
" Big Box "
"This is a Big Box"
```

Performing string substitution on these string literals within a USER-Access TEXT command produces the following results:

```
User> text {" " }.
```

```

User: .

User> text {"Box"}.
User: Box.

User> text {" Big Box "}.
User: Big Box .

User> text {"This is a Big Box"}.
User: This is a Big Box.

```

A string variable, also referred to simply as a variable, is an arbitrary name that is associated with a predefined character string value. Assume the following string variables exist and are defined as indicated:

<u>Variable</u>	<u>Definition</u>
Hostname	BLUESKY
A	Sample string
Day	28

String substitution involves the replacement of a string variable by its currently assigned value. Therefore, performing string substitution on these variables within the TEXT command, produces the following results:

```

User> text {hostname}.
User: BLUESKY.

User> text {a}.
User: Sample string.

User> text {day}.
User> 28.

```

A string function refers to one of the USER-Access defined functions that may accept strings as parameters and return a string as a result. A few simple string functions with sample arguments appear below:

```

date()
upper("this is a test")
cmp("good", "bad", "Compared", "Didn't compare")

```

Performing string substitution on these example string functions result in the following:

```

User> text {date()}.
User: Sun Apr 2, 1989.

User> text {upper("this is a test")}.
User: THIS IS A TEST.

User> text {cmp("good", "bad", "Compared", "Didn't compare")}.
User: Didn't compare.

```

Although the TEXT command was used in all of the examples above, string substitution can be performed anywhere within a USER-Access command line, whether it is part of another USER-Access command, or on a line by itself. It's important to remember that the result of any string substitution is simply another string. Therefore, the resulting string could even be a USER-Access command.

String Variables

USER-Access variable names can be from one to twenty alphanumeric characters long, including underscores and similar special characters. There are two types of variables, local and global. A local variable exists only within the input level, or input file, in which it was initially defined. If an input file is nested, it cannot refer-

ence local variables defined by its caller. A local variable defined within an input file is no longer valid after that input file is exited.

A global variable can be defined from any input level, or input file, and referenced by any other one. That is, once a global variable is defined within a USER-Access session, that variable is known throughout the session, regardless of the current input level. Generally it is better to use local variables whenever possible since these do not get left around from input file to input file. Global variables, on the other hand, take up USER-Access internal storage and can eventually lead to an "Environment overflow" condition. This condition may be relieved by undefining some previously defined global variables as described later in this section. This will recover internal storage space, even though the undefined variable will still be displayed with a null value.

Variables can be defined in a couple of ways. The most obvious is with the SET VARIABLE and SET GLOBAL commands. SET VARIABLE is used to define a local variable. SET GLOBAL defines a global variable. An example of each of these appears below:

```
User> set variable username smith
```

and,

```
User> set global days Saturday and Sunday
```

In the first case, local variable username was given the value 'smith'. In the second case global variable days was assigned the value 'Saturday and Sunday'. Keep in mind, all variables are defined as character strings. To show the current value of the variables defined above use the SHOW VARIABLE and SHOW GLOBAL commands respectively:

```
User> show variable username
User: USERNAME ..... smith
```

and,

```
User> show global days
User: DAYS ..... Saturday and Sunday
```

To undefine a local or global variable, use the SET command with the variable name and no value. For example, to undefine the two variables described above, use the following commands:

```
User> set variable username
```

and,

```
User> set global days
```

An undefined variable will appear in a SHOW VARIABLE or SHOW GLOBAL display as a variable without a definition. If an undefined variable is referenced within a USER-Access command, a null string is substituted in its place.

The second way to define a local variable is with the ASK command. The example below defines the variable username again, but using ASK:

```
User> ask -prompt "Enter Username: " username
Enter Username: smith
```

The real significance of string variables is the ability to use them within USER-Access aliases and input scripts. To reference the value of a variable, enclose the variable name in braces {} within a USER-Access command line (this invokes string substitution). Refer back to the variables username and days above. Their values can be used in the TEXT command as:

```
User> text The value of variable username is {username}.
User: The value of variable username is smith.
```

and,

```
User> text {days} are coming soon.
User: Saturday and Sunday are coming soon.
```

The braces around the variable name tell USER-Access to replace it with its assigned value.

Since local and global variables are stored differently within USER-Access, it is possible to create a global variable with the same name as a local variable. For example:

```
User> set variable hostname alpha
User> set global hostname omega
```

The variable `hostname` has been defined twice, once as a local variable with a value of ‘alpha’ and again as a global variable with a value of ‘omega’. Because USER-Access gives precedence to local variables, referencing `{hostname}` will result in the local value of ‘alpha’. A special syntax is used to reference the value of a global variable when a local variable of the same name exists. An example follows:

```
User> text The local value is {hostname}.
User: The local value is alpha.
User> text The global value is {hostname:global}.
User: The global value is omega.
```

By default, when a variable is enclosed in braces (without the ‘:global’ syntax), USER-Access looks for a local variable by that name. If one is found, its value is returned. If one is not found, USER-Access looks next for a global variable by the same name and uses its value if found. Appending the variable name with ‘:global’ within the braces tells USER-Access not to look for a local variable but instead look immediately for a global variable of that name.

USER-Access carries this special syntax one step further in allowing the substitution of USER-Access command qualifier values. These values can be used as variables as shown below. The syntax is:

```
{qualifier:cmd}
```

where `qualifier` is a valid qualifier (including informational qualifiers) for the specified command `cmd`. For example, if the current value of the SEND qualifier ‘CREATE’ was defined to be ‘new’, this could be referenced as:

```
User> text SEND qualifier CREATE is {create:send}.
User: SEND qualifier CREATE is new.
```

The following example shows how the command qualifiers can be used to set the USER-Access input prompt. Since the prompt is controlled by the INPUT command qualifier PROMPT, this can be modified to the users liking. To change the prompt from the default of ‘User> ’ to the current name of the remote host (assume it’s called STARMAN), the following command is used:

```
User> set input prompt {} {host:remote}>
STARMAN>
```

The syntax `{host:remote}` says to extract the value of informational qualifier HOST from the REMOTE command defaults, and replace this value on the command line. (The empty `{}` is explained in the section entitled “Disabling String Substitution” on page 74.) The following command produces an equivalent result:

```
User> set input prompt STARMAN>
STARMAN>
```

Although the result is equivalent, the second example above does not allow for flexibility within an alias or input script, nor is it flexible enough to change for each connection made to a different remote host.

String Literals

A string literal as mentioned earlier, is any quoted string. Quoted strings refer to a string of characters, enclosed in double quotes, from zero to *n* long, where *n*. is arbitrarily long depending on the space remaining in the USER-Access input buffer.

Below is an example of a string literal used with string substitution and the string function LOWER() (described in “LOWER and UPPER Functions” on page 68) to define the USER-Access prompt as command? with two leading and two trailing spaces:

```
User> set input prompt {lower("  COMMAND?  ")}
```

This will result in the following prompt, including the two leading and two trailing spaces:

```
command?
```

In order to have embedded double quotes within a string literal, the user must escape each one with a second double quote. The example below shows this:

```
User> set input prompt {lower("\"Enter a Command:\" " )}
```

The resulting prompt would be:

```
"enter a command: "
```

with a single trailing space.

The examples above demonstrate the use of string literals within the SET command. Quoted strings within a TEXT command, however, are taken literally. Therefore, to display the same “enter a command:” with double quotes using the TEXT command, the following syntax is used:

```
User> text "enter a command:"  
User: "enter a command:"
```

String Functions

USER-Access string functions perform certain predefined tasks and return a string as output. String functions perform such tasks as comparing two strings, forcing a string to upper/lower case, returning the status of the previous command, and sleeping for a predetermined amount of time. Some string functions require arguments and some do not. All arguments passed to a string function must be a numeric constant, a string literal, a string variable, or another string function. For example, the string function *lower()* takes a single argument which is a string that will be forced to all lower case characters. The following are all valid arguments:

```
lower("sample string")    - a string literal  
lower(hostname)           - a variable named hostname  
lower(date())             - a string function date()  
lower(ext(time()),1,5))   - string functions with numeric constants
```

USER-Access performs string substitution from the inside out. Therefore, if a string function exists as an argument to another string function, the innermost string function is executed first, and the resultant string is passed as an argument to the outer string function. In the example *lower(date())* above, the *date()* function would get processed first, then the actual date string would be passed as an argument to *lower()*.

The greatest use of string functions comes within USER-Access scripts (input files or aliases). Often it is desirable to perform a particular USER-Access command based on a certain condition. String functions make this possible. The following is an example of a simple script that tests the results of a command with the *status()* function and operates accordingly:

```
set input continue on  
*
```

```

*      Loop until successful connection.
*
again:
ask -prompt "Hostname? " host
connect -quiet {host}
{eqs(status(), "S", "text Connect worked.", "goto again")}

```

This script also makes use of the *eqs()* function. *Eqs()* compares the result of *status()* with the string “S” (Success). If the strings compare (i.e., if the CONNECT was successful), the third argument of *eqs()* is used to replace the function in the substitution. If the strings do not compare, the fourth argument is used.

Notice that these last two arguments are simply USER-Access commands enclosed in double quotes. The third argument “text Connect worked.” prints a message at the user’s terminal and continues processing. The fourth argument “goto again”, causes processing to loop back to the ‘again:’ label where the user is prompted for a new hostname. The GOTO command is discussed in a later section.

The remainder of this section describes the USER-Access string functions in more detail. The functions are listed in alphabetical order except where functions are grouped by a logical association (for example, arithmetic operations). Each function is indexed individually.

The descriptions assume the user is familiar with USER-Access strings and string variables as described in the section entitled “String Variables” on page 49 is a list of the available string functions. Most of these functions follow the table in alphabetical order, however the arithmetic and logical functions are grouped together.

Table 2. List of Functions		
Function	Description	Page
ADD	Adds two numeric string expressions and returns the result.	55
CHR	Returns a single character represented by the specified number in the local host machine’s native character set (ASCII or EBCDIC).	56
CMP	Compares two strings. Allows for partial string match by specifying the required characters in upper case.	57
DATE	Returns the system date of the local host.	58
DEC	Subtracts one from a numeric string expression and returns the result.	59
DFN	Tests if a variable is defined.	60
DIV	Divides the first numeric string expression specified by the second and returns the result.	55
ENCRYPT	Used to encrypt user passwords which will be used by USER-Access to establish connections to remote hosts.	61
ENV	Returns the value of the local host environment variable if the local host supports such variables. If local host environment variables are not supported or if the specified variable is not defined, a null string is returned.	62
EQ	Tests if the first number specified is equal to the second number specified.	67
EQS	Tests if the first string specified is equal to the second string specified.	63
EXT	Extracts and returns a bounded sequence of characters from a string.	64
GE	Tests if the first number specified is greater than or equal to the second.	67

Table 2. List of Functions

Function	Description	Page
GT	Tests if the first number specified is greater than or equal to the second.	67
INC	Tests if the first number specified is greater than the second.	55
INDEX	Returns the position of the second string specified within the first string. The function returns zero if the second string is not found.	65
LE	Tests if the first number specified is less than or equal to the second.	67
LEN	Returns the count of characters that make up the specified string.	66
LOWER	Returns the lower case equivalent of a string expression, all characters but upper case are left untouched.	68
LT	Tests if the first number specified is less than the second.	67
MOD	Returns the remainder of the division of the first numeric string expression specified by the second.	55
MUL	Multiplies two numeric string expressions and returns the result.	55
MSG	Returns the information of the last message. MSG is most often used to tailor the USER-Access output message format	69
NDF	Tests if a variable is not defined.	60
NE	Tests if the first number specified is not equal to the second.	67
NES	Tests if the first string specified is not equal to the second.	63
PARAMS	Substitutes the positional parameters specified. It is important to note that quoted parameters remain quoted and are considered one whole parameter – regardless of imbedded spaces.	70
SLEEP	Causes USER-Access to pause or “sleep” for a specified number of seconds. This process is not interruptible. This function results in a null string	71
STATUS	Returns the single status character of the previous command: S = Success, E = Error.	72
SUB	Subtracts the second numeric string expression specified from the first and returns the result.	55
TIME	Returns the system time of the local host.	73
UPPER	Returns the upper case equivalent of a string expression, all characters but lower case are left untouched.	68

Arithmetic Operations

The following is a list of arithmetic operators.

ADD add two numeric string expressions and return the result.

DIV divide the first numeric string expression specified by the second and return the result.

MOD return the remainder of the division of the first numeric string expression specified by the second.

MUL multiply two numeric string expressions and return the result.

SUB subtract the second numeric string expression specified from the first and return the result.

Format:

```
add(number1, number2)
div(number1, number2)
mod(number1, number2)
mul(number1, number2)
sub(number1, number2)
```

Where:

number1, number2 numbers to be operated on.

Examples:

Add two constants and return the result:

```
User> text {add(5,10)}
User: 15
```

Ask the user to enter three numbers:

```
User> ask -prompt "Enter 3 #'s: " num1 num2 num3
User: Enter 3 #'s: :usr.2 3 4
```

Find the square of the first number:

```
User> text The Square of {num1} is: {mul(num1,num1)}
User: The Square of 2 is: 4
```

Find the total sum of the three numbers:

```
User> text {num1}+{num2}+{num3} {add(num1,add(num2,num3))}
User: 2+3+4 = 9
```

Divide the third number by the first number:

```
User> text {num3}/{num1} = {div(num3,num1)}
User: 4/2 = 2
```

CHR Function

The CHR function returns a single character represented by the specified number in the local host's native character set (ASCII or EBCDIC).

Format:

<code>chr(number)</code>

Where:

number a number corresponding to the host's native character set.

Examples:

To display the quote character on a system on an ASCII host:

```
User> text This is a quote {chr(34)}  
User: This is a quote "
```

To display the quote character on a system on an EBCDIC host:

```
User> text This is a quote {chr(127)}  
User: This is a quote "  
User> text This is a quote {chr(0x7F)}  
User: This is a quote "
```

CMP Function

The CMP function compares two strings. It allows for partial string match by specifying the required characters in upper case.

Format:

```
cmp(string, key, if_true [, if_false])
```

Where:

- string** a string expression whose letters are compared with the argument key.
- key** a string expression defining the letters required for partial string match. Upper case letters define the minimum required spelling. Key is used to validate the argument string.
- if_true** a string expression whose value the function takes if the test is successful.
- if_false** an optional string expression whose value the function takes if the test fails. If this argument is omitted, the function takes on the value of a null string.

Examples:

Ask the user for a Yes/No response, and compare the reply with the key “Yes”. Require the user to type at least “Y”:

```
User> ask -prompt "Yes/No? " reply
User: Yes/No? y
User> text {cmp(reply,"Yes","YES","NO")}
User: YES
```

Ask the user for a Yes/No response, and compare the reply with the key “YES”. Force the user to type the entire word “YES”:

```
User> ask -prompt "YES/No? " reply
User: YES/No? yes
User> text {cmp(reply,"YES","YES","NO")}
User: YES
```

In order to not compare to the key, do the same commands, but reply to the prompt with text that doesn’t meet the minimum spelling requirements:

```
User> ask -prompt "YES/No? " reply
User: YES/No? y
User> text {cmp(reply,"YES","YES","NO")}
User: NO
```

DATE Function

The DATE function returns the system date of the local host.

Format:

```
date([number])
```

Where:

number this is a number that specifies the format of the date. For the DATE function:

0 = WWW MMM DD, YYYY

1 = YYMMDD

where W = Weekday; M = Month; D = Day; Y = Year. Specifying a value other than 0 or 1 returns a null value. The default is 0.

Examples:

Display the system date:

```
User> text Today is {date()}  
User: Today is Thu Mar 16, 1989
```

Display the system date in YearMonthDay format:

```
User> text Today is {date(1)}  
User: Today is 890316
```

DEC and INC Functions

DEC subtract one from a numeric string expression and return the result.

INC adds one to a numeric string expression and return the result.

Format:

<pre>dec(number) inc(number)</pre>

Where:

number number to be operated on.

Examples:

Display the results of incrementing “5”:

```
User> text Increment 5 = {inc(5)}
User: Increment 5 = 6
```

Assume the variable CNT exists and has a value of 12. The value of CNT minus one can be displayed as follows:

```
User> text New cnt = {dec(cnt)}
User: New cnt = 11
```

This procedure did not change the actual value of CNT; it only displayed the decremented value. To decrement the value of the variable CNT, use the SET VARIABLE command as shown:

```
User> text cnt = {cnt}
User: cnt = 12
User> set variable cnt {dec(cnt)}
User> text cnt = {cnt}
User: cnt = 11
```

DFN and NDF Functions

The DFN function tests if a variable is defined. The NDF function tests if a variable is not defined.

Format:

```
dfn(variable , if_true [ , if_false ])  
ndf(variable , if_true [ , if_false ])
```

Where:

variable a string variable that is to be tested. A string is considered to be undefined if it has no value (i.e., null string).

if_true a string expression whose value the function takes if the test is successful.

if_false an optional string expression whose value the function takes if the test fails. If this argument is omitted, the function takes on the value of a null string.

Examples:

To find out if a variable is defined, such as the variable COUNT:

```
User> text {dfn(count,"YES","NO")}
```

If COUNT was defined, the command would result in the following display:

```
User: YES
```

Set the input prompt to be the remote host variable if it is defined; otherwise use the string literal "User":

```
User> set input prompt {} {dfn(host:remote,host:remote,"User")}>
```

If the remote host name is BLUESKY, the prompt would be:

```
BLUESKY>
```

NDF can also be used to set the input prompt to be the remote host variable or to the string literal "User":

```
User> set input prompt {} {ndf(host:remote,"User",host:remote)}>  
User>
```

If the remote host name is BLUESKY, the prompt would be:

```
BLUESKY>
```

Define an alias "PRINT" to output the first parameter passed to it, or to output "no parameter" if no parameter is passed.

```
User> set alias print {} {ndf(1,"text no parameter","text {1}")}
```

Execute the alias with no parameters:

```
User> print  
User: no parameter
```

Now execute the alias with the parameters "this text":

```
User> print this text:  
User: this
```

ENCRYPT Function

The ENCRYPT function can be used to encrypt user passwords for later use by USER-Access when establishing connections to remote hosts. This eliminates the security risk of having readable (clear-text) passwords in files.

Format:

```
encrypt(password, [username])
```

Where:

- password** Specifies the password you want to encrypt. The encrypted form of this password is returned by the ENCRYPT string function. The encrypted form can be stored in script files containing USER-Access CONNECT commands.
- username** Optionally specifies the user name associated with the local USER-Access process that will issue the CONNECT command. This user name is used as a secondary encryption key for the specified password. When USER-Access is later run it queries the operating system for the user name running the current process. USER-Access then uses this user name as one of its keys in decrypting the password. The value for user name must be entered in uppercase to match the user name value returned by the Stratus operating system. A value of '*' (single asterisk) tells the USER-Access ENCRYPT function to use the current user name running the USER-Access process as the secondary key. You must be running as the same user which will later run USER-Access to issue the CONNECT command.

Example:

For example, encrypt the password COBRA using the Stratus user name GEM1 as the local username for secondary encryption. Use the USER-Access TEXT command to display the encrypted results:

```
User> text {encrypt("cobra", "GEM1")}  
User: *249eece8e4203b189
```

ENV Function

The ENV function returns the value of the local host environment variable if the local host supports such variables. If local host environment variables are not supported or if the variable is not defined, a null string is returned. In Stratus USER-Access, the value of an environment variable is obtained from the Stratus CLIST variable pool.

Format:

<code>env(variable)</code>

Where:

variable a local host environment variable. This variable may be upper/lower case sensitive. The definition such a variable depends on the local host.

Examples:

Assume a variable “bite” is defined to be “apple” in the local host’s environment. Display the value of the host environment variable with ENV:

```
User> text {env("bite")}  
User: apple
```

To return the host environment variable “SYSUID”, the Stratus user ID:

```
User> text {env("sysuid")}  
User: GUEST1
```


EQS and NES Functions

The EQS function tests if the first string is equal to the second. The NES function tests if the first string is not equal to the second. These tests are case sensitive, so “Ab” is not equal to “AB”.

Format:

```
eqs(string1, string2, if_true [ , if_false])
nes(string1, string2, if_true [ , if_false])
```

Where:

- string1, string2** these can be variable or literal string expressions.
- if_true** a string expression whose value the function takes if the test is successful.
- if_false** an optional string expression whose value the function takes if the test fails. If this argument is omitted, the function takes on the value of a null string.

Examples:

Assume when connecting to most systems with a Username “GUEST”, a password is usually not required. Set the Username variable “usr” to the text “person”. Then, if the Username is NOT “GUEST”, ask for a Password:

```
User> set var usr person
User> {nes(upper(usr), "GUEST", "ask -prompt " "Password? " " pass") }
Password? _____
```

Set the Username variable “usr” to the text “guest”. Then, if the Username is NOT “GUEST”, ask for a Password:

```
User> set var usr guest
User> nes(upper(usr), "GUEST", "ask -prompt " "Password? " "pass") }
User>
```

In most cases, when using a Username “GUEST” to make a connection, if a Password is required (and known) it can be automatically set to the correct input. To try this, set the Username variable “usr” to the text “guest”. Then, if the Username is “GUEST”, set the password to “NETEX”:

```
User> set var usr guest
User> {eqs(lower(usr), "guest", "set var pass netex") }
User>
```

EXT Function

The EXT function extracts and returns a bounded sequence of characters from a string.

Format:

```
ext(string, number1, number2)
```

Where:

- string** a string expression in the form of a variable, literal, or function.
- number1, number2** the lower and upper boundary limits for the characters to be extracted from 'string'. Parameter values less than or equal to zero are interpreted relative to the end of the string.

Example:

Display the sequence "CDE" from the string "ABCDEF":

```
User> text {ext("ABCDEF",3,5)}  
User: CDE
```

The same sequence ("CDE") may be displayed by using parameter values relative to the end of the string, as shown:

```
User> text {ext("ABCDEF",-3,-1)}  
User: CDE
```

Display only the Hours and Minutes of the system time:

```
User> text Time: {ext(time(),1,5)}  
User: Time: 14:27
```

Define the input prompt to display 'User' and the Version Number of USER-Access extracted from 'VERSION:LOCAL':

```
User> set input prompt {} User {ext(version:local,1,3)}>  
User 2.0>
```

INDEX Function

The INDEX function returns the position of the second string specified within the first string. The function returns zero if the second string is not found.

Format:

```
index(string1, string2)
```

Where:

string1, string2 string expressions in the form of a variable, literal, or function.

Examples:

Display the position of the sequence “CDE” within “ABCDEF”:

```
User> text {index("ABCDEF", "CDE")}
User: 3
```

Find the position of the month March in a string containing a list of the months:

```
User> text {Index("JanFebMarAprMayJunJlyAugSepOctNovDec", "Mar")}
User: 7
```

The following is an example of an index() search that failed to find the second string within the first:

```
User> text {index("abcdef", "cat")}
User: 0
```

LEN Function

The LEN function returns the count of characters that make up the specified string.

Format:

<code>len(string)</code>

Where:

string a string expression in the form of a variable, literal, or function.

Examples:

Display the number of characters in "ABCDE":

```
User> text {len("ABCDE")}
User: 5
```

Display the length of the results of the DATE() string function:

```
User> text Length = {len(date())}
User: Length = 16
```

Display only the year portion of the system date by subtracting three from the length of the date and then extracting the last four characters:

```
User> text Year: {ext(date(),sub(len(date()),3),len(date()))}
User: Year: 1989
```

A simpler form of the example above is:

```
User> text Year: {ext(date(),-3,0)}
User: Year: 1989
```

Logical Operations

The following is a list of the operators for numerical equivalence tests:

- EQ** tests if the first number is equal to the second number.
- NE** tests if the first number is not equal to the second number.
- LT** test if the first number specified is less than the second.
- GT** test if the first number specified is greater than the second.
- LE** test if the first number specified is less than or equal to the second.
- GE** test if the first number specified is greater than or equal to the second.

Format:

```
eq(number1, number2, if_true [, if_false ])  
ne(number1, number2, if_true [, if_false ])  
lt(number1, number2, if_true [, if_false ])  
gt(number1, number2, if_true [, if_false ])  
le(number1, number2, if_true [, if_false ])  
ge(number1, number2, if_true [, if_false ])
```

Where:

- number1, number2** numbers to be compared.
- if_true** a string expression whose value the function takes if the test is successful.
- if_false** an optional string expression whose value the function takes if the test fails. If this argument is omitted, the function takes on the value of a null string.

Examples:

Ask the user to enter a number between 1 and 10:

```
User> ask -prompt "Enter a # (1-10): " num1  
User: Enter a # (1-10): 10
```

Check if the number is less than or equal to 10:

```
User> text {le(num1,10, "Good", "Bad")}  
User: Good
```

Check the number for proper entry as defined in the first example, if it is between 1 and 10:

```
User> text {ge(num1,1,le(num1,10, "Good", "Bad"), "Bad")}  
User: Good
```

LOWER and UPPER Functions

The UPPER function returns the upper case equivalent of a string expression, all characters except lower case characters are left untouched.

The LOWER functions return the lower case equivalent of a string expression, all characters except upper case characters are left untouched.

Format:

<pre>upper(string) lower(string)</pre>
--

Where:

string a string expression in the form of a variable, literal, or function.

Examples:

Display the lower case equivalent of “ABCdef”:

```
User> text {lower("ABCdef")}
User: abcdef
```

Display the upper case equivalent of “ABCdef”:

```
User> text {upper("ABCdef")}
User: ABCDEF
```

Display the system date in lower case:

```
User> text Today is: {lower(date())}
User: Today is: thu mar 16, 1989
```

To display the Weekday in upper case:

```
User> text The Day is: {ext(upper(date()),1,3)}
User: The Day is: THU
```

To set a predefined USER-Access variable “password” to its lower case equivalent:

```
User> set var password {lower(password)}
User>
```

MSG Function

The MSG function returns the information of the last message. MSG is most often used to tailor the USER-Access output message format.

Format:

```
msg(component [, facility])
```

Where:

component the code for the type of message data to return. Valid message components are: Text, Facility, Code, Severity, Retry, or Purge.

Text requests the text from the message.

Facility requests the source of the message: NETEX, UA, UAxxx (where xxx represents the host product code), SI, SIxxx, MUX, MUXxxx, or the operating system mnemonic for the host generating the error.

Code requests the message number from the facility.

Severity requests the severity of the message: E, W, or I for Error, Warning, and Informational respectively.

Retry requests whether or not the network error can be retried (not fatal). Results are either Y, for can-be-retried, or N, for cannot-be-retried.

Purge will purge the message stack. One may find it useful to ensure that the next attempt to read a message resulted in a message from the last command, in this case the user will need to Purge the message stack.

facility to get the message from a certain facility: NETEX, UA, UAxxx (where xxx represents the host product code), SIxxx, MUXxxx, or the operating system mnemonic for the host generating the error. The default is UA.

Examples:

Display the facility of the last error message:

```
User> text The Last Error Message came from: {msg("f")}
User: The Last Error Message came from: UA
```

Display the text of the last error message from USER-Access:

```
User> text The Last Error Message was: {msg("t")}
User: The Last Error Message was: Invalid command 'oops'
```

Display the code of the last error message from USER-Access:

```
User> text The Last Error Code was: {msg("c")}
User: The Last Error Code was: 4708
```

Set the OUTPUT FORMAT qualifier to display only the message text when an error occurs:

```
User> set output format {} {msg("text")}
```

PARAMS Function

The PARAMS function substitutes the positional parameters specified. It is important to note that quoted parameters remain quoted and are considered one whole parameter - regardless of imbedded spaces.

Format:

```
params(number1, number2 [, char])
```

Where:

number1, number2 positional parameters number1 through number2 for substitution. Use number2 = "0" to mean "the rest".

char is the optional parameter separator to use. The default is a space.

Examples:

Define an alias "PRINT" to display the parameters passed to it:

```
User> set alias print {} text {params(2,3)}
```

Execute the alias with no parameters:

```
User> print  
User:
```

Now execute the alias with the parameters: this text string

```
User> print this text string  
User: text string
```

Execute the alias with the parameters: this "is a" test:

```
User> print this "is a" test  
User: "is a" test
```


SLEEP Function

The SLEEP function causes USER-Access to pause or “sleep” for a specified number of seconds. This process is not interruptible. This function results in a null string.

Format:

<code>sleep(number)</code>

Where:

number number of seconds to pause/sleep.

Examples:

To pause/sleep a USER-Access session for ten seconds:

```
User> {sleep(10)}  
User>
```

If an error occurs during a connect command, the alias “RECONnect” will go to sleep for thirty seconds and then attempt to connect again:

```
User> set alias RECONnect {} set input continue on !  
More>> start: !  
More>> con {params(1,0)} !  
More>> eqs(status(),"S","exit")} !  
More>> {sleep(30)} !  
More>> goto start
```

STATUS Function

The STATUS function returns the single status character of the previous command: S = Success, E = Error. Successful execution of the following commands leave the previous command status intact: CONTINUE, EXIT, GOTO, QUIT, and TEXT. This gives the user the ability to position within a script (for example, to an error processing section) without clearing the status from previous failures. If any of these commands fail, an error status is set. A status return specified by EXIT or QUIT (for example, EXIT ERROR) will override previous command status.

The status returned from a LOCAL or REMOTE command is the result of the USER-Access LOCAL or REMOTE command and not the completion code set by the resulting host command. To check the status of the host command, use the variables {status:local} or {status:remote}. If LOCAL or REMOTE command execution is not supported on the requested host, the status() function will be set to 'E'.

Format:

status()

Examples:

To display the output of the status function:

```
User> text {status()}
User: S
```

To display the status of the last command:

```
User> text Last Command {eqs(status(),"S","Succeeded","Failed")}
User: Last Command Succeeded
```

Define the input prompt to signal a message when an error has occurred:

```
User> set input prompt {} {nes(status(),"S","Error ")}User>
```

With the above definition, issue an invalid command and then a valid command to test the new prompt:

```
User> oops
User: Invalid command 'oops' (UA-4708).
Error User> text Correct the Prompt
User: Correct the Prompt
User>
```

TIME Function

The TIME function returns the system time of the local host.

Format:

```
time([num])
```

Where:

num this is a number that specifies the format of the time. For the TIME function:

0 = HH:MM:SS

1 = HHMMSS

Where H = Hours; M = Minutes; S = Seconds. Specifying a value other than 0 or 1 returns a null string. Default is 0.

Examples:

Output the time without the colon separator:

```
User> text The Time is: {time(1)}  
User: The Time is: 142448
```

Output the time:

```
User> text The Time is: {time()}  
User: The Time is: 14:24:48
```

Redefine the input prompt to prompt with the system time:

```
User> set input prompt {} {time()}>  
14:25:04>
```

The empty braces ({}) in the above example are needed to disable string substitution until each prompt is displayed. The new system time is then evaluated each time the prompt is displayed.

Disabling String Substitution

When USER-Access sees {string} on a command line, it immediately tries to perform string substitution on the string. To disable string substitution, place an empty {} on the command line prior to the string substitution syntax. The typical place to do this is during an alias definition. For example:

```
User> set alias put send {} {sourcefile} {destinationfile}
```

Upon seeing the empty {} before the string substitution syntax {sourcefile} and {destinationfile}, USER-Access knows to not substitute the values of 'sourcefile' and 'destinationfile' at this time. The resulting definition for alias PUT is:

```
User> show alias put
User: PUT ..... send {sourcefile} {destinationfile}
```

If the empty {} had not been included during the definition above, USER-Access would have replaced {sourcefile} and {destinationfile} by their current values at the time the alias was defined. If they were undefined, they would have been replaced by the null string. The goal of an alias is usually to replace the value of the variable at the time the alias is run, not when it is defined.

The empty {} is actually used as a toggle to turn string substitution on and off. In the following example, the first occurrence of {} turns off string substitution, which results in {sourcefile} not being replaced by its value. The second occurrence of {} turns string substitution back on. This results in {destinationfile} being replaced by its current value (assume 'dest.new'):

```
User> set alias put send {} {sourcefile} {} {destinationfile}
```

The resulting alias definition looks like:

```
User> show alias put
User: PUT ..... send {sourcefile} dest.new
```

In this case, the alias PUT becomes a send command where the destination file is always 'dest.new'. Since {} is used as a toggle, it should only appear once within an alias definition (including multicommand aliases) when string substitution is to be ignored for all variables declared.

Nested String Substitution

The string substitution syntax also allows for nested substitution. Nested substitution provides for embedding string substitution syntax within string substitution. Figure 5 is a representation of nested substitution.

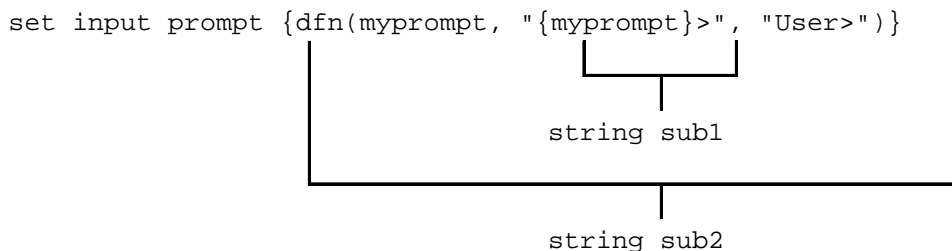


Figure 5. Nested String Substitution

The example above sets the USER-Access input prompt to either the value of variable *myprompt* (if it is defined), or else to the string User>. Since there exists nested string substitution, USER-Access first processes the innermost one (labeled string sub1 above) to evaluate the variable *myprompt*. The double quotes outside of {myprompt} turns the resulting value into a string literal that is then used as the second argument to the

dfn() function. Once this is done, USER-Access processes the outermost string substitution syntax (labeled string sub2 above).

All string substitution processing is performed by USER-Access from the inside out. This is important to keep in mind when creating such things as custom prompts or scripts. Since the inside string substitution syntax is processed first, it is treated as a separate entity in itself. This is significant because it affects the use of double quotes for string literals. Normally double quotes must be escaped when they are used within another set of double quotes in order for USER-Access to take them literally. However, if the outer set of double quotes is not part of the immediate string substitution syntax containing the inner set of double quotes, then the inner set of double quotes should not be escaped. The following example illustrates this point.

```
User> ask -prompt "{upper("enter your name")}" name
```

Since the outer set of double quotes (“{...}”) is outside of the string substitution syntax, the inner set should not be escaped. This is because the string substitution syntax is processed first, resulting in the nonquoted string ENTER YOUR NAME. The outer set of quotes then is applied to that string and the remainder of the command is processed.

Developing USER-Access Scripts Using Input Files and Aliases

USER-Access was designed to be very easy to use for all types of users. The commands are simple and the syntax is straightforward. However, it is often the case that a site wants to customize the USER-Access interface to be even more simple or familiar for its users. This customizing is generally done by more sophisticated USER-Access users then handed back to the general user base. This section addresses the areas important to developing USER-Access scripts.

A script can be in the form of an input file or an alias; USER-Access treats them the same internally. The difference is in the way they are defined. Input files are created using a standard text editor. Aliases are created using the SET ALIAS command and require special command line syntax. The examples that are given in this section, although they apply to all types of scripts, are generalized to emphasize the topic of discussion and do not include the special syntax required for creating aliases. The reader should be familiar with USER-Access command line processing, most notably, the sections on “Special Characters” on page 47 and “USER-Access String Substitution” on page 48.

USER-Access Input Files

USER-Access input files or input scripts give users the ability to write powerful program-like procedures that can be run on several different host types, without regard to host-specific command language differences. A USER-Access input file or input script is a file that contains a list of USER-Access commands. Input scripts are created using any standard text editor.

Assume for the following examples, that there exists an input script called HOSTDIR, shown below, which connects to a predefined host named BETA, issues a REMOTE DIRECTORY command, and then disconnects.

```
* Input script HOSTDIR - Give directory listing of
*                       remote host named BETA.
*
connect beta default test
remote dir
disconnect
```

There are three ways to make use of an input script:

1. With the INPUT command. The INPUT command tells USER-Access to read and execute the USER-Access commands in the input file given. For example:

```
User> input hostdir
```

This command tells USER-Access to look for an input script with the file specification HOSTDIR and then read it line by line, executing commands along the way.

2. Using the INPUT qualifier SEARCH - When USER-Access reads a command from the command line, it first looks for an alias by that name and translates it if found. If the command is not an alias, USER-Access determines if it is a USER-Access command. If not, it looks at the INPUT qualifier SEARCH. If that qualifier is defined, USER-Access uses the SEARCH path to find an input file by the name of the command it read. If an input file is found, USER-Access reads and executes it. If no input file is found, USER-Access issues an “Invalid command” error. Therefore, the second way to use the HOSTDIR input script is to first define the INPUT SEARCH qualifier (see the INPUT Command in “Command Descriptions” on page 115), and then type HOSTDIR on the command line:

```
User> hostdir
```

Typing HOSTDIR here gives the appearance that HOSTDIR is actually a USER-Access command. The INPUT SEARCH qualifier can also contain SEARCH keywords (SITE) and (USER). Refer to “Stratus USER-Access SEARCH Keywords (SITE), (USER), and (NONE)” on page 92 for more information

3. On the USER-Access command line - The third option is to specify the input file on the USER-Access command line when USER-Access is invoked. This option is most often used when running USER-Access within a batch job. It simply tells USER-Access to read and execute all of the commands within the input script and then exit USER-Access. For more information, see “Invoking USER-Access on Stratus” on page 11.

Echoing Input Scripts to the Terminal

In order to have USER-Access display the INPUT commands as they are executed, the user must turn on input echo with the SET INPUT command:

```
* Echo all commands as they execute
*
set input echo on
```

This command can be issued interactively, prior to the INPUT command, or given as the first command in the input file. INPUT ECHO will echo each command as it appears prior to string substitution. To display each input command after string substitution, turn on the INPUT VERIFY qualifier as:

```
* Echo all commands after string substitution
*
set input verify on
```

Refer to the INPUT command in “Command Descriptions” for more details on these qualifiers. The section “Debugging a USER-Access Alias or Input Script” on page 87 also addresses the VERIFY qualifier.

Displaying Output and Accepting Input within a Script

The TEXT command is used to display output within a USER-Access session. For example, the commands in a script to display a welcome message to a user could be:

```
text *****
text      Welcome to EFT!
text *****
```

A TEXT string can also include USER-Access string substitution syntax {...}. Therefore, string literals, string variables, or string functions can be substituted within a TEXT command line to provide additional information. For example, to enhance the example above, the string function DATE() could be used, along with qualifier values VERSION and DIRECTORY from the LOCAL environment:

```

text *****
text      Welcome to EFT Version {version:local}.
text
text                {date()}.
text
text      Your current directory is: {directory:local}.
text *****

```

The ASK command can be used to make scripts more friendly for novice users. For example, a TRANSFER script could be defined that would prompt a user for worthwhile information then execute the commands on the user's behalf:

```

* This is a sample TRANSFER script
*
ask -prompt "Transfer being made to what host? " host
ask -prompt "User ID on host {host}? " uid
ask -secure -prompt "Password for user {uid}? " pw
ask -prompt "File to be transferred? " file
connect -quiet {host} {uid} {pw}
send -quiet {file}
text *****
text File '{file}' has been transferred to {host}.
text *****
disconnect -quiet

```

To execute the script (if appropriate setup was done by the site administrator), the user would type only TRANSFER:

```

User> transfer
Transfer being made to what host? BLUESKY
User ID on host BLUESKY? guest
Password for user guest?         
File to be transferred? tmpfile
*****
File 'tmpfile' has been transferred to BLUESKY.
*****
User>

```

Passing Parameters to a Script

Parameters may be passed to a USER-Access script. These parameters are referred to as positional parameters because they are identified by their position on the command line. For example, the following INPUT command passes the input file SETUP three positional parameters, the first positional parameter is HOSTA, the second is SMITH, and the third is JOHN:

```
User> input setup hosta smith john
```

An input file can be passed several positional parameters. Each parameter is identified in the input file by its position number in braces ({1}, {2}, {3}, etc.). In the example above, HOSTA is represented by {1} in the input script SETUP, SMITH is {2}, and JOHN is {3}. In general, the mapping is :

```
{0} ... positional parameter zero - the entire parameter string
{1} ... positional parameter one
{2} ... positional parameter two
.
.
.
{n} ... positional parameter n
```

Positional parameters are used with an input script using the syntax described above. For example, the file SETUP may contain the three lines:

```
text {3} {2} is attempting to connect to host {1}
connect {1} {2} {3}
text hello {3} {2}. How are you?
```

When the INPUT command is issued, USER-Access performs string substitution. After string substitution the command lines appear as:

```
User> input setup hosta smith john
text john smith is attempting to connect to host hosta
connect hosta smith john
text hello john smith. How are you?
```

To pass multiple words or strings with embedded blanks as a single parameter, enclose them in double quotes. Refer again to the example above. If the third parameter was JOHN HENRY instead of just JOHN, the string JOHN HENRY would have to be enclosed in double quotes:

```
User> input setup hosta smith "john henry"
```

Now, parameter 1 is HOSTA, parameter 2 is SMITH, and parameter 3 is JOHN HENRY.

Up to this point positional parameter passing and input prompting has been discussed. The next section combines these features by making use of string functions.

Using String Functions within a USER-Access Script

In order to gain even more flexibility, a script can be designed to use positional parameters if they are passed, and to prompt for input in the absence of positional parameters. This added feature of scripting makes use of USER-Access string functions. (See the section entitled “String Functions” on page 52 for a more detailed discussion.)

In the sample below, the string function `ndf()`, (if Not DeFined), is used. `ndf()` tests its first argument (variable 1, 2, 3, or 4) to see if it's defined. If it's not, `ndf()` executes its second argument (ASK -prompt ...). If argument 1 is defined, `ndf()` executes argument three (optionally left out). The new definition of the TRANSFER script is:

```
* This is a sample TRANSFER script that uses String Functions
*
{ndf(1, "ask -prompt ""Transfer being made to what host? "" 1")}}
{ndf(2, "ask -prompt ""User ID on host {1}? "" 2")}}
{ndf(3, "ask -secure -prompt ""Password for user {2}? "" 3")}}
{ndf(4, "ask -prompt ""File to be transferred? "" 4")}}
connect -quiet {1} {2} {3}
send -quiet {4}
text *****
text File '{4}' has been transferred to {1}.
text *****
disconnect -quiet
```

The logic of the script when invoked is:

- If no parameters are on the TRANSFER command line, prompt the user for all four pieces of information and read them into variables 1, 2, 3, and 4.
- If one parameter is passed, use it as variable 1 (Host), and prompt for the other three.
- If a second parameter is passed on the TRANSFER command line, use it as variable 2 (User id), and prompt for the third and fourth variables.
- If three parameters are passed on the command line, read them into variables 1, 2, and 3 (Password) respectively and prompt only for the fourth variable.
- If four parameters are passed on the command line, read them into variables 1, 2, 3 and 4 (file) respectively, and do not prompt at all.
- Finally, execute the remaining commands, substituting the variables or parameters as needed.

The following is an example execution of the TRANSFER script with two parameters passed to it:

```
User> transfer BLUESKY guest
Password for user guest? _____
File to be transferred? tmpfile
*****
File 'tmpfile' has been transferred to BLUESKY.
*****
User>
```

Since two parameters were passed, the script prompted only for the last two. Scripts like this one are especially useful when there is a need to run both interactively and in batch. Batch jobs require all the parameters to be defined since they cannot prompt for user input.

Using USER-Access Labels and GOTOs

To make USER-Access scripts even more powerful, users can merge string functions with GOTO processing. The GOTO command instructs script processing to continue at the specified label, either backwards or forwards. The format is:

Command	Parameters
GOTO	label

Where:

label is an alphanumeric string from one to twenty characters (including underscores) in length.

All labels are case sensitive and must appear somewhere within the current input level. That is, if the GOTO appears within an input script, the matching label must also be in that input script. If the GOTO appears in an interactive input level, the matching label must also be found within that interactive level. The format of a command line that contains a label is:

```
label: [command]
```

The colon immediately following the label is required. The label can appear on a line by itself, or it may be followed by a valid USER-Access command or alias. The following is an example of a simple loop alias:

```
* Sample GOTO/Label script. Send 5 files
* having the names FILE1 thru FILE5.
*
set variable count 1
LOOP:
send FILE{count}
set variable count {inc(count)}
{le(count, 5, "goto LOOP")}
*
text All files sent.
```

The variable count is first initialized to 1. The next line simply declares a label called LOOP. The SEND command is then issued for a file named FILE_x where x is the current value of count. Following that, the variable count is reset to its value plus one (incremented by one). Finally, a check is made on the value of count. If it is less than or equal to 5, the GOTO LOOP command is substituted as the next USER-Access command and processing branches up to label LOOP. If count exceeds 5, processing falls through to the next command outside of the loop.

It is important to remember that USER-Access treats labels as case sensitive. Therefore, one must make sure that a label specified on a GOTO command matches the actual label's case exactly. Duplicate labels (labels that have the same name with identical case) are considered an error.

Whenever a GOTO or a label is encountered during command line processing, all future commands get stored internally within USER-Access. The number of commands that can be stored is limited by the amount of available memory allocated for the process which varies from machine to machine. The best practice is to avoid letting scripts that contain GOTOs become too large.

Since scripting is really an interpretive command language, USER-Access must parse each command as it executes it. Therefore, an error within a script will not be caught until the script is run and the erroneous condition is encountered. A missing label, for example, will result in the entire script being read before an error message is given.

GOTO and labels may also appear at the interactive session level. Refer to the GOTO command in “Command Descriptions” on page 115 of this manual for further information.

Using the ON (ERROR/INTERRUPT) Command

A useful command for building more robust scripts is the ON command. ON has the format:

Command	Parameter
ON	exception [action]

Where:

exception is any one of the following:

ERRor	on USER-Access error perform action
INTerrupt	on keyboard interrupt perform action
LOCAl_error	on LOCAL command error perform action
REMote_error	on REMOTE command error perform action

action is any valid USER-Access command or alias, the most likely of which being:

CONTinue	ignore the exception
Exit	exit the current USER-Access session
GOTO	GOTO a specified label
Input	input a specified USER-Access script
TEXT	display a message
<none>	turn off the specified exception

If *action* is more than one command, the results are unpredictable.

The ON command allows a user to catch any one of the exceptions listed above and perform a predefined action. It is most useful within USER-Access scripts for tailoring exception handling. ON commands generally appear at the beginning of a USER-Access script since they set up actions to be taken on future processing within that script. For example, the following script catches any keyboard interrupt (initiated by the user), and automatically causes the session to terminate:

```
* Cause keyboard interrupt to exit session
*
on interrupt exit
connect hosta guest netex
send src_file
disconnect
```

The ON INTERRUPT exception as shown above, establishes an alternative action to be taken in the case of a user generated keyboard interrupt. By default, without an ON INTERRUPT specified, USER-Access terminates all input levels (if within nested input scripts), and returns to the interactive level. If the INPUT CONTINUE qualifier was set, USER-Access terminates only the current input level and continues processing in the next level up.

The ON ERROR exception establishes an alternative action to be taken when a USER-Access error occurs. Without an ON ERROR specified, USER-Access terminates all input levels (for nested input scripts), and begins processing at the interactive level. If the INPUT CONTINUE qualifier was set, USER-Access displays the error but continues processing the next command. The ON ERROR command allows for more flexibility on an error condition.

The ON LOCAL_ERROR and ON REMOTE_ERROR exceptions give the user the ability to take special action when a LOCAL or REMOTE host command fails. On hosts that support it, a special LOCAL or REMOTE STATUS qualifier will be set reflecting that host's error code for the particular error condition. Normally, without ON LOCAL_ERROR or ON REMOTE_ERROR specified, USER-Access just ignores host errors and continues processing the next command.

For further details about the ON command, see "Command Descriptions" on page 115.

Checking Command Status

The *status()* function allows the user to write scripts that check the status of the previous command and take action accordingly. A simple example appears below:

```
*
*   Print status of a single file transfer.
*
set input continue on
send -quiet src_file
text Transfer {eqs(status(), "S", "Successful.", "Failed.")}
```

The example above demonstrates how *status()* can be used to print the results of the previous command (in this case SEND). The *status()* function returns either an "S" for Success, or an "E" for Error. Since it is a string function, the result is a string. The *status()* function can be combined with any other string function or USER-Access command to enhance script processing. Above, it is used with the *eqs()* function to check the results which are then printed by the TEXT command.

Refer to the section on string functions for more details and examples of the *status()* function.

Creating USER-Access Aliases

Aliasing is simply another form of USER-Access scripting. Although the previous sections discussed scripting in the context of INPUT files, everything described applies to USER-Access aliases. Aliases can be thought of as USER-Access input files. In fact, a multicommand alias (discussed later) is treated in exactly the same way. The difference between a multicommand alias and an input script is the way in which they are defined. As mentioned earlier, input scripts are defined using a standard text editor. Aliases are defined using the SET ALIAS command.

The alias capabilities of USER-Access provide a means of creating a custom command set that can be used by all users or a group of users. An alias is simply a new name for a USER-Access command or set of commands. The following example shows how to create a simple alias (or new command) called FETCH that is equivalent to the RECEIVE command:

```
User> set alias fetch receive
```

To display the definition of the new alias, use the SHOW ALIAS command:

```
User> show alias fetch
User: FETCH ..... receive
```

The alias name FETCH appears on the left and its translation or definition appears on the right. Now to transfer a file from the remote host to the local host, either the RECEIVE command or the FETCH alias can be used. All qualifiers and parameters for the RECEIVE command are also valid for FETCH since USER-Access just maps FETCH to RECEIVE. The RECEIVE command is executed, but as far as the user is concerned, the command executing is FETCH. Therefore, the following are equivalent:

```
User> fetch -quiet -mode character sourcefile
```

and,

```
User> receive -quiet -mode character sourcefile
```

Aliases can be more complex than a single mapping of FETCH to RECEIVE. For example, the FETCH alias can be defined to include parameters or qualifiers as part of its definition. Below is an example of the FETCH alias that includes some RECEIVE qualifiers in its definition:

```
User> set alias fetch receive -quiet -mode character
```

To display the new definition alias FETCH, the SHOW ALIAS command is again used:

```
User> show alias fetch
User: FETCH ..... receive -quiet -mode character
```

Now every time FETCH is invoked, the QUIET qualifier is turned on and the MODE is set to CHARACTER. Therefore, the following are equivalent:

```
User> fetch sourcefile
```

and,

```
User> receive -quiet -mode character sourcefile
```

As was true earlier, additional qualifiers and parameters may be passed to FETCH (and thus passed through to RECEIVE) simply by adding them to the FETCH command line when it is invoked.

To remove a previously defined alias, simply define the alias again without a definition:

```
User> set alias fetch
```

This will in effect “undefine” the specified alias. Following the command issued above, FETCH will no longer be a valid alias.

USER-Access Aliases vs. Host Aliases

There are two types of aliases, Host Aliases and USER-Access aliases. Host aliases are either Local aliases or Remote aliases. That is, their definitions translate to either Local host commands or Remote host commands. USER-Access aliases translate to USER-Access commands (SEND, RECEIVE, CONNECT, ASK, etc.).

In addition to the obvious differences between the two alias types, there are a couple of other distinctions that must be mentioned. First, USER-Access aliases are defined using SET ALIAS. The format is:

Command	Parameter
SET ALIAS	alias_name ua_command

where ua_command must translate into a USER-Access command. Host aliases on the other hand are defined using SET LOCAL ALIAS or SET REMOTE ALIAS. The format is:

Command	Parameter
SET LOCAL ALIAS	loc_alias_name host_command

and,

Command	Parameter
SET REMOTE ALIAS	rem_alias_name host_command

where host_command must be a valid command on the local or remote host respectively.

The other big distinction between USER-Access aliases and Host aliases is that USER-Access aliases may have multicommand definitions (see the section entitled “Creating Multicommand USER-Access Aliases”), whereas Host aliases can translate only into a single host command. This is not typically a problem since most hosts support command procedures or script files which can then be accessed by a Host alias.

The remainder of this section is devoted to issues relating to USER-Access aliases. The topics apply to host aliases as well unless specifically designated as “USER-Access Aliases”.

Creating Multicommand USER-Access Aliases

An alias can translate to a single USER-Access command as shown above where FETCH was mapped to RECEIVE. USER-Access aliases can also be defined to be a sequence of several commands. The method for defining a multicommand alias is to put each command of the definition on a separate line, where each line, except for the last one, is terminated by the escape character ‘!’. The escape character tells USER-Access to continue the alias definition on the next line. The following is an example of a three line FETCH alias that connects to HOSTA, receives a file called ‘sourcefile’, then disconnects:

```
User> set alias fetch connect hosta guest netex !
More>>                receive sourcefile !
More>>                disconnect
```

The SHOW ALIAS command can be used to display the new FETCH alias:

```
User> show alias fetch
User: FETCH ..... connect hosta guest netex
User:                receive sourcefile
User:                disconnect
```

Notice that USER-Access strips the ‘!’ from the alias definition. It is only needed for the initial definition of the alias.

Although they are very powerful, multicommand aliases are limited to roughly 500 characters in length. It is suggested that if an alias approaches this limit, it should be made into an input script and stored in a file.

Unlike single command aliases (such as the simplest FETCH alias defined earlier), multicommand aliases do not implicitly pass command parameters or qualifiers through to the actual USER-Access commands. USER-Access has no way of determining which command a particular parameter is destined for unless the alias is set up to pass its own parameters, as explained in the next section.

Passing Parameters to an Alias

When creating multicommand aliases, it is often desirable to allow parameters to be passed on the alias command line. These parameters can then be substituted into various places within the command sequence in the same way as was done for input scripts.

Refer back to the FETCH alias defined in the previous section. To make that alias more flexible, one can define it to take as parameters such things as userid and password on the CONNECT line, and file name on the RECEIVE line. The definition then is:

```
User> set alias fetch {} connect hosta {1} {2} !
More>>                receive {3} !
More>>                disconnect
```

Note the empty ‘{}’ above. These are important when defining alias parameters and will be discussed shortly. The resulting alias becomes:

```
User> show alias fetch
User: FETCH ..... connect hosta {1} {2}
```

```
User:          receive {3}
User:          disconnect
```

To invoke the alias with the same parameters that were embedded in the alias earlier, a user would type:

```
User> fetch guest netex sourcefile
```

Parameter 1, represented as {1}, gets replaced by 'guest', parameter 2, represented as {2}, gets replaced by 'netex', and parameter 3, represented as {3}, gets replaced by 'sourcefile'. The actual commands that get executed would be equivalent to typing the following:

```
User> connect hosta guest netex
User> receive sourcefile
User> disconnect
```

Now refer back to the empty '{}' used in the definition of FETCH earlier. This special notation is used to tell USER-Access to "turn off" parameter substitution while the alias is being defined. The empty '{}' is crucial when defining aliases that use positional parameters or any string substitution (refer to "USER-Access String Substitution" on page 48). If parameter substitution is not turned off USER-Access will attempt to replace the positional parameters with their values at the time the alias is defined (which generally means they get replaced with a null string). The empty '{}' may be placed anywhere within the alias definition and turns off string substitution until the alias definition ends or another '{}' is encountered.

Notice that the empty '{}' appears only on the first line even though the second line is referencing positional parameter 3. The empty '{}' is really a toggle that turns string substitution from off to on (or from on to off) each time it is encountered within a definition. See the section entitled "Disabling String Substitution" on page 74 for further details.

Accepting Input within a USER-Access Alias

Although aliases that accept parameters are more flexible than those that do not, it may become confusing to a new user which parameters to pass and in what order. Therefore, a more desirable solution is to create aliases that prompt for input. Refer once again to the FETCH alias defined in the previous section. It was set up to take three parameters: userid, password, and file name. FETCH can be made much more usable by having it prompt for the required parameters, as follows:

```
User> set alias fetch {} -
More>>      ask -prompt "User Id?   " uid
More>>      ask -prompt "Password?  " -secure pw !
More>>      ask -prompt "File Name? " fname !
More>>      connect hosta {uid} {pw} !
More>>      receive {fname} !
More>>      disconnect
```

Note the empty '{}' again. These tell USER-Access not to perform string substitution (in this case on variables uid, pw, and fname), until the alias is actually executed. The dash (-) on the first line is used to tell USER-Access to continue the definition on the next line.

Quickly breaking this alias apart, the first line simply prompts the user for a userid and reads the response into variable uid. The next line prompts for a password and reads the result into variable pw. (The SECURE qualifier is used to tell USER-Access not to echo the response back to the terminal. It should be noted that some versions of USER-Access cannot support this feature due to operating system limitations. For more information, refer to the ASK command in the "Command Descriptions" on page 115.) Next, the file name is requested and read into variable fname. The CONNECT and RECEIVE commands are then issued with the appropriate variables to be substituted. Finally the DISCONNECT is performed.

To run the new FETCH alias, a user need not know anything about the parameters; the alias will take care of that by prompting for them. Below is an example execution of FETCH, including user responses:

```
User> fetch
User Id?  guest
Password? _____
File Name? sourcefile
User:    < connect results >
User:    < receive results >
User:    < disconnect results >
```

The output here has been removed to emphasize how prompts can now be used to interact with a user in order to make a more friendly interface. It should be noted here that string functions can also be used within an alias to provide the option of parameter passing or input prompting in exactly the same way as explained in “USER-Access Input Files” on page 75.

Abbreviating Alias Names

To make life simpler for the user, alias names can be defined to allow abbreviations when they are invoked. For instance, to allow the simplified FETCH alias to be invoked by typing only FET, define it as:

```
User> set alias FETch receive
```

Now the definition can be displayed:

```
User> show alias fetch
User: FETch ..... receive
```

By capitalizing only a portion of the alias name (leading consecutive uppercase characters only), a user can define aliases that can be abbreviated or spelled out. In the example above, the FETCH alias can now be invoked as “fetch”, “fetc”, or “fet”. By default, aliases are created with all capital letters unless a combination of upper and lower case characters are given in its definition, the first character of which must be upper case. The minimum spelling of an alias name includes all letters up to the first lower case letter.

If alias names are defined which cause duplicate abbreviations, (for example, ABc and ABdef), the first alias alphabetically is processed. In the example, AB would cause ABc to execute.

Defining Multiword Alias Names

For those interested in being even more creative, an alias can be defined with a multiword name. That is, by putting double quotations around the alias name on its definition, the name can contain embedded blanks. For example, to create an alias called “FETCH A FILE”, define it as:

```
User> set alias "FETch A File" receive
```

Its definition can then be displayed as:

```
User> show alias "fetch a file"
User: FETch A File ..... receive
```

And it can be invoked as:

```
User> fet a file sourcefile
```

Where “sourcefile” is the file name to receive.

Multiword alias names are particularly useful for users that prefer a more English-like command set. They can also be used to redefine multiword USER-Access commands such as SHOW HOSTS, SET ALIAS, etc.

Debugging a USER-Access Alias or Input Script

It may be necessary from time to time to step through an alias or input script as it is executing, to see exactly what parameters it is using once string substitution has been performed. In the normal case, each command of an alias or input script is silently issued by USER-Access when the alias is invoked. To tell USER-Access to display each command in its “string substituted” form, set the VERIFY qualifier of the INPUT command:

```
User> set input verify on
```

Now every command issued from the command line gets redisplayed before it is executed, with all positional parameters and string variables replaced by their actual values. This enables a user to debug an alias or input script, making sure what was expected to be substituted actually gets substituted. Assume the following FETCH alias is defined:

```
User: FETCH ..... connect hosta {1} {2}
User:                receive {3}
User:                disconnect
```

To debug or verify each command as it gets executed, the INPUT VERIFY qualifier is turned on and FETCH is invoked with parameters:

```
User> set input verify on
User> fetch guest "fast netex" sourcefile
User: connect hosta guest fast netex
User: ----- connect results -----
User: receive sourcefile
User: ----- receive results -----
User: disconnect
User: ----- disconnect results -----
User>
```

Another debugging tool is the CONTINUE qualifier of the INPUT command. By default, USER-Access stops processing an alias or input script as soon as one of its commands fails. For instance, in the example above, if the CONNECT failed, the RECEIVE and DISCONNECT commands would not be processed in the normal case. To tell USER-Access to keep processing an alias or input script even if an error condition occurs, set the CONTINUE qualifier of the INPUT command:

```
User> set input continue on
```

Now when the FETCH alias (or any alias) is invoked USER-Access will continue to process the remaining commands even if an error is encountered. The CONTINUE qualifier may even be set within an alias definition if the need arises.

The INPUT qualifiers ECHO, VERIFY, and CONTINUE are initialized to off by USER-Access. Once turned on, they remain on until the user turns them off by typing SET INPUT VERIFY OFF or SET INPUT CONTINUE OFF respectively. The values of these qualifiers also is determined by the INPUT level at any given time. Changing the setting of ECHO, VERIFY, or CONTINUE in an alias or input file changes the setting for the current level and lower levels. Settings for higher input levels are restored when the current input level terminates.

Error Message Formatting

USER-Access messages consist of the following components:

SEVERITY A single character severity level indicator. Possible values are:

I	Information
W	Warning
E	Error
F	Fatal

FACILITY The facility or subsystem name generating the message. This will generally be some version of the following:

UA	USER-Access host independent message.
UAxxx	USER-Access host dependent message where xxx represents the product number of the host generating the error (e.g. EFT263, H893V, etc.)
SI	USER-Access Service Initiator (SI) host-independent message.
SIxxx	USER-Access Service Initiator (SI) host-dependent message.
MUX	USER-Access Multiplex Server (MUX) host-independent message.
MUXxxx	USER-Access Multiplex Server (MUX) host-dependent message.
NETEX	A NETEX generated error.
opsys	An operating system specific error where OpSys is replaced by the operating system name generating the error.

CODE The unique error or message code.

TEXT The single line message text describing the error code.

The format of a message display is controlled by the OUTPUT command qualifier FORMAT. Specific components of a USER-Access message are extracted using the string function msg() (described in the section on “String Functions”). The default message format can be displayed as:

```
User> show output format
User: FOrmat ..... {msg("text")} ({msg("facility")}-{msg("code")}).
```

With this format defined, a simple “Invalid command” error would generate the following:

```
User> badcommand
User: Invalid command 'badcommand' (UA-4708).
```

The user can modify the message format simply by changing the value of the FORMAT qualifier of the OUTPUT command. The value can be any string so long as it includes some reference to string substitution when it gets interpreted. That is when OUTPUT FORMAT is defined, it must disable string substitution using the {} syntax. An invalid FORMAT specification will result in USER-Access returning the value to its original, default value. This is done to make sure error messages are properly displayed in the event that they were inadvertently shut off.

The following example modifies the error message format to print only the error severity, facility, and code. Note the use of {}

```
User> set output format {} {msg("severity")}:{msg("facility")}-{msg("code")}
```

With this message format, the same “Invalid command” error would generate the following display:

```
User> badcommand
User: W:UA-4708
```

It is advised that error message format tailoring should be left up to the site administrator. Most users will never need to modify the default format.

USER-Access Code Conversion

NETEX is ordinarily responsible for performing code conversion between computer systems. USER-Access provides an alternative code conversion facility intended for environments requiring more flexibility than that offered within NETEX. USER-Access code conversion adds the following capabilities:

- It supports ASCII to ASCII and EBCDIC to EBCDIC code conversion, allowing a site to handle differences among “like” conversion tables (Unisys A versus IBM EBCDIC, for example).
- It supports full (256-character) ASCII as well as the NETEX 128-character ASCII tables. This is particularly useful for handling the variety of country codes that appear in the last half of the ASCII tables.
- It allows a site to specify incoming code conversion and outgoing con-versions separately.
- It allows USER-Access to offer optional data verification facilities (CRC) for character file transfer as well as bit-stream transfer. Refer to “USER-Access Data Verification” below for more information.

All USER-Access code conversion is controlled by means of the TRANSLATE command. Refer to the TRANSLATE command as described in “Command Descriptions” on page 115. Using TRANSLATE, a site administrator can define code conversion tables, review the current tables, and enable or disable the USER-Access code conversion facility.

Although a user may use the TRANSLATE command to specify changes to the conversion tables “on the fly,” it is strongly suggested that code conversion be treated as a site operations issue and that any code conversion table changes be established at a site level by means of the TRANSLATE SEARCH qualifier. By using the TRANSLATE SEARCH Path mechanism, a site administrator can define the tables and enable USER-Access code conversion as part of the CONNECT/LOGIN process to particular systems.

When USER-Access code conversion is enabled, it replaces NetEx code conversion in all communications between systems. To protect the USER-Access protocol from code conversion changes the following characters may not be modified:

- Uppercase alphabetic characters (A-Z)
- Digits (0-9)
- Space, equal sign (=), and null

If USER-Access code conversion is to be used for only certain file transfers, it is recommended that aliases be set up to control enabling and disabling of the facility.

USER-Access Data Verification

USER-Access offers an optional data verification facility or Cyclic Redundancy Check (CRC) that can be involved on file transfers. When CRC is turned on, as a SEND or RECEIVE qualifier, USER-Access appends a block number, and the result of a CRC calculation to each block of the file transferred. If a block is lost or if the source CRC calculation does not match the destination calculation, the file transfer is aborted.

A sample alias, SENDCRC, shown below, will attempt to resend an aborted file a specified number of times. To make this alias easier to read, it uses the ASSIGN and TEST aliases that are defined in USER-Access.

```

* SENDCRC - send a file with CRC enabled - if the transfer
*           fails with a retryable error (this includes a
*           CRC failure), sleep for a while and try again.
*           A retry counter limits the number of attempts.
*
set alias sendcrc {} on error goto check!
send: send -crc {0}!
exit success!
check: test msg("retry") eqs "N" exit error!
      {sleep(10)}!
assign count inc count!
test count le 100 goto send!
text Maximum retries exceeded - SENDCRC cancelled.!
exit error

```

Since during normal operation a CRC error should be extremely rare, the SENDCRC alias (or similar logic incorporated in user-defined scripts) is an effective way to guarantee the delivery of data.

If CRC is performed on character files, USER-Access code conversion is used (refer to “USER-Access Code Conversion” on page 89) automatically. Although the CRC facility invokes the code conversion facility without any user action required, it must be noted that the code conversion facility will process the data based on the current user conversion tables (including SEARCH paths) only if translation has been enabled (i.e., TRANSLATE ON). If translation has not been enabled, USER-Access uses its own default tables and does not use any user specified translation tables or search paths.

The CRC algorithm is performed using 32-bit values. The integrity of a data stream is checked by comparing its state at the sending host and the receiving host. Each character in the data stream is used to generate a value based on a polynomial. The values for each character are then added together. This operation is performed at both ends of the data transmission, and the two results compared. If the results are different, an error has occurred during transmission.

USER-Access Data Compression

Support has been added to USER-Access for data compression and expansion during file transfer. The new SEND/RECEIVE qualifiers are:

```

COMPRESS - compress the source data stream (on/off)
EXPAND   - expand the destination data stream (on/off)
METHOD   - the method of compression (LZW, RLE)

```

The two compression methods currently supported are LZW and RLE. The LZW method uses the Lempel-Ziv-Welch algorithm for finding common substrings. This method is deterministic and can be performed on the fly. Block compression is performed with an adaptive reset whereby the code table is cleared when the compression ratio decreases. This method generally provides the best overall compression ratio, but requires significantly more CPU resource than the RLE method. LZW compression ratios for character data are typically 40% to 60% of the original data size. LZW compression ratios for binary data are difficult to predict. Applying LZW compression to already compressed data could actually increase the data size up to 130% of the original size.

The RLE method uses a simple Run Length Encoding algorithm that counts strings of repeated characters (usually spaces or nulls). This method is much faster than LZW compression but provides much poorer compression ratios (typically 80% to 95%). Unlike LZW, the RLE method will never grow data that is already compressed (except for the addition of the compression header).

The following examples demonstrate file transfers using the `-COMPRESS` and `-EXPAND` qualifiers.

Send a binary source file 'data' with data compression enabled. The destination file 'data.cmp' contains the compressed data:

```
User> send -mode stream data data.cmp -compress
```

Receive the same compressed file expanding the data stream back to the original binary file:

```
User> receive -mode stream data.cmp data -expand
```

The same binary data file can be compressed, sent across the network and expanded into the destination file:

```
User> send -mode stream data -compress data -expand
```

One sided compress/expand (the first two examples) is possible when connected to earlier releases (pre-R10) of USER-Access for all supported modes except CHARACTER. Two-sided compress/expand (the last example) requires that both sides (client and server) support compression.

Only certain combinations of `-COMPRESS` and `-EXPAND` are valid with the various USER-Access transfer modes. The following table shows which combinations are valid (Yes) and which are not valid (No):

Transfer Mode	-COMPRESS only	-EXPAND only	Both -COMP/-EXP
CHARACTER	Yes	Yes	Yes
RECORD	No	No	No
STREAM	Yes	Yes	Yes
BACKUP	Yes	No	Yes
RESTORE	No	Yes	Yes
COPY	No	No	Yes
VICHAR	No	No	No

Character Mode Compression

Both sides (client and server) of a CHARACTER mode transfer must support compression (release R10 or later). In addition, when transferring between hosts with different native character sets (e.g. ASCII to EBCDIC) there are some subtle problems caused by the fact that only the USER-Access client performs code conversion.

The character set of the compressed data is stored in the compression header that prefixes the compressed data stream. This information can be used during expansion to determine if code conversion must be performed.

The following table illustrates the various combinations of CHARACTER mode compress/expand. The source and destination file types are shown as well as any code conversion issues:

USER-Access command	Source	Destination	Code Conversion performed
send-compress	text	stream	by client before compress
receive -compress	text	stream	not done - server pushes an informative message - flags its native char set in header

USER-Access command	Source	Destination	Code Conversion performed
send -expand	stream	text	not done - error if char set in header does not match server's char set
receive -expand	stream	text	by client after expand if char set in header does not match client's native char set
send -compress -expand	text	text	by client before compress
receive -compress -expand	text	text	by client after expand

Stratus USER-Access SEARCH Keywords (SITE), (USER), and (NONE)

This section assumes the reader has a working knowledge of the USER-Access commands as well as a general understanding of the SEARCH qualifier for each of them.

The USER-Access commands CONNECT, INPUT, HELP, and TRANSLATE have SEARCH qualifiers associated with them. The SEARCH qualifiers generally instruct USER-Access to look for a file or set of files (depending on the command it is associated with), with the given name and in the specified location. One option the user always has is to give the entire file specification (path and file name) for each path or file that is to be searched. The other option, implemented simply as a shortcut for the user, is to specify the keywords (SITE), (USER), or (NONE), in some form, as the definition for qualifier SEARCH. The SEARCH keywords have the following "generic" definitions:

(SITE) This keyword refers to the USER-Access site or root directory on the local or remote host. On a Stratus host (SITE) is a partitioned data set. (SITE) is where system wide USER-Access files (members) are stored. The actual value for (SITE) can be determined by displaying the "hidden" LOCAL or REMOTE qualifier called ROOTDIR:

```
User> show local rootdir
```

or

```
User> show remote rootdir
```

(USER) This keyword refers to the user's login or home directory (or equivalent) on the local or remote host. On a Stratus host, (USER) refers to the user's login Stratus prefix. Often (USER) is where users keep personal files (such as USER-Access startup files) that are intended for tailoring USER-Access to personal taste. The actual value for (USER) can be determined by displaying the local or remote HOMEdir qualifier.

(NONE) This keyword simply tells USER-Access to not search for any files. It is different than leaving a SEARCH qualifier value blank in that a SEARCH qualifier with no definition often implies some default files should be located. Specifying (NONE) as a definition for SEARCH specifically tells USER-Access not to look for any files.

The keywords (SITE) and (USER) can be used as they appear above, or with a member or file name appended to them respectively. Appending a member or file name to (SITE) or (USER) tells USER-Access to look in the specified location for the given member or file name. When specified without any appended text, the keywords may have implied names that USER-Access attempts to locate. The rest of this section addresses each USER-Access command that has a SEARCH qualifier and how that qualifier interprets the keywords (SITE), (USER), and (NONE).

CONNECT

The SEARCH qualifier for the CONNECT command is used to locate USER-Access startup files on the remote host following a successful connection. The default definition for SEARCH is the string “(SITE) (USER)”. When the connection is being made to a Stratus host, (SITE), when declared without a member name, always refers to the member SSERVER on the remote Stratus host in the (SITE) partitioned data set. When (USER) is declared without a file (or data set) name, it implies a sequential data set by the name of SERVER.UA appended to the user’s TSO prefix. (NONE) tells USER-Access to not process any remote startup files.

For example, to define the CONNECT SEARCH qualifier to locate the default site startup file, the default user startup file, and a third startup file called NEW located in the USER-Access (SITE) partitioned data set, issue the following command:

```
User> set connect search (SITE) (USER) (SITE)NEW
```

The user should be reminded that the order here is important. USER-Access uses the SEARCH definition from left to right. In the example above, (SITE) would be searched first, then (USER), and finally (SITE)NEW.

HELP

The HELP SEARCH qualifier is used to locate USER-Access help files on the local host. The default definition for SEARCH is “(SITE)”. (SITE), when declared without a member name, always implies the member EFTHELP in the (SITE) partitioned data set. (USER) has the default file name HELP.UA associated with it. Users may define SEARCH as a string containing (SITE)xxx or (USER)xxx where ‘xxx’ is the name of a user defined help file, either a member name or sequential data set name, respectively.

For example, issue the following command to define the HELP SEARCH qualifier to look for a user created help file called MYHELP.HLP.UA appended to the user’s TSO prefix, and the default USER-Access help file:

```
User> set help search (USER)myhelp.hlp (SITE)
```

INPUT

The SEARCH qualifier for the INPUT command (explicit or implicit) is used to locate USER-Access input scripts. There are no default file names associated with (SITE) or (USER) for the INPUT command SEARCH qualifier. This means that specifying (SITE) or (USER) without a suffix is the same as specifying (NONE). You can suffix (SITE) or (USER) with any character that will generate a valid Stratus data set name. Including an asterisk (*) in a suffix causes the input file name to be substituted at that point.

Note: Do not specify “(SITE)*” as it will find the distributed help text files and attempt to execute them as the alias commands.

The search order for an explicit INPUT command is first check the user’s directory (Stratus prefix) for a data set with the input file name, and then check the INPUT SEARCH qualifier path(s). The search order for an implicit INPUT command (generated just before the Invalid Command error) skips the user’s directory search and proceeds directly with the INPUT SEARCH qualifier path(s).

For example, the INPUT SEARCH qualifier can be defined to search all the members in the site partitioned data set that end in ‘UA’ by issuing the command:

```
User> set input search (SITE)*UA
```

Now any input requests (via the INPUT command or implied), will only require the member name minus the 'UA'. For example, if a USER-Access input script by the name of MYJOBUA is located in the root directory, it can be invoked as:

```
User> input myjob
```

or

```
User> myjob
```

Note that if a sequential data set with the name TSO PREFIX.MYJOBUA exists, the INPUT command in the first example will process that data set instead, whereas the myjob command will not.

TRANSLATE

The SEARCH qualifier for the TRANSLATE command is used to locate user defined script files. The default definition for SEARCH is "(SITE)". (SITE), when declared without a member name, always refers to a member in the site partitioned data set on the local host with the name {HOSTCODE:REMOTE}, where {HOSTCODE:REMOTE} is the string substitution syntax for the host character code (ASCII8, EBCDIC, etc.) of the remote host. (USER) has no implied file name associated with it for the TRANSLATE command. Users may define SEARCH as a string containing (SITE)xxx or (USER)xxx where 'xxx' is the name of a user defined script file (member name or sequential data set, respectively) containing TRANSLATE commands.

For example, issue the following command to define the SEARCH qualifier for the TRANSLATE command to locate a user created script file called *vostoibm.ua* located in the user's login directory:

```
User> set translate search (USER)vostoibm.ua
```

User-Definable HELP Files Under Stratus

A site has the ability to create its own HELP files that USER-Access will look for upon request. User definable help files allow a site or user to write help text for newly created aliases and input scripts. The HELP qualifier SEARCH is used to tell USER-Access to look for additional help files. By default, the HELP SEARCH qualifier is defined as:

```
User> show help search
User: SEArch ..... (SITE)
```

(SITE) is a special SEARCH keyword used by Stratus USER-Access to indicate the default help file member EFTHELP from the local USER-Access site partitioned data set directory. This help file contains all USER-Access commands, qualifiers, examples, etc. To instruct USER-Access to also look in a site or user defined help file (partitioned data set member or sequential data set), for example, 'JONESX.MYALIAS.UA', type the following:

```
User> set help search 'JONESX.MYALIAS.UA' (SITE)
```

The order of the SEARCH list is important. In the example above, USER-Access will look first in the MYALIAS.UA help file upon any help request. If no match is found there, USER-Access continues on to read the next help file in the SEARCH list, in this case the (SITE) default help file.

It may be desirable to store any site help files along with the USER-Access default one, in the root directory. The (SITE) keyword allows for appending member names as shown below:

```
User> set help search (SITE)myalias (SITE)
```


This example instructs USER-Access to first look for any help requests in the member MYALIAS in the (SITE) partitioned data set, followed by the USER-Access default file EFTHELP from the same pds. Refer to “Stratus USER-Access SEARCH Keywords (SITE), (USER), and (NONE)” on page 92 for more information.

To create a site or user help file, use any standard text editor available on the system. The format of the help file must follow the example below:

```

Topic_Level  Topic_Name
.....
.....Help text.....
.....
Topic_Level  Topic_Name
.....
.....Help text.....
.....

```

Where:

Topic Level is a numeric value from 1 to 9 indicating the help level representing this Topic_Name. The first Topic_Level of all help files must be level 1. Subsequent topics can then be sub-topics (2, 3, etc.) to the level 1 topic, or new top-level topics.

Topic Name is the character string representing the help topic. This is the name the user types to obtain help text (e.g. HELP SEND). This string should be limited to 15 characters in length for output formatting purposes. The Topic_Name may include upper case characters followed by lower case characters in order to allow an abbreviation on the help request.

Help Text is the actual help text the user will see in response to a help request. All help text should be character (text) data only. Users should be careful not to include unprintable characters, including tabs (multiple spaces are recommended) and control characters.

The following is an abbreviated version of the USER-Access standard help file to be used as a reference when creating site or user help files:

1. ASK
 - The ASK command prompts a user for one or more responses...
2. QUALifiers
 - This is where the qualifiers for the ASK command would be described within the help file -- as a sub-topic to the ASK command.
2. EXamples
 - And this is where any ASK examples would be shown. This sub-topic to ASK is the same level as QUALifiers.
1. CONNect
 - The CONNECT command is used to establish a connection to a remote host on the network...
 - (This is a new top-level topic.)
2. EXamples
 - Any examples for the CONNECT command would appear here as a sub-topic.
3. MORE_Examples
 - This is here just to show where a level three help sub-topic would appear.
1. DISconnect
 - And so on...

Note that under a particular top-level topic there may be multiple sub-topics and even sub-topics to them. It is up to the site to make sure these user-written help files are formatted properly. It may be useful to refer to the standard USER-Access help file as a guide.

Running USER-Access as a Batch Job Under Stratus

Since the USER-Access Initiator is a OpenVOS application program, it can easily run as a batch job. The technique for doing this is simple. First, using a standard text editor, create a command macro which invokes USER-Access and performs the required operations (e.g. connect to a remote host, transfer a series of files, and exit). The command macro should use the `attach_input` command while it is in USER-Access. To submit the job to a batch queue, issue the following command:

```
batch '<command_macro_name> [macro_parameters]' [batch_options]
```

macro_parameters These are any optional parameters that the command macro may accept.

batch_options These are any desired arguments to the batch command.

Figure 6 is a sample command macro with embedded USER-Access commands. This sample may be run as a batch job by entering:

```
batch sample.cm

& sample.cm
& Sample Stratus Command Macro that invokes USER-Access and issues
& USER-Access commands. The macro connects to a remote host, sends
& some files, and exits with status.
&

!display_line Entering USER-Access...
&attach_input
&
& Invoke USER-Access
&
user
set input echo on
*
*      Catch any errors and exit with status. Disabling of string
*      substitution using {} is required to return the last status.
*
on error exit {} {status()}
*
*      Display the date and time, then login to remote host and
*      transfer all files with an extension of '.doc'.
*
text Today is {date()} - {time()}.
login sun guest netex
send -create replace *.doc
exit success
&detach_input
#
#      At this point any OpenVOS command may be issued.
#
!display_line Job complete.
```

Figure 6. Sample Command File

Running a Stratus Stand-Alone USER-Access Server

The USER-Access Server (or Responder) is typically invoked by the Service Initiator following a successful login request under Stratus USER-Access. It may be desirable at some time to run the server as a stand-alone task, without it being tied to the Service Initiator. This type of operation is generally desired for batch processing, when login is not crucial, or when the server is invoked from some unprivileged user account.

The USER-Access Stand-Alone Server (or Responder) can be invoked from a OpenVOS command line (interactively or in batch), as:

```
uaserver [-keyword value]
```

uaserver

This is the command to invoke the USER-Access Stand-Alone Server. UASERVER should be defined at installation time. If it is not, contact the site administrator.

-keyword value

This optional parameter specifies optional command line keywords that may be given to affect the operation of the USER-Access session. The following are valid keywords:

- BLOCKsize** specifies an alternative default NETEX blocksize value in which to offer. The default is 16384.
- HOMEdir** specifies the name of the user's "login" or "home" directory when USER-Access is invoked. Changing this keyword's value redefines the location USER-Access uses to locate user startup files. The default is the user's login directory.
- OUTput** specifies the name of an output file that is to receive the server output displays from this session. The default is the terminal or batch log file.
- PASSword** specifies an optional password that will be checked (by the server) after a connection is established to this server. A connecting initiator must pass a password matching in length and exact character case. The default is no password.
- PREfix** an alternative server prefix string that precedes all server displays. The default is "Server:".
- ROOTdir** specifies the name of the installed USER-Access root directory containing site specific server help and startup files. There is generally no reason to modify this keyword. The default is the installed USER-Access root directory.
- SEArch** specifies the search path USER-Access follows to locate server startup files following a successful connection. This value is only used if the CONNECT SEARCH path from the initiator is empty. The default is (SITE) (USER).
- SERvice** specifies a NETEX service name to offer. The default is "USER".

As mentioned earlier, the Stand-Alone Server would generally be run as part of a Stratus batch job. Using the exit status returned by each USER-Access command, special processing can be built into a batch job as seen in the section entitled "Running USER-Access as a Batch Job Under Stratus" on page 96. A Stand-Alone Server could be run in the same manner as described there.

Advanced Stratus Transfer Modes

As discussed in "File Handling Under Stratus USER-Access" on page 39, several file transfer modes are supported: CHARACTER (including V1char), RECORD, STREAM, BACKUP, RESTORE, and COPY. Not all of these modes are supported by all USER-Access implementations. This section attempts to provide more

detail on what these different modes imply and when each might be appropriate. Because of the differences in byte/word addressability among USER-Access hosts, a user may often see differences in file and/or record sizes when transferring files. Some provisions are made in CHARACTER, BACKUP, and RESTORE modes to try to correct for this. In general, though, because of the differences between byte addressability (eight bits on a VAX) versus word addressability (60 bits on a CYBER), there will tend to be some slight variations.

CHARACTER mode is used to transfer text files, such as program source files. USER-Access assumes the file is record-oriented and provides character code conversion between the Stratus ASCII character set and the remote host's native character set. USER-Access also strips trailing blanks from fixed length record files when sending a file and pads fixed length record files with blanks when receiving a file. While -MODE CHARACTER is specified, internally one of two formats will be used for transmitting the data over the network. These two formats are designated internally as *v1char* and *text*. *v1char* is the version 1 USER-Access character mode format while internal *text* mode is the version 2 enhanced format. Each mode involves packing multiple records into a single block for transmission over the network. However, the file access and network block-packing algorithms are different.

Internal *v1char* format (which is identical to -MODE V1char in version 2 of USER-Access, or -MODE CHARACTER in version 1), causes the file to be accessed one record at a time. Each record is preceded by its length in the network block and each block is headed by a block record count.

In internal *text* format (which is identical to -MODE CHARACTER in version 2 of USER-Access), Stratus USER-Access attempts to access the file •a block at a time and pack blocks into one of two formats. A line-feed terminated record format (format < LF>) or a carriage-return, linefeed terminated record format (format < CR-LF >) is used.

The *text* format of CHARACTER mode provides significant performance advantages if one of the hosts in the transfer supports files defined by one of these two formats. *Text* format is generally faster when communicating with other hosts as well due to not needing to encode and decode the record header containing the record length.

When the user specifies mode CHARACTER, Stratus USER-Access internally uses *text* format whenever the remote host supports it, otherwise it uses *v1char* format. *v1char* format may be manually selected at any time by simply specifying -MODE V1CHAR on the SEND or RECEIVE command line.

RECORD mode is used to transfer record-oriented binary files. The file is accessed using record I/O. No code conversion is attempted nor is any other attempt made to reorient the individual records between the hosts. The record can contain a mixture of characters (text), integers, floating point numbers or any other site specific structures. However, USER-Access has no knowledge of this format. The user may need to convert the data once it arrives at its destination, allowing for differences in byte ordering or floating point representation. Note also that records may be padded (with nulls) to some multiple of addressable units at the destination host. For instance, when sending a file between a VAX with eight-bit byte addressability and a machine which has 16-bit word addressability, an odd length record of length 1 byte would be padded to 2 bytes on the 16-bit machine. RECORD mode is not supported on all hosts since not all operating systems support the same record concept, such as OS1100 and NOS (records have different meaning for NOS). A pseudo-record mode has been implemented, however, for UNIX which does not normally support records for non-character data.

STREAM mode is used to transfer files in an unstructured manner. It provides no code conversion nor provides any structure to the network transmission. The file is accessed in block I/O mode and read/written as a continuous stream of bits or bytes. This means that files containing embedded structures (such as variable length record files containing embedded record lengths) will have those structures or headers copied along with the data. It is the user's responsibility to understand those possibly embedded formats and to process them appropriately on the destination host. When sending between like hosts there may be no requirement to interpret these embedded formats. When transferring between Stratus hosts, COPY mode is recommended rather than STREAM, because COPY mode copies the file attributes as well as the data. Block I/O is only

valid for disk files (i.e., SEQUENTIAL, FIXED, RELATIVE, and STREAM). Devices and pipe files are accessed in record I/O mode.

BACKUP mode is similar to STREAM mode but entails copying a file in such a way as to be able to later restore it as it originally existed. The file is stored as a bitstream on the destination host and will be in a format such that it would likely not be able to be processed directly on the destination host. With Stratus it involves accessing the file in record I/O mode, copying the file as a bitstream. BACKUP mode is also used in Central Archiving, as described in the Archiving section of this manual.

RESTORE mode is the counterpart of BACKUP mode. It is used to restore a file which has previously been copied in BACKUP mode. Again, record I/O mode is used to create the file on the destination Stratus system. Again, the file is accessed merely as a bitstream on the originating host. This mode is also used internally as part of the COPY mode for transferring files Stratus to Stratus. RESTORE mode is also used in Central Archiving, as described in the Archiving section of this manual.

COPY mode is a special means of copying files between peer hosts. On Stratus systems, COPY mode allows copying of most any sequential disk file from one Stratus to another Stratus. It is efficient because it essentially uses BACKUP mode on the originating host and RESTORE mode on the destination host. COPY mode requires the *ua_save.cm* and *ua_restore.cm* command macros to capture and restore file attributes.

Central Archiving

Introduction

Central Archiving entails backing up systems over the network into a controlled, central environment. The development of Central Archiving is based on the concept that the files stored on all the systems on a network could be backed up to a central location. If done, the safe guarding of these files is simplified and more assured. The responsibility for the activity can be fixed and procedures can be developed to carry out corporate policies regarding backup and archiving.

Most corporate data centers have established archiving procedures already in place and, except for satellite locations, are executing backup and archival on a routine basis. Central Archiving provides a way to incorporate satellite files into the same process by operating in conjunction with the satellite system's own backup and restore utilities.

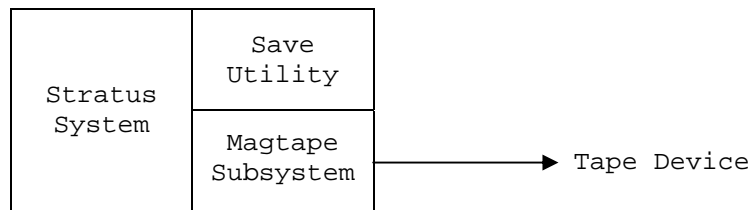


Figure 7. Traditional Backup

As depicted in Figure 7, a traditional backup archives Stratus files to a local device, generally tape. Figure 8 diagrams Central Archiving across the network. In a Stratus-zOS environment, Central Archiving captures the output of the Stratus save command and sends that output via USER-Access to the zOS site's disk or tape drive. The satellite files being transferred are combined into one container file on the central side which "looks like" a locally-generated save file; that is, it is readable only by the Stratus restore facility through USER-Access.

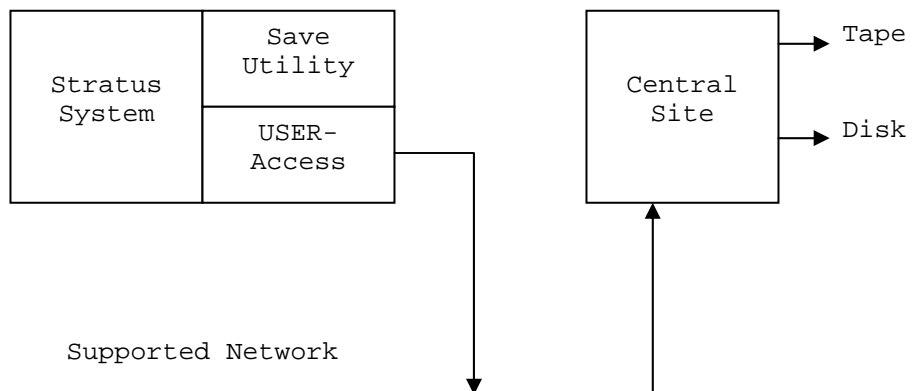


Figure 8. Central Archiving

Central Archiving can be invoked from either the satellite system or from the central site. When initiated by a satellite system, Central Archiving is a variation of a simple SEND command. But the implication is that the activity will be controlled better if it is the central site's scheduling utility that initiates Central Archiving.

When invoked by the central site, the script includes a logon to the satellite system followed by a RECEIVE command that is modified to trigger Central Archiving.

Users of satellite systems can make use of Central Archiving in any of several ways:

- They can format and execute their own SEND and RECEIVE commands as described in “Archiving from a Satellite Site” on page 104.
- They can make use of the USER-Access aliases BACKUP, LIST, and RESTORE, described in “

The Central Archiving Aliases” on page 111.

- They, or site administrators, can create and execute aliases or command procedures tailored for the site.

Central site operators can also make use of Central Archiving in any of several ways:

- They can format and execute their own SEND and RECEIVE commands as described in “Archiving from a Satellite Site” on page 104.
- They can invoke the satellite host procedures provided for the various systems supporting archiving.
- They can create and execute aliases or command procedures tailored for the site.

It is often desirable to invoke and control Central Archiving from a central site’s job scheduling facility. Because USER-Access provides status indications to the requestor (messages to a user or operator, status codes to a procedure) the central job scheduler can be used effectively to control the archiving process; these procedures are described in “Archiving from a Central Site”.

Using Archiving

Archiving under OpenVOS looks like a normal USER-Access file transfer. An archive can be invoked either from the satellite site, by issuing a SEND command, or from a central site, by issuing a RECEIVE command. Two transfer modes exist to facilitate the backup and restore of Stratus OpenVOS files. BACKUP mode archives Stratus files. RESTORE mode restores Stratus files that were previously archived.

Archiving under OpenVOS requires the use of three OpenVOS command macros released with USER-Access in the USER-Access Root Directory (‘(master_disk)>NETEX>sicom>user’):

- ua_save.cm
- ua_restore.cm
- ua_list.cm

Each of these command macros contains the OpenVOS commands necessary for the OpenVOS SAVE/RESTORE utilities to transfer data through USER-Access.

The command macros listed above provide the ability to pipe the data from a SAVE command, or pipe data into a RESTORE command using the named pipe facility within USER-Access. The special “(PIPE)” tag indicates that USER-Access should create a temporary pipe file. The Stratus OpenVOS SAVE/RESTORE commands exchange data with USER-Access by writing and reading from this pipe file. An example of a backup using the UA_SAVE command macro looks like:

```
User> send -mode backup "!ua_save (PIPE) *" destination_file
```

In this example USER-access scans the OpenVOS command line “ua_save (PIPE) *” for the “(PIPE)” tag, if it finds the tag it creates a temporary pipe file and replaces the “(PIPE)” tag with the name of the pipe file. USER-Access then invokes the OpenVOS command and transfers data from the pipe file. Within the command macro UA_SAVE.CM the output from the OpenVOS SAVE utility is directed to the pipe file.

A restore works very similar to a backup:

```
User> receive -mode restore source_file "!ua_restore (PIPE)”
```

In this example USER-Access pipes the data coming off the network into the executing OpenVOS command. In this case the OpenVOS command is the UA_RESTORE.CM command macro. Again a pipe file is used to move data from USER-Access into the executing OpenVOS RESTORE utility.

A listing and short description of each of the command macros provided with this release of USER-Access are at the end of this section.

The following is a diagram of Central Archiving both from a satellite and central site's perspective. The diagram depicts the relationship between the SEND and RECEIVE commands, the file transfer modes SAVE and RESTORE, and the source and destination file specifications.

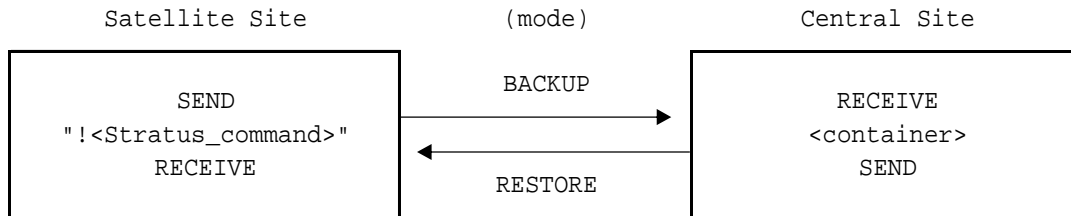


Figure 9. Satellite and Central Sites

For example, a Stratus satellite site would backup to a central site with the command:

```
SEND -MODE BACKUP "!<Stratus_command>" <container>
```

A Stratus satellite site would restore from a central site with the command:

```
RECEIVE -MODE RESTORE <container> "!<Stratus_command>"
```

A central site would backup a Stratus satellite site with the command:

```
RECEIVE -MODE BACKUP "!<Stratus_command>" <container>
```

A central site would restore a satellite site with the command:

```
SEND -MODE RESTORE <container> "!<Stratus_command>"
```

The string '<Stratus_command>' is an existing Stratus command procedure containing a Stratus SAVE/RESTORE command. The '!' and double quotes are required around the specification '<Stratus_command>' to tell USER-Access to spawn the specified procedure (a Stratus command macro). The string '<Stratus-command>' must be a command macro accepting at least one argument.

```
<Stratus_command> == "command_macro_name (PIPE)"
```

Where the string "command_macro_name" is the name of the OpenVOS command macro. The (PIPE) tag must be passed as one of the arguments to the command macro. USER-Access replaces the (PIPE) tag with the name of the temporary pipe file. The pipe is created in the user's process directory. USER-Access then invokes the command macro. It is the job of the command macro to attach a port to the temporary pipe and use the port as input device for a OpenVOS RESTORE command or the output device for the OpenVOS SAVE command.

Archiving from a Satellite Site

Archiving from a satellite site implies initiating both the backup and restore operations from a local Stratus system. A backup is initiated by using the SEND command. A restore is initiated using the RECEIVE command. The following is a description of each operation from a satellite site's perspective.

In the following example, all of the files in the current Stratus directory tree are archived to the remote system and stored in the container file called 'guest.bck' using the backup macro ua_save.cm

```
User> send -mode backup "!ua_save (PIPE) *" guest.bck
```

The source specification is a valid Stratus save procedure. The output of the save command in this procedure is written to a pipe file. In this case, the pipe file is used as input for the USER-Access SEND command. Any data written to the pipe file is captured by USER-Access and sent across the network as an Archive file.

In the following example, the remote container file called 'guest.bck' is disassembled by the local restore utility and the files are restored to disk:

```
User> receive -mode restore guest.bck "!ua_restore (PIPE)"
```

The source specification 'guest.bck' is the remote container file. The destination specification is a valid Stratus restore procedure. In this example USER-Access receives the remote file 'guest.bck' and writes the incoming data to a pipe file. The Stratus restore command then disassembles the data.

All valid options of save and restore commands may be used in conjunction with the USER-Access RECEIVE and SEND commands. This helps the user to tailor archives to their specific needs. General procedures may be created which accept path names and additional parameters are input. For example, ua_restore could be written to accept path names as input. The following would allow only program modules (*.pm) to be restored from the container file:

```
User> receive -mode restore "!ua_restore (PIPE) *.pm" guest.bck
```

USER-Access transfers the container file to the satellite site, where the Stratus restore command extracts the selected files. There is no staging of the data on the Stratus satellite host. Generally a pipe file will only be a few blocks during processing.

Four aliases are predefined with USER-Access. The aliases are BACKUP, COPY, LIST, and RESTORE. Each of these aliases provides a simple interface to the USER-Access archiving capabilities. For more information on each of these aliases refer to "The Central Archiving Aliases" on page 111.

The command macros described in this section are provided as part of the USER-Access release and can be found in the Stratus root directory ">NETEX>SICOM>USER".

Archiving from a Central Site

One of the largest benefits of Central Archiving is the expanded use of data center resources. A central data center can provide the personnel, procedures, and peripherals for archiving remote sites. It is rare to find all three in existence at each satellite site. USER-Access and Central Archiving provide the interface for sharing the resources of a central data center with remote satellite facilities. The figure below depicts this concept:

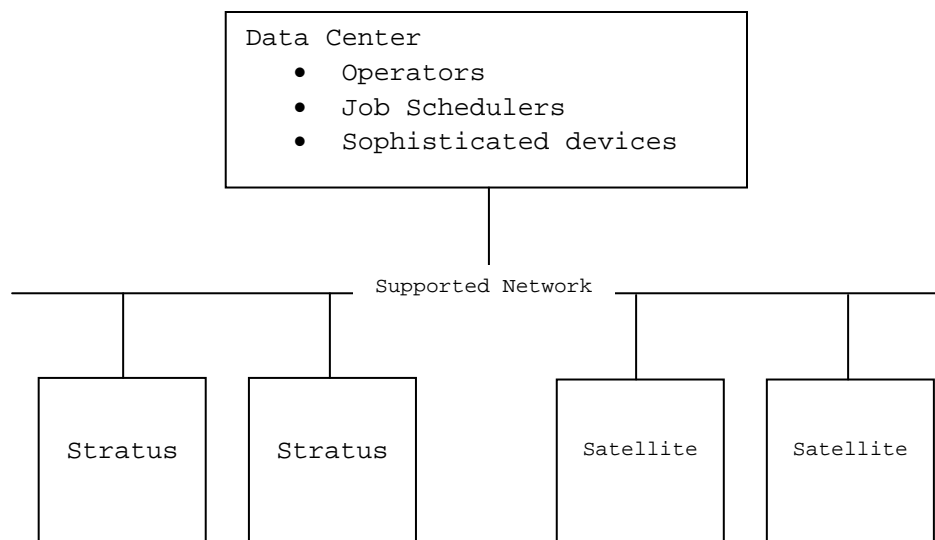


Figure 10. Data Center

A central data center can invoke archive procedures to backup the remote Stratus sites. A central site's job scheduler can automatically start a Stratus host backup, directing the Stratus save file to a central system peripheral (e.g. tape, disk, ...). Central procedures can be set up to check status of each job step, verifying that the backup is operating successfully.

Since many of the functions of archiving from the central site depend on the specific central host type, only a general overview can be presented here. One general approach is described below.

Assume the satellite site is a Stratus running OpenVOS, and the central site is some large mainframe with the resources and procedures to maintain and organize the archives of several satellite sites.

On the satellite site, create a Stratus Archive procedure that directs its output through a pipe file (described earlier). The output from the `-log` parameter qualifier, on the Stratus save command, should be saved to a Stratus file. Also, a separate error log file should be saved for quick reference should any errors occur. An error log can be captured by issuing a `start_logging` command prior to starting the Stratus Archive procedure.

On the central site, create a command procedure, or JCL job that first allocates a local tape or disk device, and then invokes USER-Access with input. USER-Access connects to the remote satellite Stratus host, and then invokes a RECEIVE command to start the archive. An example of what the RECEIVE command could look like is:

```
receive -mode backup "!ua_save (PIPE) *" stratus.bak
```

The remote Stratus procedure 'ua_save.cm.' is invoked as the source of the file transfer. Because (PIPE) appears in the source specification, the special pipe file is created and initialized for use. The 'ua_save.cm' command macro uses the pipe file as the output device for the OpenVOS SAVE utility. Figure 11 on page 107 is an example of the central procedure; the procedure is only a skeleton of what could actually exist. The central site procedure does not detail any system specific commands, since knowledge of the central site's host type is limited.

```

                                [initial JCL]

    ALLOCATE TAPE DEVICE "TAPE"

                                [additional JCL]

USER
  * On error exit with an error status
  *
  ON ERROR exit Error
  *
  * Connect to the remote satellite site (Stratus)
  *
  CONNECT Stratus USERNAME PASSWORD
  *
  * Invoke the remote archive procedure and receive
  * the archived files into the local container file
  * '{host:rem}.{date(1)}'. Receive the log file after
  * the backup is complete.
  *
  RECEIVE "!ua_save (PIPE)*" {stratus.container} -
    -CRC -MODE BACKUP
  RECEIVE LOG FILE
  RECEIVE ERROR_LOG
  *
  * Exit with a successful status
  *
  EXIT Success

                                .
                                [additional JCL]
                                .

```

Figure 11. Sample Central Site JCL Command Procedure

USER-Access is implemented in such a way that each command returns status that can be checked by the central site either within USER-Access or within the central host's JCL. The interaction offered by USER-Access allows a central site to set up closed-loop operations with a Stratus or a group of Stratus'. A central site operator, or job scheduler, receives much the same feedback from the Stratus task as it does from a non-networked local job. This feedback allows the operator/scheduler to employ retry procedures, delay the execution of jobs dependent on a failed job, and notify personnel as required.

File Transfer Options

This section discusses the USER-Access SEND and RECEIVE qualifiers that are of particular interest for Archiving. They are discussed here as they relate to Archiving. More discussion can be found in "File Handling Under Stratus USER-Access" on page 39.

CRC The USER-Access file transfer commands, SEND and RECEIVE, allow for a CRC qualifier. If the CRC qualifier has a value of 'ON', a cyclic redundancy check is done for each block of data transferred. If an error is detected the transfer is aborted and notification is given to the user. An example backup from a central site with CRC active is:

```

User> receive -mode backup "<restore-command-file>"-
More>> guest.bck -CRC

```

FLOW Central Archiving may require writing the remote container file to tape. USER-Access has a FLOW qualifier to allow for suspended file transfers. It is primarily provided to allow for delays in tape handling when writing directly to tape on a central site. The FLOW qualifier uses additional “handshake” protocol to prevent one side of a connection from “over running” the other side. An example backup using the FLOW qualifier is:

```
User> receive -mode backup "<restore-command-file>"-  
More>> guest.bck -crc -FLOW
```

A performance penalty is paid when the FLOW qualifier is used. It is highly recommended that an alternative method be used. Refer to “NETEX Considerations for Archiving.”

SPACE When a SEND command is issued from the local Stratus OpenVOS host, information is exchanged between the two connected hosts. One piece of information that is exchanged is the size of the file to create on the destination host. Because this is an archive of several files a total size is not known unless the user explicitly passes one. Both the SEND and RECEIVE commands have a SPACe qualifier for passing an initial allocation. The following is an example of an archive command with the SPACE qualifier:

```
User> receive -mode backup "<restore-command-file>"-  
More>> guest.bck -crc -flow -SPACE 1M
```

An initial allocation of 1 Megabyte is given for the remote container file ‘guest.bck’.

Stratus Privileges Required for Archiving

No special privileges are required to do basic backups of accessible files on the Stratus. Additional privileges may also be required to use certain SAVE options, but they are the same ones needed to run the SAVE utility directly. Besides privileges, the user must also be aware of their maximum processes. A subprocess is started as part of Central Archiving. This subprocess takes away from total number of processes on that module.

NETEX Considerations for Archiving

The user has the option on most central sites of backing up to a disk file or tape file. In general, when writing to tape at a central site, it is recommended that the central site’s NETEX datatimer or read timeout be disabled (or set as large as possible). The standard USER-Access file transfer protocol is a continuous one which does not allow for long delays that might occur during tape handling. By turning these timers off on the site, receiving the data one can avoid NETEX errors being generated and NETEX dropping the connection. If this is not acceptable, there is also a parameter that may be set via SET SEND FLOW ON which provides a simple flow control in the send protocol (or “set receive flow on” during receive) which will permit indefinite delays in tape handling. However, use of this parameter will negatively impact the performance of the backup or restoration process.

Stratus as a Central Site

A Stratus system can also be a central site to other Stratus computers or workstations, assuming a central Stratus site exists that can provide the necessary resources, such as operators, peripherals, and command procedures.

The Central Archiving OpenVOS Command Macros

The OpenVOS Command Macros UA_SAVE.CM, UA_RESTORE.CM, and UA_LIST.CM are provided as part of the Stratus USER-Access release: They function as the interface between the OpenVOS SAVE/RESTORE utilities and USER-Access. Instead of reading or writing to a tape device (i.e. #mt1.0) the command macros use the pipe file provided by USER-Access. Each of the three macros is described on the following pages.

UA_SAVE.CM OpenVOS Command Macro

Description

The UA_SAVE.CM Command Macro is used to archive Stratus files by redirecting the data output from a OpenVOS SAVE command to a named pipe file provided by USER-Access. The following is the contents of the UA_SAVE.CM command macro:

```
&begin_parameters
    pipe    pipe_pathname:pathname,required
    rest    unclaimed
&end_parameters

!attach_port $$UA_TAPE$$ &pipe&
&attach_input
save $$UA_TAPE$$ &rest&
&set status (command_status)
&if &status& != 0
&then &goto save_failure
yes
&set status (command_status)

&label save_failure
&
&detach_input
!detach_port $$UA_TAPE$$
&
&return &status&
```

Examples

The following command archives all of the OpenVOS files in the current directory to the remote container file 'files.bck':

```
User> send -mod backup "!ua_save (PIPE) *" files.bck
```

If the backup is being invoked from the Central Site the following RECEIVE command can be used to backup all the OpenVOS files on the volume '#m1_d01' to the local container file 'volume.bck':

```
User> receive -mod backup "!ua_save (PIPE) #m1_d01" volume.bck
```

UA_RESTORE.CM OpenVOS Command Macro

Description

The UA_RESTORE.CM Command Macro is used to restore OpenVOS files by redirecting the data input to a OpenVOS RESTORE command from a named pipe file provided by USER-Access. The following is the contents of the UA_RESTORE.CM command macro:

```
&begin_parameters
    pipe    pipe_pathname:pathname,required
    rest    unclaimed
&end_parameters

!attach port $$UA_TAPE$$ &pipe&
!restore $$UA_TAPE$$ &rest&
&set status (command_status)
!detach_port $$UA_TAPE$$

&return &status&
```

Examples

The following is an example of how to restore all of the files contained in the remote container file 'files.bck':

```
User> receive -mode restore files.bck "!ua_restore (PIPE)"
```

A similar command could be issued from the Central Site:

```
User> send -mode restore files.bck "!ua_restore (PIPE)"
```

UA_LIST OpenVOS Command Macro

Description

The UA_LIST.CM Command Macro is used to list the contents of a OpenVOS container file. The UA_LIST.CM Command Macro uses the 'list_save_tape' utility to extract the directory of files from a container file. The following is the contents of the UA_LIST.CM Command Macro:

```
&begin_parameters
    pipe    pipe_path_name:pathname,required
&end_parameters

!attach_port $$UA_TAPE$$ &pipe&
!list_save_tape $$UA_TAPE$$
!detach_port $$UA_TAPE$$
```

Example

The following provides a list of the files in the remote container file 'files.bck':

```
User> receive -mode restore files.bck "!ua_list (PIPE}"
```


The Central Archiving Aliases

The BACKUP, COPY, LIST, and RESTORE aliases are provided in the default USER-Access site startup file for Stratus (as a satellite system). Their purpose is to simplify the interactive use of the USER-Access Central Archiving facility. These aliases are described on the following pages. Keep in mind that any of them may be changed to address specific site concerns.

BACKUP Alias

Description

The BACKUP alias is used to archive local files using the Stratus BACKUP utility. The output from the utility is transferred to the specified remote container file. The additional parameter allows SEND qualifiers, such as -CRC, to be appended to the SEND command in the BACKUP alias definition.

All of the parameters are optional. If the parameters are not passed on the command line, they are prompted for by the BACKUP alias.

Format

Alias	Parameters
BACKUP	[<i>source</i>] <i>container</i> [<i>qualifiers</i>]

- source** This optional parameter specifies a local file specification to be archived. The default is all files in the current directory and those in directories beneath it.
- container** This parameter specifies the name of the remote container file to be created. This parameter has no default and must be included for the alias to successfully complete.
- qualifiers** This optional parameter represents additional qualifiers that are appended to the file transfer command.

Example

Assume the current local directory is guest. All of the files in the guest directory, including sub- directories, are backed up to the container file “guest.bck” on a remote system with the following command:

```
User> backup
Source specification? *
Container file name? guest.bck
Other send qualifiers? -crc
%dra#d07>guest>PIPE_16962073 already exists.
Overwrite it? (yes no)
Begin save:89-12-11 18:22:20 est
The common path name for this save is "%dra#d07>guest"
End of save
User:
User: Source Destination Size
User: -----
User: ua_save (PIPE)* guest.bck 195420
User:
```

The above command can also be issued by passing the parameters on the command line:

```
User> backup * guest.bak -crc
```

LIST Alias

Description

The LIST alias provides a complete index of a remote container file created using the BACKUP alias. A list of the contents of the container file is returned.

All of the parameters are optional. If the parameters are not passed on the command line, they are prompted for by the LIST alias.

Format

Alias	Parameters
LIST	<i>container</i>

container This parameter specifies the name of the remote container file to be listed. This parameter must be specified if the alias is to complete successfully. There is no default.

Example

Suppose a number of local files are archived in the remote container file “guest.bck.”. The LIST alias provides an index of the specified container file:

```
User> list
Container file name? guest.bck

%dra#m2_d01>guest>PIPE_16962093 already exists.
Overwrite it? (yes, no)
Begin restore of save dated: 89-09-10 19:23:31 edt
Restore source is: %dra#m2_d01>guest
89-09-10 20:06:57 edt file %dra#m2_d01>guest>client.ua
89-09-10 20:06:57 edt file %dra#m2_d01>guest>start_up.cm
89-09-10 20:06:57 edt file %dra#m2_d01>guest>newout.ua
89-09-10 20:06:57 edt file %dra#m2_d01>guest>new.ua
89-09-10 20:06:57 edt file %dra#m2_d01>guest>update stratus
89-09-10 20:06:57 edt file %dra#m2_d01>guest>val_stratus
89-09-10 20:06:57 edt file %dra#m2_d01>guest>gosh.cm
End of restore, last saved object: 89-09-10 19:23:32 edt

User: Source                Destination                Size
User: -----
User: guest.bak             !ua_list (PIPE)          794624
```

The LIST alias can also be executed by passing the parameters on the command line:

```
User> list guest.bck
```

RESTORE Alias

Description

The RESTORE alias is used to restore local Stratus files from a remote container file created using the BACKUP alias. The local OpenVOS RESTORE utility is used to disassemble the container while the file is being retrieved from the remote host.

All of the parameters are optional. If the parameters are not passed on the command line, they are prompted for by the RESTORE alias.

Format

Alias	Parameters
RESTORE	<i>container</i> [<i>source</i>] [<i>destination</i>]

container This parameter specifies the name of the remote container file. This parameter must be specified if the alias is to complete successfully. There is no default.

source This optional parameter specifies a local file specification to be restored. The default is to restore all of the files in the container file.

destination This optional parameter specifies where the files should be restored into. The default is to restore starting from the current directory on the local Stratus host.

Example

Suppose all of the files in the [GUEST] directory are archived in a remote container file named “guest.bck”. The files can be restored as shown in the following example.

```
User> restore
Container file name? guest.bck
File(s)to restore (all)?
Destination (current_dir)?

%dra#m2_d01>guest>PIPE 16962073 already exists
Overwrite it? (yes,no)
Begin restore of save dated:89-09-18 19:23:31 edt
Restore source is: %dra#m2_d01>guest
End of restore, last saved-object: 89-09-10 19:23:32 edt

User: Source                Destination                Size
User: -----
User: guest.bak             !ua_list (PIPE) *        15848
User:                       (current_dir)
User:
```

The above command can also be issued by passing the parameters on the command line:

```
User> restore guest.bck
```


Command Descriptions

This section contains descriptions of these commands:

ASK
CONNECT
CONTINUE
DISCONNECT
EXIT
GOTO
HELP
INPUT
LOCAL
ON
OUTPUT
QUIT
RECEIVE
REMOTE
SEND
SET
SET ALIAS
SET GLOBAL
SET HOST
SET VARIABLE
SHOW
SHOW ALIAS
SHOW GLOBAL
SHOW HOSTS
SHOW QUALIFIER
SHOW VARIABLE
TEXT
TRANSLATE

The command descriptions or qualifiers for some commands may differ slightly between hosts. These variations are detailed in the software reference manual for that host.

ASK Command

Description

The ASK command prompts a user for one or more responses. If multiple responses are desired, the command line should contain multiple variables to receive the user's input. For example, if you are prompting a user for name and number, you should declare two variables (e.g., uname, unum) on the ASK command line in order to save both responses. If a user enters a carriage return (or <ENTER>), the variable gets defined to a null string unless a default value is supplied with the DEFAULT qualifier.

ASK variables get set to whatever the user types as input, whether it be one word or an entire string. If multiple variables are declared, the first one is set to the first word of input, the second one gets set to the second word of input, and so on. The last variable declared gets defined to the remainder of the input string. If you prompt for more input than the user gives (e.g., you declare four variables and the user types just two words of input), the remaining variables are defined to be a null string.

Variable names specified on the ASK command line must be alphanumeric and no longer than 20 characters in length. If no variable names are specified at all on the command line, ASK still prompts the user but no variables get defined. (This can be used to pause during input processing).

In addition to the ASK command, you can define a variable by typing SET VARIABLE name value. You can display the list of all of your session variables by typing SHOW VARIABLES.

Format

Command	Qualified	Parameter
ASK	<code>[-DEFault string]</code> <code>[-PROMpt string]</code> <code>[-SECure ON]</code> <code> Off </code> <code>[-TIMEout seconds]</code>	<code>[VAR1 [VAR2...]]</code>

Where:

- ASK** (required) the verb for this command.
- DEFault** (optional) the default string passed to ASK if the user does not provide one. This default string gets processed as if the user had typed it. The minimum spelling is -DEF.
- PROMpt** (optional) a string used for the ASK prompt. You need to enclose the PROMPT string in double quotes in order to include trailing spaces on the prompt or to use multiple word prompts. The minimum spelling is -PROM.
- SECure** (optional) tells USER-Access not to echo the user's response to this ASK command. SECURE is used primarily for reading in a user's password or any other time security is a concern. This value should be set to either ON or OFF. The default is OFF unless SECURE is specified. The minimum spelling is -SEC.
- TIMEout** (optional) tells USER-Access how long to wait, in seconds, for a response. This integer value is between 0 and 10000. The minimum spelling is -TIME. The default value is 0.
- variables** (optional) zero or more variable names separated by a space that will receive the user's response to the ASK command.

Examples

To prompt a user for input into variable name with a default name of Ed, and a prompt of Name?, type:

```
User> ask -prompt "Name? " -default Ed name
Name? Joe Smith
```

The user's response here Joe Smith is read into variable name. Now using standard USER-Access string substitution syntax, you can display the value of name with the TEXT command:

```
User> text Hello {name}.
User: Hello Joe Smith.
```

Alternatively, you can display the value of variable name: as:

```
User> show variable name
User: NAME ..... Joe Smith
```

If you wanted to prompt for the variables day and date, you could type:

```
User> ask -prompt "Enter day and date: " day date
Enter day and date: Tuesday August 10, 1987
```

The value of variable day would become Tuesday and the value of variable date would be the remainder of the line which is the string August 10, 1987.

Related Topics

INPUT Command
SHOW Command
TEXT Command

CONNECT Command

Description

The CONNECT command is used to establish a connection to a remote host on the network. The host name specified must exist in the local network configuration. By default, CONNECT attempts to connect to the service named *USER* offered on the remote host. If this service is not offered the connection request will eventually time out. A remote Service Initiator or Multiplex Server is usually the one making the service offering. You can set or display the service name with the SET CONNECT SERVICE and SHOW CONNECT SERVICE commands respectively. The recommended way for connecting to a remote host is to use the LOGIN or LOGON alias. These aliases prompt the user for a hostname, username, and password, (and account and profile for LOGON) then issues the appropriate CONNECT command for the user.

The connect process actually logs you in to the remote host using its standard login procedure; valid usernames and passwords must be provided to accomplish this. Some systems provide a default USER-Access login username and password. If the system you are connecting to supports this feature, you could optionally leave off these two parameters.

When a connection is first established it becomes the current “active” connection -- any REMOTE command refers to the host associated with that connection. When a connection to another host is attempted, the old “active” connection is put into an idle state. USER-Access allows as many as ten connections to exist at a time, although one or two connections is normally all that a user would have use for. You can switch from one connection to another by means of the SET HOST command.

Connecting to certain hosts often takes more than a few seconds. Therefore, USER-Access displays intermediate CONNECT messages informing you of its current connect status. If a connection cannot immediately be established due to the service not being offered on the specified host or because the service is busy, CONNECT will retry every INTERVAL seconds for a total of TIMEOUT seconds (INTERVAL and TIMEOUT are CONNECT command qualifiers). Intermediate connect messages display both success and failure information.

Format

Command	Qualifier	Parameter
CONnect	[-ACCCount code] [-ADAPTer string] [-APPlication string] [-BLOCKsize bytes] [-COMmand name] [-INTerval seconds] [-PASSword pw] [-PROFile name] [-PROJect code] [-QUIet ON] OFF [-SCRipt filename] [-SEArch string] [-SECondary pw] [-SERvice name] [-SITE string] [-TIMEout seconds] [-USERname user] [-VERBose ON] OFF	host [user] [password] [arg ...]

Where:

- CONnect** (required) the verb for this command. The minimum spelling is CON.
- ACCCount** (optional) login account code that may be used by the host in which you are attempting to connect. The minimum spelling is -ACC.
- APPlication** (optional) identify the login application on the remote host. The minimum spelling is -APP.
- BLOCKsize** (optional) local host's maximum network block size in bytes. This size gets sent to the remote host on a CONNECT and a new negotiated NetEx block size gets returned. The REMOTE "information only" BLOCKSIZE qualifier contains this new value. The accepted range of values for BLOCKSIZE on a Stratus system is 2048 to 65535 bytes. Most systems will only support a maximum BLOCKSIZE of 32768. The Stratus default is 16384. The minimum spelling is -BLOCK.
- COMmand** (optional) startup command file name that may be used by the host in which you are attempting to connect. The minimum spelling is -COM.
- INTerval** (optional) connect retry interval in seconds used for connection retries when connecting to a remote host. The accepted range of values for INTERVAL is 1 to 60 seconds. The default is to retry every 5 seconds. The minimum spelling is -INT.
- PASSword** (optional) default login password that is used to validate a user on a remote host during a connect. PASSWORD is usually overridden on the command line of the CONNECT command. This qualifier must be used in conjunction with the -USERname qualifier. The minimum spelling is -PASS.
- PROFile** (optional) startup profile file name that may be used by the remote host during a connect. The minimum spelling is -PROF.
- PROJect** (optional) login project code that may be used by the remote host during a connect. The minimum spelling is -PROJ.

- QUIet** (optional) tells USER-Access whether or not to display intermediate connect messages. This value should be set to either ON or OFF. The default is OFF. The minimum spelling is -QUI.
- SCRipt** (optional) script file name that is used by some remote hosts during the connect and login process. The minimum spelling is -SCRI.
- SEArch** (optional) describes the server startup files to be read during connect time. Refer to “Stratus USER-Access SEARCH Keywords (SITE), (USER), and (NONE)” on page 92 for more information. The default is “(SITE) (USER)”. The minimum spelling is -SEA.
- SECondary** (optional) secondary login password that can be used by the remote host during a connect. The minimum spelling is -SEC.
- SERvice** (optional) service name that USER-Access tries to connect to on the remote host during a connect. This service name is USER by default. The minimum spelling is -SER.
- SITE** (optional) site-specific login information that may be used by the remote host at connect time.
- TIMEout** (optional) connect timeout value in seconds. If a connection cannot be established within TIMEOUT seconds you will receive an error message from USER-Access. The default value is 2 minutes. The accepted range of values is 0-32767. The minimum spelling is -TIM.
- USERname** (optional) default login name of the user attempting to connect. USERNAME is usually overridden on the command line of the CONNECT command. This qualifier must be used in conjunction with the -PASSWORD qualifier. The minimum spelling is -USER.
- VERBose** (optional) when this qualifier is set to ON, login information returned from the remote host is displayed to the local user. When this qualifier is OFF, the login information is not displayed. The default is ON. The minimum spelling is -VERB.
- host** (required) name of the remote host to which you want to connect.
- username** (optional) your login user name on the remote host. This is the positional version of the -USERNAME qualifier.
- password** (optional) your login password on the remote host. This is the positional version of the -PASSWORD qualifier.
- arg** (optional) any number of argument strings that get passed along to the remote host at connect time.

Host Dependencies

Many of the CONNECT qualifiers are treated differently depending on the host to which you are connecting. Also, the optional arguments are both host and site dependent. Refer to the remote host’s “Remote User’s Guide” for further detail.

Examples

To connect to a host named sun: in the NCT with a user name of smith and a password of “allen”, type:

```
User> connect sun smith allen
User: Connected to Service Initiator on host 'SUN'
=====
                Welcome to SUN SYSTEM
                *      *      *
=====
User: logged in as user 'smith'.
User: Connected to service 'USER31' on host 'SUN'
```

To connect to host vax with a user name of jones, a password of “jane”, a blocksize of 4096 bytes, and connect timeout of 10 seconds, you would type:

```
User> connect -blocksize 4096 -timeout 10 -
More>> vax jones jane
User: Connected to Service Initiator on host 'VAX'
=====
                Welcome to VAX VMS
                *      *      *
=====
User: Logged in as user 'jones'.
User: Connected to service 'USER01' on host 'VAX'
```

An alternative way of setting the blocksize and timeout qualifier values would have been to issue the following:

```
User> set connect blocksize 4096
User> set connect timeout 10
User> connect vax jones jane
User: Connected to Service Initiator on host 'VAX'
=====
                Welcome to VAX VMS
                *      *      *
=====
User: Logged in as user 'jones'.
User: Connected to service 'USER01' on host 'VAX'
```

This second approach would cause the CONNECT default values to be changed for all subsequent connects during this USER-Access session whereas the first approach would only affect the connect being issued.

Related Topics

DISCONNECT Command
SET HOST Command
SHOW HOSTS Command

CONTINUE Command

Description

The CONTInue command is a no-op command. Its most useful purpose is to provide an action for the ON command when the particular exception is to be ignored.

Format

Command	Qualifier	Parameter
CONTInue		

Where:

CONTInue (required) is the verb for this command. The minimum spelling is CONT.

Example

In this example the CONTInue command is used as the action part of an ON ERRor command. In the following script any errors are completely ignored:

```
* Sample USER-Access script - this script
* continues should any USER-Access error occur
*
on error continue
set variable count 1
LOOP:
send file{count}
set variable count {inc(count)}
{le(count, 5, "goto LOOP")}
```

Related Topics

ON Command

DISCONNECT Command

Description

Terminate the connection from the current remote host which was previously connected with CONNECT. Following a DISCONNECT, you will not have any “active” remote connections even though you may still be connected to other hosts. Use the SHOW HOSTS command to display all of your remote connections. The SET HOST command can be used to make an idle connection active again.

An implied disconnect of all connections takes place following an EXIT or QUIT command from a USER-Access session.

Format

Command	Qualifier	Parameter
DISconnect	[-QUIet ON] OFF]	

Where:

DISconnect (required) the verb for this command. The minimum spelling is DIS.

-QUIet (optional) tells USER-Access whether or not to confirm the disconnect with a message. This value should be set to either ON or OFF. The default is OFF. The minimum spelling is -QUI.

Examples

Assume connections have already been made to the hosts VX1 and IBM. This results in the following output from SHOW HOSTS:

```
User> show hosts
User:          (1) Host=VX1      User=scott
User: active --> (2) Host=IBM    User=meyers
```

A DISCONNECT at this point terminates the current “active” connection (host IBM) as seen below:

```
User> disconnect
User: Disconnected from host IBM.
User> show hosts
User:          (1) Host=VX1      User=scott
```

The connection to IBM has been terminated and only the connection to host vx1 remains (still idle). To disconnect from it, you would have to use SET HOST, then DISCONNECT as seen below:

```
User> set host vx1
User> disconnect -quiet
```

When the QUIet qualifier is used, the message confirming disconnect is not displayed.

Related Topics

CONNECT Command
SET HOST Command
SHOW HOSTS Command

ENCRYPT Command

The ENCRYPT command is an alias defined in the distributed SCLIENT startup file to simplify the use of the ENCRYPT string function.

Format

Command	Qualifier	Parameter
ENCrypt		

Where:

ENCrypt (required) the verb for this command. The minimum spelling is ENC.

The ENCRYPT alias is defined as:

```
set alias ENCrYpt {} {dfn(1, "goto skip")} !
ask -secure -prompt "Enter password? " 1 !
ask -prompt "Enter optional username (or '*'?) " 2 !
skip: set global pw {encrypt(1, upper(2))} !
text The encrypted password is {pw}
```

Note the following items regarding the ENCRYPT alias:

- The password is prompted in -SECURE mode to avoid displaying on the terminal.
- The ENCRYPT alias can be invoke with 'password' and optional 'username' passed as alias parameters to avoid prompting. However, the password will display.
- The optional 'username' is forced to uppercase using the UPPER string function.
- The resulting encrypted password is stored in a global variable PW for later reference.

Examples

Example 1: Encrypting Passwords Stored in a USER-Access Input Script File

Suppose a job running under the local Stratus username NSC.JONES inputs the USER-Access script >nsc>sicom>user>scripts.mvs1 during program execution, and this script contains the following line:

```
CONNECT mvs1 admin7 secret
```

To avoid storing the password 'secret' in readable form in the script file, the password is encrypted by invoking the USER-Access client and using the ENCRYPT alias:

```
User> encrypt secret NSC.JONES
User: The encrypted password is *26f17e2a4c9c65c56
```

Username NSC.JONES is specified because that is the local Stratus username under which the USER-Access job that uses the connect/login information will run. Using a local text editor, modify the input script >nsc>sicom>user>scripts.mvs1 to look like:

```
CONNECT mvs1 admin7 *26f17e2a4c9c65c56
```

Example 2: Using USER-Access to Generate the Input Script File

As you can see in the ENCRYPT alias definition, the global variable 'pw' is set to the encrypted password value. This value can be used to generate an input file containing the USER-Access CONNECT command to be later referenced by a USER-Access script. We can use the USER-Access OUTPUT command to generate the script file >nsc>sicom>user>scripts.mvs1 to connect to the host 'mvs1' as user 'admin7' with the password 'secret' (as shown in example #1):

```
User> encrypt secret NSC.JONES
User: The encrypted password is *26f17e2a4c9c65c56
User> set output prefix
User> output >nsc>sicom>user>scripts.mvs1
User> text CONNECT mvs1 admin7 {pw}
User> output
```

The resulting file >nsc>sicom>user>scripts.mvs1 will contain the following line:

```
CONNECT mvs1 admin7 *26f17e2a4c9c65c56
```

Related Topics

ENCRYPT Function

EXIT Command

Description

The EXIT command causes USER-Access to exit to the previous input level. Only when EXIT is issued from within an input script, the current input script is exited and control is returned to the previous input level (either an input script or interactive command line). EXIT differs from QUIT in that it returns control only to the previous input level. QUIT always returns control to the interactive input level (command line).

When issued from the interactive input level, EXIT causes USER-Access to terminate. If issued from an input script as part of a noninteractive USER-Access session (e.g., a batch job), the session terminates.

Format

Command	Qualifier	Parameter
EXit		[status]

Where:

EXit (required) the verb for this command. The minimum spelling is EX.

status (optional) the value to be returned by an input script, or USER-Access if used at the interactive level. The valid values for status are: Success, Warning, Error, and Fatal. The ON ERROR command can be used to capture an error resulting from an input script exiting with a status of Warning or Error. An exit status of Fatal causes USER-Access to immediately abort. If status is not specified, an exit status of Success is assumed.

Note: A FATAL exit from nested scripts while running under TSO may cause some files to be left open preventing them from being freed until logoff.

Examples

If you desire to leave the USER-Access session and return to your local system's command line interpreter, you would type:

```
User> exit
READY 12:00:00
```

At this point you should receive the prompt you normally receive from your system's command line interpreter (e.g., READY 12:00:00).

The following script exits with an Error status if any error occurs within the script, otherwise the script exits with a status of Success:

```
* Sample USER-Access script
*
on error exit error
send -crc sample.file
exit success
```

If the SEND command above results in an error, the script exits with a status of Error, otherwise it exits with a status of Success.

Related Topics

DISCONNECT Command
QUIT Command

GOTO Command

Description

The GOTO command instructs USER-Access to continue processing at the given label. The label must be the first item to appear on the USER-Access command line and must be succeeded immediately by a colon (":"). All labels are case sensitive and must appear somewhere within the current input level.

Format

Command	Qualifier	Parameter
GOTO		label

Where:

GOTO (required) is the verb for this command.

label (required) an alphanumeric string from one to twenty characters in length including underscores and other special characters.

Examples

The following is an example of a simple loop alias:

```
* Sample GOTO/Label script. Send 5 files
* having the names FILE1 thru FILE5
*
set variable count 1
LOOP:
send file{count}
set variable count {inc(count)}
{le(count, 5, "goto LOOP")}
*
text All files sent.
```

Related Topics

INPUT Command

ON Command

HELP Command

Description

The USER-Access help facility gives you on-line access to USER-Access topics including host specific qualifier information (both locally and remotely), formats and descriptions of all USER-Access commands and examples of how to use them. Some help text is actually retrieved from the remote host and therefore requires a remote connection. Typing HELP without a topic name will generate a top level help message followed by a list of all topics and commands for which help is available.

You can abbreviate any topic name on the HELP command line (including USER-Access commands) although the abbreviation must be unique to the topic name itself. The unique portion of the topic is represented in upper case letters as shown in the subtopics list.

Format

Command	Qualifier	Parameter
HELp	[-SEArch <i>string</i>]	[<i>topic</i> [<i>subtopic</i>]]

Where:

- HELp** (required) the verb for this command. The minimum spelling is HEL.
- SEArch** (optional) allows the user to search alternate paths for USER-Access help files. For more information, refer to “Stratus USER-Access SEARCH Keywords (SITE), (USER), and (NONE)” on page 92. The default is (SITE). The minimum spelling is -SEA.
- topic** (optional) name of a USER-Access topic or command in which you desire additional information.
- subtopic** (optional) a subtopic for which further help is available. These subtopics are displayed in the top level help information.

Examples

To get the highest level of help, you would just type:

```
User> help
```

This will provide you with a list of all topics in which help is available. From that list of topics you can begin getting help on more specific items of interest. For example, if you want help on the SEND command, you would type:

```
User> help send
```

Now, depending on the subtopics of SEND available, you might type:

```
User> help send example
```

which would display a sample SEND command.

The SEArch qualifier for the HELP command is used to define where the USER-Access help files exist:

```
User> show help search
User:
User: SEArch ..... (SITE)
```

User:

The default is "(SITE)", when "(SITE)" is not succeeded by a file name the partitioned data set member "userhelp.ua" is assumed. Suppose a user help file is defined, such as "alias.hlp", which contains the help text for the LOGIN alias. The alias LOGIN is defined as:

```
User> show alias login
User:
User: LOfin ..... ask -prompt "Host? " host
User:                ask -prompt "Username? " user
User:                ask -sec -prompt "Password? " pass
User:                !connect {host} {user} {pass}
User:
```

The HELP command does not find the text for the LOGIN alias unless the user help file, "myhelp", has been included on the HELP SEARCH path, in this case the help file "alias.hlp" is in the (SITE) location:

```
User> help login
User: Help is not available for 'login' (UA-4301)
```

Now add to the HELP SEARCH path "(SITE)alias.hlp":

```
User> set help search {search:help} (SITE)alias.hlp
User> show help search
User:
User: SEArch ..... (SITE) (SITE)alias.hlp
User:
User> help login
User:
User: FORMAT
User:
User:     LOgin
User:
User: DESCRIPTION
User:
User:     The LOgin alias is used to prompt user's for the
User:     necessary LOgin information.
User:
```

Now when the HELP command is used, two locations, (SITE) and (SITE)alias.hlp, are searched for help on the requested topic.

INPUT Command

Description

The INPUT command instructs USER-Access to take its commands from the specified input file on the local host. This file may contain any number of USER-Access commands. These commands can be structured in such a way that a sophisticated user could create predefined USER-Access procedures that can be used by beginning USER-Access users. These procedures can prompt users for input, give them instructions, and issue USER-Access commands for them.

Format

Command	Qualifier	Parameter
INput	<pre> [-CONTinue ON] OFF [-ECHO ON] OFF [-PROMpt string] [-PROMpt2 string] [-SEArch string] [-VERify ON] OFF </pre>	[source] [arguments]

Where:

- INput** (required) the verb for this command. The minimum spelling is IN.
- CONTinue** (optional) tells USER-Access how to respond to an error encountered when processing input files. This value should be set to either ON or OFF. ON tells USER-Access to continue processing even if an error is encountered while processing commands in the input file. OFF says to terminate processing of the input file if an error is encountered. The initial setting is OFF. The minimum spelling is -CONT.
- ECHO** (optional) tells USER-Access whether or not to echo input to the terminal as it reads input commands. Commands are echoed as they appear in the input file, before string substitution (and alias translation) is performed. This value should be set to either ON or OFF. The initial setting is OFF.
- PROMpt** (optional) the string used as the USER-Access command prompt. The initial setting is **user>**. The minimum spelling is -PROM.
- PROMPT2** (optional) a secondary USER-Access command prompt string used for command continuation. The initial setting is **more>>**.
- SEArch** (optional) search path used for default INPUT commands (the location of input files). SEARCH is only used if USER-Access cannot locate the source file specified on the command line. A SEARCH path is a space-separated list of Stratus file specifications. If SEARCH is defined, USER-Access will use it in an attempt to locate input files. Refer to “Stratus USER-Access SEARCH Keywords (SITE), (USER), and (NONE)” on page 92 for more information. The minimum spelling is -SEA.

- VERify** (optional) works like ECHO, but displays input commands after string substitution (and alias translation) has taken place. This value should be set to either ON or OFF. The initial setting is OFF. The minimum spelling is -VER.
- source** (optional) the file specification for the input file on the local host. USER-Access attempts to open this file before using the INPUT SEARCH path.
- arguments** (optional) zero or more arguments that are passed to the input file as positional parameters for parameter substitution.

Using the INPUT command causes a new INPUT environment to be established. The values for all the above qualifiers are initialized to the then current values. Changing a qualifier value changes the value for the duration of the input script only. Exiting the input script restores the INPUT qualifier values to the values existing before the INPUT command was issued.

Examples

To input USER-Access commands from a sequential data set named setup.si located in the current local TSOPREFIX, you would type:

```
User> input -echo on setup.si
USER-Access will then attempt to execute all of the commands in setup.si
before displaying the interactive User> prompt again. The -echo on switch
forces each line from the input file to be echoed to the screen as it is
processed.
```

If input file setup.si was set up to accept positional parameters you could also pass arguments on the INPUT command line as:

```
User> input setup.si HOST3 Smith
```

Suppose there exists a file by the name of “myalias.ua” in the user’s login directory. By default there is no INPUT SEARCH path defined. If a search path is defined as follows:

```
User> set inp search (USER)*.ua
```

The user need only specify the filename portion of the file specification if the files extension is “.ua”, and the file is located in the user’s default TSOPREFIX:

```
User> input myalias
```

The “input myalias” command uses the SEARCH path to find the USER-Access script file “myalias.ua” in the user’s login directory. Notice that the INPUT SEARCH qualifier is defined as “(USER)*.ua”, the “*” is replaced by the argument to the input command, in this case “myalias”.

The user can also execute an input file without preceding the name of the input file by the INPUT command. The order of processing a command is: check for alias, check for command, and check for an input file on the search path. Setting the search path as in the previous example:

```
User> set input search (USER)*.ua
```

The “myalias.ua” script file can be executed by issuing the command:

```
User> myalias
```

Command processing checks for an alias by the name “myalias”, then a USER-Access command by the name “myalias”, and, then uses the INPUT SEARCH qualifier to look for the file “myalias.ua” in the user’s login directory.

Related Topics

OUTPUT Command
SET ALIAS Command

LOCAL Command

Description

LOCAL executes a command on the local host and displays the results. The command can be a valid command for the local host's command line interpreter, an alias command defined using SET LOCAL ALIAS, or one of the predefined host independent commands (e.g. DIRECTORY, TYPE, STATUS, etc.). Whatever the case may be, the command specified must translate into a valid command on your local host or it will return an error to you. Any qualifiers passed to the local command must come before the *command* parameter.

If the *command* parameter is missing, you will enter an interactive local terminal mode. You will remain in local terminal mode until you leave it using the appropriate command for your local host (e.g. exit, logout, ...), at which time you will again see the User> prompt. All of your remote host connections will be intact.

Format

Command	Qualifier	Parameter
LOCa1	<pre>[-INTEractive ON] OFF } [-PREFix <i>string</i>] [-QUIet OFF] ON </pre>	[<i>command</i>]

Where:

- LOCa1** (required) the verb for this command. The minimum spelling is LOC.
- INTEractive** This optional qualifier can be set to either ON or OFF. Interactive ON forces an interactive mode of command execution, which is to say the command you invoke on the local host expects an interactive user (an editor for example). The default is OFF. The minimum spelling is -INT.
- PREFix** This optional qualifier is a prefix string that appears before each line of local command output. Its purpose is to "flag" output as coming from the local host versus USER-Access or remote output. You can define this qualifier to be null if no prefix is desired. The minimum spelling is -PREF. The default is the local host type followed by a colon and a space.
- QUIet** This optional qualifier forces local command output not to be displayed. The default is OFF. The minimum spelling is -QUI.
- command** (optional) a valid local host command or local alias command.

Informational Qualifiers

The following informational qualifiers are provided to give the user information about the local Stratus environment. With the exception of DIRectory, these qualifiers cannot be modified by the user.

- DIRectory** (string) the current working directory on the local Stratus host. This qualifier can be modified by using the SET LOCAL DIRECTORY command. This qualifier is the same as TSOPREFIX.
- HOSTCODE** (string) the native host character code (EBCDIC).
- HOSTTYPE** (string) Operating system type (Stratus).

PID (integer) Stratus Process ID.

PRODUCT (string) NESi Product number (H893V).

STATUS (string) exit status of the last LOCAL command.

VERSION (string) USER-Access version number.

Examples

To display all users currently logged into your local Stratus host you would execute the command from USER-Access as:

```
User> local list_users -full
Stratus: list_users -full
Stratus:   LOGIN TIME           CPU    PF   USER
Stratus: 11-11-30 09:44:39      7.46  124  Eran.Stratusoft (DRMS_APPL_01)
Stratus: 11-11-30 09:44:41      2.93   92  Eran.Stratusoft (DRMS_APPL_02)
Stratus: 11-11-30 09:44:43      2.89   92  Eran.Stratusoft (DRMS_APPL_03)
Stratus: 11-11-30 09:44:46      1.91  109  Eran.Stratusoft (DRMS_BACKUP_01)
Stratus: 11-11-30 09:59:36     15.17  541  Eran.Stratusoft (SPS/Alert_Manager)
Stratus: 11-11-30 09:44:39      4.23  121  Eran.Stratusoft SPS/Performance_Server)
Stratus: 11-11-30 09:44:39      0.37 7571  Eran.Stratusoft (SPS/Scheduler_PROD)
Stratus: 11-12-03 02:24:26     12.01 1041  Jon_Schmidt.SysAdmin (m2_tdi_as)
Stratus: 11-12-03 02:24:47      7.46 191672  Jon_Schmidt.SysAdmin (m2_tdi_oth)
Stratus: 11-12-03 02:24:02      4.22  434  Jon_Schmidt.SysAdmin (m2_tdi_svr)
Stratus: 11-12-05 13:34:54      0.03  667  Milt_Grimm.RDHD
Stratus: 11-11-23 14:35:11     19.59 4434  mqm.mqm (login)
Stratus: 11-12-05 13:23:25      0.01   96  Richard_Boylan.SysAdmin
User>
```

The prefix Stratus: tells you results are being returned from the local Stratus host. Since there is a host independent command (which really is a local alias) called 'WHO' that accomplishes the same task as list_users, you could have also typed the following to get the same results.

```
User> local who
```

On either of these commands you could force the prefix to be something else by changing the value of the PREFIX qualifier. For example:

```
User> local -prefix "HOSTA: " list_users
```

This command would cause the prefix HOSTA: to display before each line of output instead of the default "Stratus:".

To drop into the local system's command line interpreter without losing remote host connections, you can simply type:

```
User> local
ready 12:00:00
```

The local system's prompt should appear. At this point you can interact with your local system in the usual manner until you again want to return to your USER-Access session. To do this, just exit your system's command line interpreter in whatever way that is done for your host (e.g., exit, logout, <CTRL> <Z>, bye, etc.).

```
Ready 12:00:00   logout
User>
```

Related Topics

REMOTE Command
SET Command

ON Command

Description

The ON command allows users to catch any one of the exceptions:

ERRor	on USER-Access error
INTerrupt	on keyboard interrupt
LOCal_error	on LOCAL command error
REMote_error	on REMOTE command error

The ON command initializes the exception by specifying an action that should occur each time the exception takes place. To turn off the exception handler, issue the same command without an action.

ON ERRor

The ON ERRor exception establishes an alternative action to be taken when a USER-Access error occurs. Without an ON ERRor specified, USER-Access terminates all input levels (for nested input scripts), and begins processing at the interactive level. If INPut CONTInue is ON, USER-Access displays the error and continues processing the next command.

ON INTerrupt

The ON INTerrupt exception establishes an alternative action to be taken when a keyboard interrupt occurs. Without an ON INTerrupt specified, USER-Access terminates all input levels (for nested input scripts), and begins processing at the interactive level. If INPut CONTInue is ON, USER-Access terminates the current level and continues processing in the next level up.

ON LOCal_error

The ON LOCal_error exception establishes an alternative action to be taken when a LOCAL command error occurs. A local error occurs when a Stratus command returns any nonzero status.

ON REMote_error

The ON REMote_error exception establishes an alternative action to be taken when a REMote command error occurs. A remote error occurs when the remote command issued returns an unsuccessful status. The definition of REMote_error is dependent upon the remote host. Some hosts cannot detect command execution errors, in which case ON REMote_error becomes ineffective.

Format

Command	Qualifier	Parameter
ON		<i>exception</i> [<i>action</i>]

Where:

ON is the keyword for this command.

exception is any one of the following: ERRor, INTerrupt, LOCal_error, or REMote_error.

action is any USER-Access command or alias.

Examples

The following is a short USER-Access script that immediately exits should any error occur:

```
* Sample USER-Access script - this script
* exits should any EFT error occur
*
on error exit
set variable count 1
LOOP:
send file{count}
set variable count {inc(count)}
{le(count, 5, "goto LOOP")}
*
text All files sent.
```

The exception being handled is ERRor and the action to be taken, should an error occur, is the USER-Access command EXit.

The following is a short USER-Access script that immediately continues execution should any keyboard interrupt occur:

```
* Sample USER-Access script - this script
* continues should any keyboard interrupt
* occur.
*
on interrupt continue
set variable count 1
LOOP:
send file{count}
set variable count {inc(count)}
{le(count, 5, "goto LOOP")}
*
text All files sent.
```

The exception being handled is INTerrupt and the action to be taken should a keyboard interrupt occur is the USER-Access no-op command CONTinue.

The following script issues a "loc directory" command with a file name as an argument. If the command fails, implying the file does not exist, the ON LOCal_error action is to receive that file:

```
* Sample USER-Access script
*
* Setup ON LOCal_error, the action is to
* receive 'file{count}'.
*
on LOCal_error {}receive file{count}
set variable count 1
LOOP:
loc -quiet directory file{count}
set variable count {inc(count)}
{le(count, 5, "goto LOOP")}
*
exit
```

Related Topics

GOTO Command
INPUT Command

OUTPUT Command

Description

The OUTPUT command instructs USER-Access to capture all of its standard output to a file on the local host. OUTPUT gives the user the capability of saving the output of a local or remote command execution to a file on the local host.

Format

Command	Qualifier	Parameter
OUTput	[-COLumns <i>integer</i>] [-CREate APPend] BACKup DELete <u>NEW</u> REPlace [-FORMat <i>string</i>] [-HOLD ON] OFF [-LINES <i>integer</i>] [-PREFix <i>string</i>] [-QUIet OFF] ON [-TRUNcate OFF] ON	[<i>destination</i>]

Where:

- OUTput** This required parameter is the verb for this command. The minimum spelling is OUT.
- COLumns** This optional qualifier is the maximum number of columns per terminal page of output. This represents the maximum number of characters across a page or terminal screen. The minimum spelling is -COL. The default is 79.
- CREate** This optional qualifier describes how to create the output file on the local system. The valid values are APPend, BACKup, DELete, NEW, and REPlace. The default is NEW. The minimum spelling is -CRE.
- FORMat** This optional qualifier provides the format used to display all USER-Access messages. The 'msg()' string function is used to construct an appropriate format for USER-Access messages. The default is to display the message "text (facility-code)".
- HOLD** This optional qualifier suspends scrolling of the output from a command or input file. The number of lines that scroll by before the output is suspended is specified by the LINES qualifier. The default is OFF.
- LINES** This optional qualifier is the maximum number of lines per terminal page of output. The minimum spelling is -LIN. The default is 24.
- PREFix** This optional qualifier is the prefix string displayed before each line of USER-Access output. The default is "User: ". The minimum spelling is -PREF.

- Quiet** When this optional qualifier is set to ON, no USER-Access output is displayed to the user's terminal. If an output destination file exists, the output is still captured to the destination file. The minimum spelling is -QUI. The default is OFF.
- TRUNcate** This optional qualifier works in conjunction with the COLUMNS qualifier. If any USER-Access output lines are longer than the COLUMNS value, the lines are truncated when this qualifier is ON. The default is OFF. The minimum spelling is -TRUN.
- destination** This optional parameter is a local file specification that will receive the captured USER-Access output.

Informational Qualifiers

The following qualifiers are provided to give the user information about the OUTput command.

DESTination (string) the output destination file specification.

Examples

To begin capturing USER-Access output to a file on the local host named session, you would type:

```
User> output session
```

Now every line of USER-Access output that appears on the screen will also be sent to the output file until you close the file with another output command:

```
User> output
```

To tell USER-Access to hold the screen every time a full screen of output is displayed, type the following:

```
User> set output hold on
```

Related Topics

INPUT Command

QUIT Command

Description

The QUIT command causes USER-Access to return control to the interactive (command line) input level. When QUIT is issued from a nested input script, control is returned all the way back to the interactive input level (i.e., any input scripts nested before the one issuing the QUIT are also terminated). QUIT differs from EXIT in that QUIT always returns control to the interactive level whereas EXIT returns control back to the previous input level, whether it was interactive or another input script.

When issued from the interactive input level, QUIT causes USER-Access to terminate. If issued from an input script as part of a noninteractive USER-Access session (e.g., a batch job), the session terminates.

Format

Command	Qualifier	Parameter
Quit		[status]

Where:

Quit (required) the verb for this command. The minimum spelling is Q.

status (optional) the value to be returned by an input script, or USER-Access if used at the interactive level. The valid values for status are: Success, Warning, Error, and Fatal. The ON ERROR command can be used to capture an error resulting from an input script exiting with a status of Warning or Error. An exit status of Fatal causes USER-Access to immediately abort. If status is not specified, an exit status of Success is assumed.

Examples

If you desire to leave USER-Access and return to your local system's command line interpreter, you would type:

```
User> quit
READY 12:00:00
```

At this point you should receive the prompt you normally receive from your system's command line interpreter (e.g., READY).

If you desire to leave USER-Access and return an Error status to your local system, you would type:

```
User> quit error
READY 12:00:00
```

At this point you should receive the prompt you normally receive from your system's command line interpreter (e.g., READY).

Related Topics

DISCONNECT Command

RECEIVE Command

Description

The RECEIVE command receives the source file from the current active remote host and saves it as a destination file on the local host. If no path to the file is specified on either the source or destination file (i.e. if a file name is given without a directory or device specification), the default remote and local directories are used respectively. That is, the source file is assumed to exist in the remote default directory, and the newly received file will be created in the local default directory. If the destination parameter is not specified at all, a file by the same name as the source file name will be created in the local default directory.

The source file name may include the USER-Access wildcard characters * and ? as well as host specific wildcard characters where the two do not conflict. See the discussion on USER-Access wildcarding in “Source Wildcard Support for Stratus File Transfers” on page 43 for further details.

Format

Command	Qualifier	Parameter
RECEive	[-qualifiers]	source [destination]

Where:

- RECEive** (required) the verb for this command. The minimum spelling is REC.
- qualifiers** (optional) the qualifiers that apply to the RECEIVE command. Refer to “File Handling Under Stratus USER-Access” on page 39 for a description of the RECEIVE command qualifiers. See also the file handling section of the remote host manual for details on qualifiers supported.
- source** (required) the file specification for the file on the remote host that you intend to receive.
- destination** (optional) the file specification for the new file that is to be created on the local host.

Examples

Refer to “File Handling Under Stratus USER-Access” on page 39 for examples of using the RECEIVE command.

Related Topics

- LOCAL Command
- REMOTE Command
- SEND Command
- SET Command

REMOTE Command

Note: For details on how the REMOTE command operates, refer to the User’s Guide for the remote host. The description provided here gives a general overview of the command from the perspective of the local USER-Access initiator.

Description

REMOTE executes a command on the remote host and displays the results on the local system. The command can be a valid command for the remote host’s command line interpreter or an alias command defined using SET REMOTE ALIAS, or one of the predefined host independent commands (e.g., DIRECTORY, WHO, TYPE, etc.). Whatever the case may be, the command specified must translate into a valid command on the remote host or it will return an error.

There is no interactive mode for the REMOTE command. You must specify a remote host command containing all required parameters or an error will result. Also, since REMOTE doesn’t operate in an interactive mode, you will not be able to successfully execute commands or programs on the remote host that require an interactive user (e.g., one that prompts for input, such as graphics programs). You could however, use REMOTE to submit a job file on the remote host (or execute a script) that would accomplish much the same task given proper input data.

Format

Command	Qualifier	Parameter
REMOte	<code>[-PREFIX <i>string</i>]</code> <code>[-QUIet OFF]</code> <code> ON </code>	command

Where:

- REMOte** (required) the verb for this command. The minimum spelling is REM.
- PREFIX** (optional) a prefix string that appears before each line of remote command output. Its purpose is to “flag” output as coming from the remote host versus USER-Access or local output. You can define this qualifier to be null if no prefix is desired. The minimum spelling is -PREF.
- QUIet** (optional) ON forces remote command output not be displayed. The default is OFF. The minimum spelling is -QUI.
- command** (required) a valid remote host command or remote alias command.

Informational Qualifiers

The following qualifiers are provided to give the user information about the remote environment. Most are informational and cannot be modified by the user. -DIRECTORY, -PREFIX, and some host-specific qualifiers may be modified.

- BLOCKsize** (integer) NETEX negotiated block size.
- DIRectory** (string) the current working directory on the remote host. This qualifier can be modified using the SET REMOTE DIRECTORY command.
- HOST** (string) the remote host name.

- HOSTCODE** (string) the native host character code.
- HOSTTYPE** (string) Operating system type.
- PID** (integer) Process ID.
- PRODUCT** (string) NESi product number.
- SERVICE** (string) the NETEX service connected to.
- STATUS** (integer) exit status of the last REMote command.
- TRANSLATE** (string) the current translation in effect.
- USERNAME** (string) the user name.
- VERSION** (string) the USER-Access version number.

Examples

To display all users currently logged into the remote Stratus host you would execute the Stratus `list_users` command from USER-Access as:

```
TNPSTRAT> remote list_users -full
Stratus: list_users -full
Stratus:  LOGIN TIME          CPU      PF      USER
Stratus: 11-11-30 09:44:39    7.48    124    Eran.Stratusoft (DRMS_APPL_01)
Stratus: 11-11-30 09:44:41    2.94    92     Eran.Stratusoft (DRMS_APPL_02)
Stratus: 11-11-30 09:44:43    2.90    92     Eran.Stratusoft (DRMS_APPL_03)
Stratus: 11-11-30 09:44:46    1.91    109    Eran.Stratusoft (DRMS_BACKUP_01)
Stratus: 11-11-30 09:59:36    15.20   541    Eran.Stratusoft (SPS/Alert_Manager)
Stratus: 11-11-30 09:44:39    4.24    121    Eran.Stratusoft (SPS/Performance_Server)
Stratus: 11-11-30 09:44:39    0.37    7588   Eran.Stratusoft (SPS/Scheduler_PROD)
Stratus: 11-12-03 02:24:26    12.08   1041   Jon_Schmidt.SysAdmin (m2_tdi_as)
Stratus: 11-12-03 02:24:47    7.50   192746 Jon_Schmidt.SysAdmin (m2_tdi_oth)
Stratus: 11-12-03 02:24:02    4.24    434    Jon_Schmidt.SysAdmin (m2_tdi_svr)
Stratus: 11-12-05 13:34:54    0.03    730    Milt_Grimm.RDHD
Stratus: 11-12-05 14:08:36    0.00    237    Milt_Grimm.RDHD
Stratus: 11-12-05 14:11:18    0.00    94     * Milt_Grimm.RDHD (ua_command_process)
Stratus: 11-11-23 14:35:11    19.63   4434   mqm.mqm (login)
Stratus: 11-12-05 13:23:25    0.01    96     Richard_Boylan.SysAdmin
TNPSTRAT>
```

The prefix `Stratus` identifies results returned from the remote Stratus host. Since there is a predefined host independent command (which is really a remote alias) called `WHO` that accomplishes the same task as `list_users`, you could have also typed:

```
User> remote who
```

Assume the remote host is running UNIX. To display what it thinks is the current date, you would type:

```
User> remote date
```

The results would be whatever the current date is on the remote host.

Related Topics

LOCAL Command
SET Command

SEND Command

Description

The SEND command sends the source file from the local host to the current active remote host and saves it as a destination file. If no path to the file is specified on either the source or destination file (i.e. if a file name is given without a directory or device specification), the default local and remote directories are used respectively. That is, the source file is assumed to exist in the local default directory, and the new file is created in the remote default directory. If the destination parameter is not specified at all, a file by the same name as the source file name will be created in the remote default directory.

The source file name may include the USER-Access wildcard characters * and ? as well as host specific wildcard characters where the two do not conflict. See “Source Wildcard Support for Stratus File Transfers” on page 43 and “File Handling Under Stratus USER-Access” on page 39 for more details.

Format

Where:

Command	Qualifier	Parameter
SENd	[-qualifiers]	source [destination]

Where:

- SENd** (required) the verb for this command. The minimum spelling is SEN.
- qualifiers** (optional) the qualifiers that apply to the SEND command. Refer to the file handling section of the remote host for details about these qualifiers. For a remote Stratus host, refer to “File Handling Under Stratus USER-Access” on page 39.
- source** (required) the file specification for the file on the local host that you intend to send to the remote host.
- destination** (optional) the file specification for the new file that is to be created on the remote host.

Examples

Refer to “File Handling Under Stratus USER-Access” on page 39 for examples of the SEND command under Stratus. Refer to the file handling section of the appropriate remote host manual for examples under other operating systems.

Related Topics

- LOCAL Command
- RECEIVE Command
- REMOTE Command
- SET Command

SET Command

Description

This form of the SET command allows you to change the default value of a command qualifier. Most qualifiers are initially assigned reasonable defaults by USER-Access so novice users can issue commands without being concerned with qualifiers on the command line. Once a user becomes more familiar with USER-Access and wants to perform more complex tasks, he can set up commands with qualifier defaults of his own choosing. This is done with the SET command.

The value assigned to a command qualifier with SET becomes the new default for the command. The value of a qualifier is the remainder of the SET command line following the qualifier parameter. If a value is not specified on the SET command line, the qualifier is defined to be nothing (assigned a null value). The qualifier specified must be valid for the command. Use the SHOW QUALIFIER command to see which qualifiers and values are valid for a given command.

The value parameter is taken literally unless it is enclosed in double quotes ("value"). If the value is enclosed in double quotes, USER-Access expects any embedded quotes (that is, within the value) to be "escaped" using two double quotes together. This special processing allows value. to include leading and/or trailing spaces.

Note: The SET command is the only way to change the DIRECTORY qualifier for the LOCAL and REMOTE commands.

The commands SET VARIABLE and SET GLOBAL are detailed in later sections.

Format

Command	Qualifier	Parameter
SET		command qualifier [value]

Where:

- SET** (required) the verb for this command.
- command** (required) name of a USER-Access command that supports the use of command qualifiers (e.g., CONNECT, REMOTE, etc.)
- qualifier** (required with **command**) name of a valid qualifier for the command specified.
- value** (optional) the new default value you are assigning to this qualifier.

Examples

To change the default INPUT prompt string to >>, type:

```
User> set input prompt >>
>> show input prompt
User: PROMpt ..... >>
```

To change the current REMOTE default directory to DRA2:[TEMP] (assuming the remote host is a VAX running VMS), you would type:

```
>> set remote directory DRA2:[TEMP]
>> show remote directory
User: DIrectory ..... DRA2:[TEMP]
```

Related Topics

SHOW Command

SHOW QUALIFIER Command

SET ALIAS Command

Description

The SET ALIAS command allows you to define your own alias commands. By creating aliases you can tailor your own command language making things very simple for both beginning and advanced USER-Access users.

You can define three kinds of aliases: USER-Access, local, and remote. USER-Access alias definitions are made up of other USER-Access commands and are invoked simply by typing the alias command name in response to the USER-Access prompt. Local alias definitions are commands understood by your local host's command line interpreter and are invoked by typing LOCAL followed by the alias command name. Remote alias definitions are commands understood by the remote host's command line interpreter and are invoked by typing REMOTE followed by the alias command name. A remote connection is required to create a remote alias. The remote alias is only in effect for the remote connection active when the remote alias is defined.

The definition of the alias is the remainder of the line following the alias command name. You redefine an existing alias by using SET ALIAS to overwrite the previous definition with the new definition. If you do not supply the definition parameter, the alias becomes undefined. Use the SHOW ALIAS command to see which aliases are defined.

You can also create multicommand USER-Access aliases using the USER-Access escape character “!” at the end of the line. Multicommand aliases are discussed further in “Creating Multicommand USER-Access Aliases” on page 84.

Format

Command	Qualifier	Parameter
SET [LOCAL REMOTE] ALias		alias [definition] [!]

Where:

- SET** (required) the verb for this command.
- LOCAL or REMOTE** (optional) entering either LOCAL or REMOTE here tells USER-Access to create a local or remote alias command instead of a USER-Access alias. The minimum spellings are LOC and REM.
- ALias** (required) the subject for this command. The minimum spelling is AL.
- alias** (required) name of the new or existing alias in which you are attempting to define. If a portion of this is capitalized, that portion will be the minimum required spelling for the alias.
- definition** (optional) the string definition of the alias command you are defining.
- !** (optional) indicates the alias definition continues on the next line.

Host Dependencies

LOCAL and REMOTE aliases should translate to host dependent commands on the local or remote host respectively.

Examples

To define a USER-Access alias called `hc` to be the USER-Access command `HELP COMMANDS`, you would type:

```
User> set alias hc help commands
```

Now, to display the USER-Access command summary, you would just type

```
User> hc
```

which is equivalent to typing:

```
User> help commands
```

If your local host supports the command `'whoami'`, which displays your local username, you can create a local alias command that displays your local username:

```
User> set local alias ? whoami
```

To invoke your new alias, all you need to do is type:

```
User> local ?
```

which is equivalent to typing:

```
User> local whoami
```

To create a multicommand USER-Access alias named `STatus` that displays the current `LOCAL` and `REMOTE` qualifier defaults, you would type:

```
User> set alias STatus show local !  
More>> show remote
```

To execute your new alias command you would type:

```
User> status
```

which would result in the list of `LOCAL` qualifier defaults followed by the list of `REMOTE` qualifier defaults.

Related Topics

LOCAL Command
SHOW ALIAS Command
REMOTE Command

SET GLOBAL Command

Description

The SET GLOBAL command assigns a value to a global variable name. The scope of a global variable within USER-Access scripts is not limited to the input level on which it was defined (global in scope). Refer to the SET VARIABLE command if the scope of the variable needs to be limited to the current input level. If the value parameter does not exist the variable becomes undefined.

The value parameter is taken literally unless it is enclosed in double quotes (“value”). If the value is enclosed in double quotes, USER-Access expects any embedded quotes (that is, within the value) to be “escaped” using two double quotes together. This special processing allows value to include leading and/or trailing spaces.

Global variables may be referenced by placing the variable name between braces, in one of two ways. Assume a global variable by the name of *num* exists. The global variable *num* can be referenced by either “{num}” if a local variable by the same name does not exist, or by “{num:global}” to explicitly request the global definition.

If a global variable is referenced and does not exist the variable is replaced by a NULL string. If the global variable *num* is not defined the string “{num:global}” is equivalent to “”. Also the number of globals that can be defined at one time is limited.

Format

Command	Qualifier	Parameter
SET GLOBAL		name [value]

Where:

- SET** (required) the verb for this command.
- GLOBAL** (required) the keyword for this command. The minimum spelling is GLOB.
- name** (required) global variable name. USER-Access variable names must be alphanumeric and no longer than 20 characters.
- value** (optional) variable value. The value assigned to the variable name is the remainder of the line starting with the first nonblank character.

Example

The following sample script assigns a numeric value to the global “num”:

```
User> set global num 1
User> text The global num = {num:global}
User: The global num = 1
User> show global num
User:
User: NUM ..... 1
User:
```

Related Topics

SHOW GLOBAL Command
SET VARIABLE Command
SHOW VARIABLE Command

SET HOST Command

Description

The SET HOST command allows you to select a host as the current “active” host. SET HOST is typically used following a DISCONNECT command in order to activate some other remote connection that is currently idle.

You can either specify a host name or host number on the command line. The host number is obtained from the SHOW HOSTS command.

Format

Command	Qualifier	Parameter
SET Host		hostname

Where:

SET (required) the verb for this command.

Host (required) the subject for this command. The minimum spelling is HO.

hostname (required) host name or host number of a previously established connection.

Examples

Assume the following connections have already been made:

```
User> connect vaxa scott john
User> connect sun01 meyers ed
```

Typing the SHOW HOSTS command then would result in:

```
User> show hosts
User:          (1) Host=vaxa      User=scott
User: active -->(2) Host=sun01   User=meyers
```

We can use SET HOST here to “reactivate” the first connection (to host vaxa) in one of two ways:

```
User> set host 1
```

or

```
User> set host vaxa
```

The result would be the same:

```
User> show hosts
User: active --> (1) Host=vaxa      User=scott
User:          (2) Host=sun01   User=meyers
```

Now any USER-Access command that requires a remote connection (e.g., SEND and RECEIVE), would communicate with host *vaxa*.

Related Topics

CONNECT Command

DISCONNECT Command
SHOW HOSTS Command

SET VARIABLE Command

Description

The SET VARIABLE command assigns a value to a local variable name. The scope of the variable is limited to the current input level (local in scope). If the variable needs to be visible outside the current input level refer to the SET GLOBAL command. If the value parameter does not exist the variable becomes undefined. A variable may also be set using the ASK command.

The value parameter is taken literally unless it is enclosed in double quotes (“value”). If the value is enclosed in double quotes, USER-Access expects any embedded quotes (that is, within the value) to be “escaped” using two double quotes together. This special processing allows value to include leading and/or trailing spaces.

Variables are referenced by placing the variable name between braces. Assume a variable by the name of *num* exists. The variable num can be referenced by “{num}”.

If a variable is referenced and does not exist the variable is replaced by a NULL string. If the variable num is not defined, and the global variable {num:global} is not defined, the string “{num}” is equivalent to “”.

Format

Command	Qualifier	Parameter
SET VArIable		name [value]

Where:

- SET** (required) the verb for this command.
- VArIable** (required) the keyword for this command. The minimum spelling is VAR.
- name** (required) variable name. USER-Access variable names must be alphanumeric and no longer than 20 characters.
- value** (optional) variable value. The value assigned to the variable name is the remainder of the line starting with the first nonblank character.

Examples

The following sample script assigns a numeric value to the variable “num”:

```
User> set variable num 1
User> text The variable num = {num}
User: The variable num = 1
User> show variable num
User:
User: NUM ..... 1
User:
```

Related Topics

[ASK Command](#)

[SHOW VARIABLE Command](#)

[SET GLOBAL Command](#)

[SHOW GLOBAL Command](#)

SHOW Command

Description

SHOW allows you to display the current default for any command qualifier. You can display the current defaults for all qualifiers of a particular command by leaving the qualifier parameter off the command line.

Note that some commands require a remote connection to show a complete list of qualifiers with the SHOW command. These commands, for example RECEIVE, gather information from the remote host and may display only a partial list without a connection. They also may just return an error message.

The commands SHOW GLOBAL and SHOW VARIABLE are described later in this document.

Format

Command	Qualifier	Parameter
SHow		command [qualifier]

Where:

- SHow** (required) the verb for this command. The minimum spelling is SH.
- command** (required) the name of a USER-Access command that supports the use of command qualifiers (e.g., CONNECT, LOCAL, etc.).
- qualifier** (optional) name of a valid qualifier for the given command.

Examples

To display the current default values for all INPUT qualifiers, you would type:

```
User> show input
User: CONTInue ..... on
User: ECHO ..... off
User: PROMpt ..... User>
User: PROMPT2 ..... More>>
User: SEARCH .....
User: VERify ..... off
```

If you are only interested in the default value of qualifier PROMPT2, you would type:

```
User> show input prompt2
User: PROMPT2 ..... More>>
```

Related Topics

- SET Command
- ASK Command

SHOW ALIAS Command

Description

SHOW ALIAS allows you to display the alias command definitions for any aliases previously defined. To display local host aliases, you need to type show local alias. To display remote host aliases, you need to type *show local alias*. If you do not specify either local or remote, USER-Access aliases will be displayed.

Format

Command	Qualifier	Parameter
SHoW [LOCaL] ALias		[alias]

Where:

- SHoW** (required) the verb for this command. The minimum spelling is SH.
- ALias** (required) the subject for this command. The minimum spelling is AL.
- LOCaL or REMote** (optional) entering either LOCAL or REMOTE here tells USER-Access to display a local or remote alias definition rather than a USER-Access alias definition. The minimum spellings are LOC and REM.
- alias** (optional) the name of a previously defined alias command.

Examples

Suppose the remote alias 'WHO' is defined by the remote command "whoami". To display the definition for the remote alias, you would type:

```
User> show remote alias who
User: WHO ..... whoami
```

To see all USER-Access alias command definitions you would type:

```
User> show alias
User: VAX ..... connect vax meyers ed
User: SL ..... show local
User: ? ..... remote who
```

Related Topics

SET ALIAS Command

SHOW GLOBAL Command

Description

The SHOW GLOBAL command displays the currently assigned value of the global variable specified. If no name is specified, then all global variables and their values are displayed.

Format

Command	Qualifier	Parameter
SHow GLOBal		[name]

Where:

SHow (required) the verb for this command. The minimum spelling is SH.

GLOBal (required) the keyword for this command. The minimum spelling is GLOB.

name (optional) global variable name. USER-Access variable names must be alphanumeric and no longer than 20 characters.

Examples

The following commands define two variables, “first” and “last”:

```
User> set global first john
User> set global last doe
```

The following SHOW GLOBAL command displays the values of the global variables “first” and “last”:

```
User> show global first
User:
User: FIRST ..... john
User:
User> show global
User:
User: FIRST ..... john
User: LAST ..... doe
User:
```

Related Topics

SET GLOBAL Command
SET VARIABLE Command
SHOW VARIABLE Command

SHOW HOSTS Command

Description

SHOW HOSTS displays all remote hosts currently connected to the local host in this USER-Access session. Connections are established with the CONNECT command. The list displayed by SHOW HOSTS includes a host connection number, the host name, logged in user name, and which host (if any) is the current “active” remote host. The host number that is displayed can be used as input to the SET HOST command.

Format

Command	Qualifier	Parameter
SHow HOsts		

Where:

SHow (required) the verb for this command. The minimum spelling is SH.

HOst (required) the subject for this command. The minimum spelling is HO.

Examples

Assume the following connections have already been made:

```
User> connect vaxa scott john
User> connect sun01 meyers ed
```

Typing the SHOW HOSTS command then would result in:

```
User> show hosts
User:
User:          (1) Host=vaxa          User=scott
User: active --> (2) Host=sun01       User=meyers
User:
```

Related Topics

CONNECT Command
DISCONNECT Command
SET HOST Command

SHOW QUALIFIER Command

Description

SHOW QUALIFIER lists all of the valid qualifiers for the specified USER-Access command and gives a brief definition of each. Note that some commands require a remote connection to show a complete list of qualifiers with the SHOW QUALIFIER command. These commands, for example RECEIVE, gather information from the remote host and may display only a partial list without a connection. They also may just return an error message.

Format

Command	Qualifier	Parameter
SHow QUALifier		command

Where:

- SHow** (required) the verb for this command. The minimum spelling is SH.
- QUALifier** (required) the subject for this command. The minimum spelling is QUA.
- command** (required) the name of a USER-Access command that supports the use of command qualifiers (e.g., SEND, LOCAL, etc.).

Examples

To list all of the valid qualifiers for the INPUT command you would type:

```
User> show qualifier input
User: CONTinue .. continue on error (on/off)
User: ECHO ..... echo input to terminal (on/off)
User: PROMpt .... prompt string for USER input
User: PROMPT2 ... secondary prompt for input continuation
User: SEARCH .... search path for default INPUT commands
User: VERify .... verify string substitution (on/off)
```

Related Topics

SHOW Command
SET Command

SHOW VARIABLE Command

Description

The SHOW VARIABLE command displays the currently assigned value of the variable name specified. If no name is specified then all variables and their values are displayed.

Format

Command	Qualifier	Parameter
SHow VARIable		name

Where:

SHow (required) the verb for this command. The minimum spelling is SH.

VARIable (required) the keyword for this command. The minimum spelling is VAR.

name (optional) variable name. USER-Access variable names must be alphanumeric and no longer than 20 characters.

Examples

The following commands define two variables, “first” and “last”:

```
User> set variable first john
User> set variable last doe
```

The following SHOW VARIABLE commands display the values of the variables “first” and “last”:

```
User> show variable first
User:
User: FIRST ..... john
User:
User> show variable
User:
User: FIRST ..... john
User: LAST ..... doe
User:
```

Related Topics

SET VARIABLE Command
SET GLOBAL Command
SHOW GLOBAL Command

TEXT Command

Description

Command TEXT writes a string of text to the user's terminal and/or output file. TEXT is usually used within USER-Access input scripts or aliases for interaction with a user. The string parameter may contain string substitution syntax.

Format

Command	Qualifier	Parameter
TEXT		[string]

Where:

TEXT (required) the verb for this command. The minimum spelling is TEX.

string (optional) a string of text to be written out to the terminal or output file. The string may contain string substitution syntax such as string functions and references to variables. The default is to write out an empty string.

Examples

To display a simple line of text on the screen:

```
User> text this is a line of text to echo
User: this is a line of text to echo
```

You can use TEXT with a string containing string substitution syntax. For example, assuming you have a USER-Access variable called NAME defined to be Paul, you can display its value within a text string as:

```
User> text Is your name {NAME}?
User: Is your name Paul?
```

Related Topics

ASK Command

INPUT Command

TRANSLATE Command

Description

The TRANSLATE command is used to specify and display USER-Access code conversion tables, to enable and disable USER-Access code conversion, and to indicate whether or not USER-Access code conversion is currently enabled.

The TRANSLATE command followed by an action lets the user display and control the use of the USER-Access translation tables. By default NETEX code conversion is done unless the user explicitly turns on USER-Access translation with the TRANSLATE ON command. USER-Access translation can be turned off (i.e., NETEX code conversion is enabled) with the command TRANSLATE OFF. The USER-Access translation tables can be displayed using the TRANSLATE command followed by the actions DISPLAY or FULL. The RESET action reinitializes the USER-Access translation tables.

The TRANSLATE command is also used to tailor the default USER-Access translation tables. The initial USER-Access translation tables are identical to the NETEX translation tables until modifications are made to the USER-Access tables by using the TRANSLATE command followed by a native character code and a remote character code.

The initial invocation of TRANSLATE (that is, any TRANSLATE command) loads the USER-Access tables with the NETEX defaults and then uses the SEARCH qualifier to make changes to the USER-Access translation tables. The TRANSLATE RESET command reinitializes the USER-Access tables with the NETEX defaults and uses the SEARCH qualifier again, to make changes to the USER-Access tables.

When USER-Access translation is enabled with the command TRANSLATE ON, the USER-Access tables are used to convert native character codes into remote character codes, and remote character code into native character codes.

The qualifiers IN_only and OUT_only allow the tables to be modified in one direction. By default, modifications are made to both the incoming and outgoing tables. For example, if the user requests that an incoming EBCDIC cent sign be converted to an ASCII7 left bracket, then an outgoing ASCII7 left bracket is converted into an EBCDIC cent sign. By specifying either IN_only or OUT_only, only the incoming or outgoing table is modified.

The TRANSLATE command without any arguments displays the status of USER-Access translation, either enabled or disabled.

The TRANSLATE command does not allow the user to change the following native characters since these characters are required for USER-Access protocol:

- upper case alphabetic (A-Z)
- digits (0-9)
- space
- equal sign (=)
- null

Format

Command	Qualifier	Parameter
TRanslate	[-IN_only OFF] ON [-OUT_only OFF] ON [-SEArch string]	[action] [native] [remote] [comment]

Where:

- TRanslate** (required) is the keyword for this command. The minimum spelling is TR.
- IN_only** (optional) is used to modify only the incoming USER-Access code conversion table. The default is OFF. The minimum spelling is -IN.
- OUT_only** (optional) is used to modify only the outgoing USER-Access code conversion table. The default is OFF. The minimum spelling is -OUT.
- SEArch** (optional) the TRANSLATE SEARCH path is used to find default translation tables for the different HOSTCODES. On a Stratus system the default search path is "(SITE)". Each time a translate command is issued USER-Access looks in the SITE partitioned data set for the member "{HOSTCODE:remote}". For additional information, refer to "Stratus USER-Access SEARCH Keywords (SITE), (USER), and (NONE)" on page 92. The minimum spelling is -SEA.
- action** (optional) describes the action TRANSLATE takes. The action can be one of the following:
- DIsplay** display differences from NETEX tables
 - FUII** display entire translate table
 - OFF** disable USER-Access translation
 - ON** enable USER-Access translation
 - REset** reset table and process search path
- native** (optional) the native character code (in octal, decimal, or hexadecimal format).
- remote** (optional) the remote character code (in octal, decimal, or hexadecimal format).
- comment** (optional) descriptive comment.

Examples

Example #1:

Suppose the native character set is EBCDIC and the remote host has a HOSTCODE value of ASCII7. A translation table is setup to translate the EBCDIC cent sign (0x4A) and solid bar (0x4F) (which are invalid ASCII7 characters) to the ASCII7 left bracket (0x5B) and right bracket (0x5D), the following USER-Access commands would update the default translation tables:

```
User> translate 0x5B 0x4A (left bracket <--> cent sign)
User> translate 0x5D 0x4F (right bracket <--> solid bar)
```

USER-Access translation is enabled with the following command:

```
User> translate on
```

This translation effects both directions. The EBCDIC cent sign (0x4A) is converted into a ASCII7 left bracket (0x5B) on its way out, and the ASCII7 left bracket (0x5B) is converted into and EBCDIC cent sign (0x4A) on its way in.

Example #2:

Suppose the native character set is EBCDIC and the remote host has a HOSTCODE value of ASCII7:

```
User> connect VAX user pw -quiet
```

The current TRANSLATE SEARCH path is:

```
User> show translate search
User:
User: SEArch . . . . . (SITE)
User:
```

USER-Access translation is turned on with the following command:

```
User> translate on
```

At this point USER-Access uses the SEArch path to look for a USER-Access script file using the remote HOSTCODE, in this case ASCII7. For a Stratus system the SEArch path tells USER-Access to look for the partitioned data set member “ASCII7” in the (SITE) location. The partitioned data set member “ASCII7” could contain the lines from the above example:

```
* Sample ASCII7 translation table for
* an EBCDIC host
*
translate 0x5B 0x4A (left bracket <--> cent sign)
translate 0x5D 0x4F (right bracket <--> solid bar)
```

Now every time a connection is active to an ASCII7 host and the USER-Access translation tables are initialized, the table is automatically loaded.

The translate table can be displayed with the command:

```
User> translate full
```

Example #3:

The TRANSLATE search path allows the user to select optional files for input when the TRANSLATE command is issued. Suppose instead of loading the default tables for the active host, “{HOSTCODE:REMOTE}”, the user would rather load a table from the user’s default TSOPREFIX. The current search path is:

```
User> show translate search
User:
User: SEArch . . . . . (SITE)
User:
```

The search path can be changed with the command:

```
User> set translate search (USER)translate.ua
User> show translate search
User:
User: SEArch . . . . . (USER)translate.ua
User:
```


The search path for the TRANSLATE command now implies, look for the Sequential Data Set “translate.ua” in the default TSOPREFIX. The following command turns on translation and initializes the translate tables by reading the sequential data set “TRANSLATE.UA” in the user’s default TSOPREFIX:

```
User> translate on
```

Example #4

Suppose a Swedish Stratus host (ASCII7) is connected to a PC-DOS host (ASCII8). The Swedish national characters represented by the ASCII7 characters [|\{}] can be converted to the actual PC-DOS ASCII8 representations. The Stratus default HOSTCODE file ascii8.ua (in the (SITE) directory) contains the following commands:

```
* USER-Access TRANSLATE code conversion defaults
*
*       Native: ASCII7
*       Remote: ASCII8
*
TRANS 0x50 0x8F      & to upper case A with ring
TRANS 0x5B 0x8E      ¢ to upper case A with umlaut
TRANS 0x5C 0x99      \ to upper case O with umlaut
TRANS 0x7D 0x86      } to lower case a with ring
TRANS 0x7B 0x84      { to lower case a with umlaut
TRANS 0x7C 0x94      | to lower case o with umlaut
TEXT Loaded the SWEDISH translation tables.
```

Once connected to PC-DOS the USER-Access translation tables are initialized with the first invocation of TRANSLATE:

```
User> translate
User:
User: USER-Access translation is currently disabled.
User:
User: loaded the SWEDISH translation tables.
```

Translation is turned on with the following command:

```
User> translate on
```

All file transfers and remote command executions that follow, perform translations as defined by the USER-Access tables.

Appendix A. USER-Access Error Messages for Stratus

This appendix is intended to give users more information about USER-Access messages that may be seen during a session. All of the USER-Access messages have been listed along with details describing them. Many of the messages are self-explanatory (e.g., “Invalid command”) and require no further discussion. At the end of this message table is a list and description of the messages that require further explanation.

It should be noted that the USER-Access messaging scheme is designed to generate various levels of messages which is why a single erroneous condition may result in two or more messages. Each level of message that is displayed (from first to last), is designed to be slightly more specific than the message preceding it. All of the messages that are displayed should be considered when attempting to diagnose an error condition.

Below is a breakdown of each column of the message table along with a description of the entries that may appear under it.

FAC	The FACility or subsystem name that generated the message. This will be UA (USER-Access host independent message), UA893 (USER-Access H893V Stratus/OpenVOS host dependent message), SI (USER-Access Service Initiator) or SI893 (USER-Access H893V Stratus/OpenVOS Service Initiator). If a message contains a UAxx3, SIxx3, or some facility other than those listed above, this message must be looked up in the appropriate USER-Access manual for that host (e.g., Hxx3 USER-Access).
Code	The unique error or message code.
SEV	A single character severity level indicator. The possible values are S (Success), I (Information), W (Warning), E (Error), or F (Fatal).
Comment	<p>A comma-separated list of zero to five special characters giving more details about the message. If no Comment characters are given, the message (along with accompanying messages), is intended to be self-explanatory. The possible Comment characters are:</p> <ul style="list-style-type: none">A An additional description of the message is given at the end of the message table.D Diagnostic or Internal error. These errors are very unlikely to occur and may indicate a more severe problem is at hand or that some unexpected internal condition occurred. The user should refer to accompanying messages, if displayed, for a further explanation of the problem. These messages should be logged and reported to the system administrator.H Host specific messages will accompany these messages. The host specific messages should provide additional information as to the cause of the condition.N Network related message. The user should refer to the appropriate NetEx or system documentation for a further explanation of the problem. These messages should be logged and reported to the system administrator unless it is obvious that the network condition is temporary and for a known reason. These messages could possibly be a sign of a network interruption of some kind.R Retriable error condition. The command used to generate this error can be retried at some later time without fear of a fatal condition occurring.
Text	The message text. In the following message texts, SSS indicates a string value is present in the actual message; NNN, a numeric value; and C, a single character.

Table 3. Error Messages

FAC	Code	SEV	Comment	Text
UA	201	E		Invalid positional parameter 'SSS'
UA	202	E		Invalid command line switch 'SSS'
UA	203	E		Missing value for switch 'SSS'
UA	302	E	A	Overflow of NNN byte environment buffer
UA	303	E	A	Failed to add 'SSS = SSS...'
UA	304	E		Environment concatenate failure
UA	413	W		Invalid numeric parameter for function 'SSS'
UA	501	E	A,D,N	Protocol error - expected [SS] - got [SS]
UA	503	E	N	Failed to receive INFORMATIVE messages
UA	701	E		Protocol buffer size NNN is less than minimum NNN
UA	704	E	D	Failed to get protocol keyword value for SSS
UA	706	E		Protocol record (NNN) is larger than buffer size (NNN)
UA	712	E	D	Failed to read STDIN
UA	713	E	D,N	Invalid protocol Record flag [S]
UA	714	E	D	Protocol buffer (NNN) is too small for record (NNN)
UA	717	E	D,N	Invalid protocol Block/Record flag [S]
UA	801	E	D	Missing HOST name
UA	802	E	D	Missing SERVICE name
UA	803	E	R	Service 'SSS' is not offered on host 'SSS'
UA	804	E		Host 'SSS' does not exist in configuration
UA	805	E	N,R	Error connecting to service 'SSS' on host 'SSS'
UA	806	E	N,R	CONFIRM timed out after NNN seconds
UA	807	E	N,R	Error on CONFIRM from service 'SSS'
UA	808	E	N,R	Error on WRITE to service 'SSS'
UA	809	E	N,R	READ timed out after NNN seconds
UA	810	E	N,R	Error on READ from service 'SSS'
UA	811	E	D,N,R	Bad data length NNN on READ
UA	812	E	D,R	Bad DATAMODE 'NNN' on exchange
UA	813	E	D,N	Failed to open protocol connection
UA	814	E	D,N	Failed to send CONNECT environment
UA	815	E	D,N	Failed to receive CONNECT response

Table 3. Error Messages

FAC	Code	SEV	Comment	Text
UA	816	E	D,N	Failed to close protocol connection
UA	817	E	D	Missing SERVICE name
UA	818	E		Offer of service 'SSS' exceeded NNN second timeout
UA	819	E	N,R	Failed to OFFER service 'SSS'
UA	820	E	N,R	CONFIRM of offer failed
UA	821	E	N,R	READ timed out after NNN seconds
UA	822	E	N,R	Error on READ of datamode
UA	823	E	D,N,R	Bad data length NNN on READ
UA	824	E	D,R	Bad DATAMODE 'NNN' on exchange
UA	825	E	N,R	Error on WRITE of datamode
UA	826	E	D,N	Failed to open protocol connection
UA	827	E	D,N	Failed to receive CONNECT environment
UA	828	E	R	Invalid process identifier (PID) on reconnect
UA	829	E	D,N	Failed to send CONNECT response
UA	830	E	D,N	Failed to close protocol connection
UA	831	E		NETEX blocksize negotiation failed (NNN)
UA	901	E		Invalid password encryption type 'S'
UA	4003	W		Alias 'SSS' is not defined
UA	4102	E		BLOCKSIZE of NNN is out of range (NNN-NNN)
UA	4103	W		Cannot have more than NNN active connections
UA	4104	E	D	Missing connect SERVICE
UA	4105	E		Failed to connect service 'SSS' on host 'SSS'
UA	4106	I	A	The requested blocksize NNN was reduced to NNN
UA	4107	E	D	Failed to open CLIENT protocol connection
UA	4108	E	D,N	Failed to receive connect information
UA	4109	I	A	There were NNN CONNECT records ignored
UA	4110	E	N	Failed to DISCONNECT
UA	4112	W		Remote host required for remote help request
UA	4113	E	N	Failed to request remote RECEIVE
UA	4114	E	N	Failed to get SEND acknowledge
UA	4116	E	N	Failed to request remote SEND

Table 3. Error Messages

FAC	Code	SEV	Comment	Text
UA	4117	E	N	Failed to get RECEIVE acknowledge
UA	4118	E	N	Failed to send SOURCE/DEST environments
UA	4121	W		Missing remote command
UA	4123	E	N	Failed to receive an ABORT acknowledge
UA	4124	E		Remote SSS failed
UA	4126	W	D	Command 'SSS' is not implemented
UA	4127	E	A,D	The MESSAGE stack is empty
UA	4128	E	N	Failed to send error message to remote server
UA	4129	E	N	Failed to communicate with remote server
UA	4131	E	A,N	Failed to establish secondary NETEX connection
UA	4132	E	A	Restricted command in server startup file
UA	4201	E		Missing SSS parameter
UA	4202	E		Invalid SSS parameter 'SSS'
UA	4203	W		There is no active remote host
UA	4301	E		Help is not available for 'SSS'
UA	4302	E		Help line is longer than NNN characters
UA	4501	E	A	Nested (or recursive) input/alias limit of NNN exceeded
UA	4502	F	D	Can't open STDOUT
UA	4503	E		Output PREFIX (NNN) exceeds COLUMNS (NNN)
UA	4504	E	A	Bad output FORMAT definition- reset to default
UA	4505	E	A	Input request (NNN byte maximum) failed
UA	4601	W	A	Variable 'SSS' contains invalid characters
UA	4602	W		Variable 'SSS' is longer than NNN characters
UA	4603	W		Qualifier SSS cannot be modified
UA	4604	W		A value is required for qualifier SSS
UA	4605	W		Invalid SSS numeric value 'SSS'
UA	4606	W		SSS value SSS is out of range (SSS)
UA	4607	W		Invalid SSS Boolean value 'SSS'
UA	4610	W		Invalid SSS option 'SSS'
UA	4701	W	A	Recursive alias 'SSS'
UA	4702	W		There is no active remote host

Table 3. Error Messages

FAC	Code	SEV	Comment	Text
UA	4703	W		Invalid SSS qualifier 'SSS'
UA	4704	W	A	Use SET LOCAL/REMOTE to modify SSS qualifier 'SSS'
UA	4705	W		Missing value for SSS qualifier 'SSS'
UA	4706	W		Too many parameters for SSS
UA	4707	W		SSS requires additional parameters
UA	4708	W		Invalid command 'SSS'
UA	4709	W	A	Command token is greater than NNN characters
UA	4802	E	D	Missing MAXRECORD specifications
UA	4803	W	D	MAXRECORD (NNN) greater than maximum (NNN)
UA	4804	E	A	MAXRECORD (NNN) + NNN) too large for BLOCKSIZE (NNN)
UA	4807	E	D	Bad header flag (NNN) in record RECEIVE
UA	4808	E	D,N	Bad header flag (NNN) in record RECEIVE
UA	4809	E	A,D,N	Sequence error (NNN vs. NNN) in record RECEIVE
UA	4812	E	D	Missing MIN_BYTE_COUNT specifier
UA	4901	E		Failure during SSS mode receive
UA	4903	E		Failure during RECEIVE file setup
UA	4906	E		Receiving process exited with data still pending
UA	5001	E		Failure during SSS mode send
UA	5003	E		Failure during SEND file setup
UA	5050	E		Invalid transfer mode for AUTO data generation
UA	5107	F	N	Failed to receive next CLIENT request
UA	5109	E		Server SSS failed
UA	5111	E		Keyboard interrupt
UA	5112	E		Keyboard interrupt
UA	5115	E	D	Missing VALIDATE qualifier
UA	5116	E	D	Invalid VALIDATE qualifier 'SSS'
UA	5118	E	D,N	Invalid code table length NNN
UA	5119	E	D	Unsupported request code 'S' – no action taken
UA	5120	E	A,D	The MESSAGE stack is empty
UA	5121	F	N	Failed to send error message to CLIENT

Table 3. Error Messages

FAC	Code	SEV	Comment	Text
UA	5122	F	N	Failed to offer service 'SSS'
UA	5123	F	D	Failed to open SERVER protocol connection
UA	5124	F		Missing or invalid password
UA	5201	W		Missing HOST specifier
UA	5202	W		Invalid HOST index 'NNN'
UA	5203	W		Host 'SSS' is not active
UA	5204	W		Missing SSS qualifier
UA	5205	W		Invalid SSS qualifier 'SSS'
UA	5206	W	A	Use SET LOCAL/REMOTE to modify SSS qualifier 'SSS'
UA	5207	W		Invalid SSS qualifier 'SSS'
UA	5208	W		There are no remote host connections
UA	5301	E	A	Invalid allocation parameters (NNN + NNN > NNN)
UA	5304	E	A	Invalid ARCHIVE file format [SSS]
UA	5305	E	A	Invalid ARCHIVE block length (NNN)
UA	5306	E	A	Incomplete ARCHIVE file - missing end-of- file.
UA	5307	E		Incomplete ARCHIVE write (NNN of NNN bytes written)
UA	5401	W	A	More than NNN levels of nested strings
UA	5402	W		Unmatched string delimiter 'S'
UA	5403	W		Invalid string format
UA	5404	W		Unmatched string delimiter 'S'
UA	5405	W		No closing quote on string literal
UA	5406	W	A	Empty string substitution
UA	5407	W		String variable 'SSS:SSS' is invalid
UA	5408	W		Invalid SSS qualifier 'SSS'
UA	5410	W		Invalid string function 'SSS'
UA	5411	W		Invalid parameter for function 'SSS'
UA	5412	W		Too many parameters for function 'SSS'
UA	5414	W		Missing parameters for function 'SSS'
UA	5415	W	D	String function 'SSS' is not supported
UA	5501	E	N	Failed to send DEBUG information
UA	5601	E	D	TRANSLATE trouble- invalid sequence

Table 3. Error Messages

FAC	Code	SEV	Comment	Text
UA	5602	E		TRANSLATE (code conversion) not supported by remote host
UA	5603	W	A	Character code cannot be translated
UA	5604	E	D,N	Failed to capture NETEX code conversion table
UA	5605	E	D,N	Invalid translate code table length NNN
UA	5606	E	N	Failed on binary read of NETEX code table (NNN)
UA	5607	E	N	Failed to exchange TRANSLATE information
UA	5608	W		Invalid TRANSLATE value 'SSS'
UA	5609	W		TRANSLATE value SSS is out of range (0-NNN)
UA	5610	W		Translate IN_ONLY and OUT_ONLY are both on
UA	5701	E		Duplicate label 'SSS'
UA	5702	E		Label 'SSS' is not 1-NNN characters in length
UA	5703	E		Label 'SSS' contains invalid characters
UA	5704	E		Missing label 'SSS'
UA	5902	E		Invalid SSS tape label [SSS]
UA	5903	E		Bad volume serial number [SSS]
UA	5904	E		There is no tape device attached
UA	5905	E		Invalid variable block size NNN
UA	5906	E		Block Descriptor Word count NNN is invalid (NNN)
UA	5907	E		Record Descriptor Word count NNN is invalid (NNN)
UA	5908	E		Record length NNN is too large for buffer NNN
UA	5909	E		Invalid tape record format 'S'
UA	5910	E		Failed to skip NNN tape files
UA	5911	E		Failed to find tape file ID: SSS
UA	5912	E		Double tape mark encountered
UA	5913	E		Missing header label (HDR1)
UA	9999	E		Invalid numeric 'SSS' for switch 'SSS'
UA893	2002	E	R	Data checksum (CRC) error at block NNN
UA893	2004	E	R	Sequence number error at block NNN
UA893	2101	E		Failed to allocate NNN bytes of dynamic memory.
UA893	8001	E		Keyboard interrupt

Table 3. Error Messages

FAC	Code	SEV	Comment	Text
UA893	8201	E	H	Failed to start subprocess
UA893	8203	E	H	Failed to set INITIAL I/O timer on command output pipe
UA893	8204	E	H	Failed to set NO WAIT mode on output port 'NNN'
UA893	8205	E	H	Failed to set WAIT mode on output port 'NNN'
UA893	8206	E	H	Failed to set I/O timer on command output pipe
UA893	8207	E	H	Failed to read from the command output pipe
UA893	8208	E	H	Failed to write command 'SSS'
UA893	8209	E		Failed to synchronize with the UA command process
UA893	8301	E	H	Failed to expand pathname 'SSS'
UA893	8302	E		Missing SOURCE file.
UA893	8303	E	H	Failed to access file 'SSS'
UA893	8304	E	H	Failed to open file 'SSS'
UA893	8305	E	H	Failed to get port attributes
UA893	8306	E		Missing MAXRECORD specification
UA893	8307	E		A record length must be specified (-RECLENgth).
UA893	8308	E	H	Failed to create file 'SSS'
UA893	8309	E		File 'SSS' already exists with -CREATE NEW option
UA893	8310	E	H	Failed to delete BACKUP file 'SSS'
UA893	8311	E	H	Failed to rename 'SSS' to 'SSS'
UA893	8312	E	H	Failed to delete file 'SSS'
UA893	8313	E	H	Failed to open file 'SSS'
UA893	8314	E	H	Failed append to file 'SSS'
UA893	8315	E	H	Failed to get attributes for file 'SSS'
UA893	8316	E	H	Failed to get port attributes of file 'SSS'
UA893	8317	E		Missing MAXRECORD specification
UA893	8318	E	H	Failed to close file 'SSS'
UA893	8319	E	H	Failed to disable terminal echo
UA893	8320	E	H	Failed to get input from terminal
UA893	8321	E	H	Failed to restore terminal echo
UA893	8322	E	H	File read of NNN bytes failed
UA893	8324	E	H	File write of NNN bytes failed

Table 3. Error Messages

FAC	Code	SEV	Comment	Text
UA893	8326	E	H	Record read of NNN bytes failed
UA893	8328	E	H	Record write of NNN bytes failed
UA893	8330	E	H	Stream read of NNN bytes failed
UA893	8332	E	H	Stream write of NNN bytes failed
UA893	8334	E		Transfer mode 'NNN' is not supported
UA893	8335	E		Missing SOURCE file name
UA893	8336	E		Invalid wildcard specifier 'SSS'
UA893	8338	E		Bad ARCHIVE record length NNN
UA893	8339	E		Invalid ARCHIVE tag field [SSS]
UA893	8340	E		Unknown archive mode tag field [SSS]
UA893	8603	E	H	Invalid directory 'SSS'
UA893	8606	E		Invalid SPACE value 'SSS'
UA893	8701	E		Missing DESTINATION specifier
UA893	8703	E		Failed to flush command output pipe
UA893	8704	I		Piped command failed with an exit status of SSS
UA893	8705	E	H	Failed to create named pipe 'SSS'
UA893	8706	E	H	Failed to access file 'SSS'
UA893	8707	E	H	Failed to open named pipe 'SSS'
UA893	8708	E		Failed to read command output pipe
UA893	8709	E	H	Failed to set READ I/O timer on data pipe
UA893	8710	E	H	Failed to reset WAIT mode on command output pipe
UA893	8711	E	H	Failed to reset I/O timer on command output pipe
UA893	8712	E	H	Pipe read of NNN bytes failed
UA893	8713	E		Piped command completed before reading all data
UA893	8714	E	H	Failed to set WRITE I/O timer on data pipe
UA893	8715	E	H	Pipe write of NNN bytes failed
UA893	8716	E		Failed to execute piped command
UA893	8717	E		Failed to execute piped command
UA893	8903	E		Remote BATCH is not supported
UA893	9101	E	H	Failed to REWIND the magnetic tape
UA893	9102	E		Missing SOURCE name.

Table 3. Error Messages

FAC	Code	SEV	Comment	Text
UA893	9103	E	H	Failed to get the port id of port name 'SSS'
UA893	9104	E	H	Failed to open port name 'SSS', name
UA893	9105	E	H	Failed to close TAPE me
UA893	9107	E	H	Failed to close TAPE me
SI	4001	W	A	Invalid OPERATOR password
SI	4002	W		Service Initiator stopped
SI	4003	W		Trace flag settings: SSS
SI	4004	W		INFO not implemented
SI	4005	W		Invalid CONTROL request 'SSS'
SI	4201	W		Invalid OPERATOR password
SI	4202	W		Service Initiator stopped
SI	4203	W		Trace flag settings: SSS
SI	4204	W		INFO not implemented
SI	4205	W		Invalid CONTROL request 'SSS'
SI	4210	F	H	Failed to set working directory 'SSS'
SI	4211	F	H	Can't open OUTPUT log file 'SSS'
SI	4212	F	H	Can't open INPUT file 'SSS' in directory 'SSS'
SI	4213	F		Service 'SSS' is more than NNN characters
SI	4214	F		No services specified in configuration file
SI	4215	E		Failed to open protocol connection
SI	4216	E		Receive of CONNECT environment failed
SI	4217	E		Original service name is missing
SI	4218	E		Service 'SSS' is not offered by SI helper
SI	4219	F		Failed to offer service - cannot continue
SI893	8001	E		Missing USERNAME
SI893	8002	E		Invalid username.
SI893	8003	E		Invalid password
SI893	8008	E	H	Failed to write command 'SSS'
SI893	8012	E	H	Failed to read from the LOGIN output pipe

Additional Descriptions

This list provides additional descriptions for some USER-Access messages. These messages are marked in the preceding table with an 'A' in the comment column. The descriptions below expand on the information in the table.

UA-302 Overflow of NNN byte environment buffer

Severity: Error

Explanation: User data is stored in fixed length environment buffers and the string that was to be added caused the environment buffer to overflow.

UA-303 Failed to add 'SSS=SSS'

Severity: Error

Explanation: The variable name and its definition (truncated to 15 characters) will be displayed. SSS=SSS represents the variable addition to the environment that would not fit. To remedy this problem reduce the length of the name or the size of the description or if attempting to add to the GLOBAL environment, use the -GLOBAL switch when invoking USER-Access to increase the GLOBAL environment.

UA-501 Protocol error - expected [SS] - got [SS]

Severity: Error

Explanation: The protocol type in USER-Access did not match the protocol type of the remote host. The probable cause is either a network interruption or a revision-level incompatibility between the Initiator and the Responder.

UA-4106 The requested blocksize NNN was reduced to NNN

Severity: Informational

Explanation: The reduction (and resultant message) will only occur during the connect process. First, the local NETEX and remote NETEX perform a blocksize negotiation, and then there is a secondary blocksize negotiation between the USER-Access Responder and Initiator. During negotiation the requested blocksize gets sent to the remote host and a negotiated blocksize gets returned. The negotiated blocksize is always the smaller of the two hosts.

UA-4109 There were NNN CONNECT records ignored

Severity: Warning

Explanation: The records that are ignored are typically records coming from a newer release of the Responder than the Initiator. In this case the Responder sends more CONNECT information than the Initiator knows how to use. The message provides a warning that the connection may not support all of the functionality offered by the Responder.

UA-4127 The MESSAGE stack is empty

Severity: Error

Explanation: An error has occurred, but there is no message associated with the error.

UA-4131 Failed to establish secondary NETEX connection

Severity: Error

Explanation: Some NETEX Responders need a second connection to perform file transfers. There is likely to be a NETEX error (e.g. too many sessions); check the NETEX message if one is provided and retry. This error could occur as a result of a timeout or because of a revision-level incompatibility between the Initiator and the Responder.

UA-4132 Restricted command in server startup file**Severity:** Error**Explanation:** For security reasons, USER-Access server startup files may not contain any of the following commands: CONNECT, DISCONNECT, LOCAL, RECEIVE, REMOTE, or SEND. These commands may not be embedded within USER-Access aliases.**UA-4501 Nested (or recursive) input/alias limit of NNN exceeded****Severity:** Error**Explanation:** USER-Access restricts the number of times an input script or alias can call itself or another script/alias; the current limit is ten levels. This error can also be a result of a user failing to escape alias processing (using the '!' escape character) when redefining a USER-Access command as an alias within a multicommand alias.**UA-4504 Bad output FORMAT definition - reset to default****Severity:** Error**Explanation:** A user can redefine the format of error messages by using the SET OUTPUT FORMAT command. This message results when the new definition does not begin with '{ }' to disable string substitution.**UA-4505 Input request (NNN byte maximum) failed****Severity:** Error**Explanation:** USER-Access provides a buffer for holding a multiline command or alias; this error occurs when that buffer is exceeded. If a large command or alias is required, it should be defined as an Input Script.**UA-4601 Variable 'SSS' contains invalid characters****Severity:** Warning**Explanation:** A variable name was created that contains invalid characters; SSS represents the variable name. Valid characters are alphanumeric 'A'..'Z', '0'..'9'. It is recommended for future compatibility that variable names and alias names begin with an alpha character 'A'..'Z'.**UA-4701 Recursive alias 'SSS'****Severity:** Warning**Explanation:** A warning message resulted when USER-Access attempted to execute a single line alias that was recursive (it calls itself). A common cause of this error is executing an alias that calls an alias that calls the first alias back again.**UA-4704 Use SET LOCAL/REMOTE to modify SSS qualifier 'SSS'****Severity:** Warning**Explanation:** Some LOCAL and REMOTE qualifiers cannot be modified with some USER-Access commands. This error occurs if a user attempts to modify either current directory by means of a -DIR switch on a SEND, RECEIVE, LOCAL, or REMOTE command line.**UA-4709 Command token is greater than NNN characters****Severity:** Warning**Explanation:** A token is a sequence of characters separated by blanks, tabs, end-of-line, or any combination thereof. The token cannot exceed NNN length. Note that a token, enclosed in quotes, can include spaces.**UA-4804 MAXRECORD (NNN+NNN) too large for BLOCKSIZE (NNN)****Severity:** Error**Explanation:** MAXRECORD plus the record header size must be less than or equal to the BLOCKSIZE negotiated at connect time. To correct the error, reduce the MAXRECORD qualifier, reconnect with a larger BLOCKSIZE, or enable the PARTIALrecord qualifier.

UA893V-2004 Sequence number error at block NNN**Severity:** Error**Explanation:** File transfer with -CRC ON detected a missing block at block number SSS. The block sequence number is computed by the source host and verified by the destination host. The verification failed due to some network interruption. The file transfer is aborted and the destination file may not be complete. Correct the problem and retry the command.**UA893V-2101 Failed to allocate NNN bytes of dynamic memory.****Severity:** Error**Explanation:** This error indicates that the host (or user process) exceeded virtual memory limits. To remedy the problem, one could take action to increase virtual memory or reduce the number of open connections.**UA893-8201 Failed to start subprocess****Severity:** Error**Explanation:** USER-Access failed to start an interactive subprocess running the OpenVOS command interpreter. Refer to the associated STRATUS specific error.**UA893-8203 Failed to set INITIAL I/O timer on command output pipe****Severity:** Error**Explanation:** When executing LOCAL or REMOTE commands, USER-Access reads the output from a pipe file. To prevent blocking on the pipe file a timer is used to periodically interrupt a read and allow a KEYBOARD INTERRUPT. Refer to the associated STRATUS specific message.**UA893-8204 Failed to set NO WAIT mode on output port 'NNN'****Severity:** Error**Explanation:** If USER-Access is piping data from a running command, the output pipe must be in NO WAIT mode to prevent any delay when returning intermittent output (default output). Refer to the associated STRATUS specific message.**UA893-8205 Failed to set WAIT mode on output port 'NNN'****Severity:** Error**Explanation:** USER-Access forces WAIT mode on the command execution output pipe to prevent a tight loop when waiting for output from an executing command. This is in association with an I/O timer to allow the command to be interrupted. Refer to the associated STRATUS specific message.**UA893-8206 Failed to set I/O timer on the command output pipe****Severity:** Error**Explanation:** Failed to set I/O timer on the command execution output pipe. The timer is used to prevent a tight loop when waiting for output from an executing command. Refer to the associated STRATUS specific message.**UA893-8207 Failed to read from the command output pipe****Severity:** Error**Explanation:** USER-Access failed to read output returning from an executing command. Refer to the associated STRATUS specific message.**UA893-8208 Failed to write command 'SSS'****Severity:** Error**Explanation:** USER-Access failed to write the specified command to the process (ua_command_process) used for LOCAL or REMOTE command execution. Refer to the associated STRATUS specific message.

UA893-8209 Failed to synchronize with the UA command process

Severity: Error

Explanation: During the process of flushing/initializing the command output pipe an unexpected error occurred.

UA893-8301 Failed to expand pathname 'SSS'

Severity: Error

Explanation: USER-Access failed to expand a STRATUS OpenVOS relative pathname. Refer to the associated STRATUS specific message.

UA893-8302 Missing SOURCE name

Severity: Error

Explanation: A source file name is required for the specified operation.

UA893-8303 Failed to access file 'SSS'

Severity: Error

Explanation: In an attempt to get information about the source file an error occurred. Refer to the associated STRATUS specific message.

UA893-8304 Failed to open file 'SSS'

Severity: Error

Explanation: USER-Access failed to open the specified file for INPUT access. Refer to the associated STRATUS specific message.

UA893-8305 Failed to get port attributes

Severity: Error

Explanation: After a port ID has been assigned to the specified file or device, USER-Access failed to get information about the port. Refer to the associated STRATUS specific message.

UA893-8306 Missing MAXRECORD specification

Severity: Error

Explanation: A MAXRECORD specification is required for RECORD mode transfers.

UA893-8307 A record length must be specified (-RECLength)

Severity: Error

Explanation: If the destination file is a FIXED or RELATIVE file then a record length must be specified for the file to be created.

UA893-8308 Failed to create file 'SSS'

Severity: Error

Explanation: USER-Access failed to create the specified file. Refer to the associated STRATUS specific message.

UA893-8309 File 'SSS' already exists with -CREATE NEW option.

Severity: Error

Explanation: USER-Access attempted to create a file which already exists. A CREATE option other than NEW must be specified to overwrite or append to the existing file.

UA893-8310 Failed to delete BACKUP file 'SSS'

Severity: Error

Explanation: USER-Access failed to delete a BACKUP file if - CREATE BACKUP was specified. Refer to the associated STRATUS specific message.

UA893-8311 Failed to rename 'SSS' to 'SSS'

Severity: Error

Explanation: USER-Access failed to rename an existing file to its backup name ('filename'.bak) before creating a new file. Refer to the associated STRATUS specific message.

UA893-8312 Failed to delete file 'SSS'

Severity: Error

Explanation: USER-Access failed to delete an existing file if -CREATE DELETE was specified. Refer to the associated STRATUS specific message.

UA893-8313 Failed to open file 'SSS'

Severity: Error

Explanation: After creating the specified file USER-Access failed to open it. Refer to the associated STRATUS specific message.

UA893-8314 Failed to append to file 'SSS'

Severity: Error

Explanation: After creating/opening the specified file, USER-Access failed to position to EOF. Refer to the associated STRATUS specific message.

UA893-8315 Failed to get attributes for file 'SSS'

Severity: Error

Explanation: After creating/opening the specified file, USER-Access failed to get information about the file. Refer to the associated STRATUS specific message.

UA893-8316 Failed to get port attributes of file 'SSS'

Severity: Error

Explanation: After a port ID has been assigned to the specified file or device USER-Access failed to get information about the port. Refer to the associated STRATUS specific message.

UA893-8317 Missing MAXRECORD specification

Severity: Error

Explanation: A MAXRECORD specification is required for RECORD mode transfers.

UA893-8318 Failed to close file 'SSS'

Severity: Error

Explanation: USER-Access failed to close the specified file. Refer to the associated STRATUS specific message.

UA893-8319 Failed to disable terminal echo

Severity: Error

Explanation: USER-Access failed to turn terminal echo off (i.e., ASK -SECURE). Refer to the associated STRATUS specific message.

UA893-8320 Failed to get input from terminal

Severity: Error

Explanation: USER-Access failed to read input from the terminal. Refer to the associated STRATUS specific message.

UA893-8321 Failed to restore terminal echo

Severity: Error

Explanation: USER-Access failed to turn terminal echo back on. Refer to the associated STRATUS specific message.

UA893-8322 File read of NNN bytes failed

Severity: Error

Explanation: USER-Access failed to read the specified number of bytes. Refer to the associated STRATUS specific message.

UA893-8324 File write of NNN bytes failed

Severity: Error

Explanation: USER-Access failed to write the specified number of bytes. Refer to the associated STRATUS specific message.

UA893-8326 Record read of NNN bytes failed

Severity: Error

Explanation: USER-Access failed to read a record. Refer to the associated STRATUS specific message.

UA893-8328 Record write of NNN bytes failed

Severity: Error

Explanation: USER-Access failed to write a record. Refer to the associated STRATUS specific message.

UA893-8330 Stream read of NNN bytes failed

Severity: Error

Explanation: USER-Access failed to read a raw disk block. Refer to the associated STRATUS specific message.

UA893-8332 Stream write of NNN bytes failed

Severity: Error

Explanation: USER-Access failed to write a raw disk block. Refer to the associated STRATUS specific message.

UA893-8334 Transfer mode 'NNN' is not supported

Severity: Error

Explanation: The specified transfer mode is not supported by this release of USER-Access.

UA893-8335 Missing SOURCE file name

Severity: Error

Explanation: USER-Access failed to get the SOURCE file name during destination wildcard processing.

UA893-8336 Invalid wildcard specifier 'SSS'

Severity: Error

Explanation: The first asterisk found in the DESTINATION wildcarding specifier indicates the NAME portion of the filename. The second asterisk indicates the EXTENSION portion of the filename. An invalid wildcard specifier would be caused by any additional asterisk characters in the DESTINATION wildcarding specification on a SEND or RECEIVE command line.

UA893-8338 Bad ARCHIVE record length NNN

Severity: Error

Explanation: Each ARCHIVE record is preceded by a tag field. If the length of the record returned is less than the length of a tag, the record must not be valid.

UA893-8339 Invalid ARCHIVE tag field [SSS)

Severity: Error

Explanation: Each ARCHIVE record has an associated tag. The tags have a limited number of values. USER-Access fails if a tag does not have a valid value. Check to be sure the data is archive data and was backed up using BACKUP mode.

UA893-8340 Unknown archive mode tag field [SSS]

Severity: Error

Explanation: Each ARCHIVE record has an associated tag. The tags have a limited number of values. USER-Access fails if a tag does not have a valid value. Check to be sure the data is archive data and was backed up using BACKUP mode.

UA893-8603 Invalid directory 'SSS'

Severity: Error

Explanation: USER-Access attempted to change the current directory to an invalid value. Refer to the associated STRATUS specific message.

UA893-8606 Invalid SPACE value 'SSS'

Severity: Error

Explanation: The value of the SPACE qualifier on a SEND or RECEIVE command is invalid.

UA893-8701 Missing DESTINATION specifier

Severity: Error

Explanation: A destination specifier is required with a piped command. A destination name cannot be derived from a SOURCE name that is a command to execute.

UA893-8703 Failed to flush command output pipe

Severity: Error

Explanation: USER-Access failed to flush out any remaining command output (default output). This is done prior to closing the data pipe used for piped commands (i.e., (PIPE) tag).

UA893-8704 Piped command failed with an exit status of SSS

Severity: Error

Explanation: A piped command failed to complete with a successful status (nonzero). The exit status returned represents the status of the STRATUS command/macro. In most cases the status displayed is a valid OpenVOS exit status. Refer to the appropriate STRATUS/OpenVOS documentation for a description of the STRATUS error.

UA893-8705 Failed to create named pipe 'SSS'

Severity: Error

Explanation: USER-Access failed to create a temporary pipe file for use by USER-Access and the executing command to transfer data. Refer to the associated STRATUS specific message.

UA893-8706 Failed to access file 'SSS'

Severity: Error

Explanation: USER-Access failed to access the named pipe file specified. Refer to the associated STRATUS specific message.

UA893-8707 Failed to open named pipe 'SSS'

Severity: Error

Explanation: USER-Access failed to open the named pipe file specified. Refer to the associated STRATUS specific message.

UA893-8708 Failed to read the command output pipe

Severity: Error

Explanation: USER-Access failed to read the command output (default output) while piping data from the executing command. USER-Access continues to monitor the default output of the piped command for intermediate output, and end of command indication.

UA893-8709 Failed to set READ I/O timer on data pipe

Severity: Error

Explanation: USER-Access failed to put a timer on the named pipe. This is to prevent blocking on a read of the named pipe. Refer to the associated STRATUS specific message.

UA893-8710 Failed to reset WAIT mode on command output pipe

Severity: Error

Explanation: USER-Access failed to reset WAIT mode on the command output pipe after a close was detected on the data pipe. Refer to the associated STRATUS specific message.

UA893-8711 Failed to reset I/O timer on command output pipe

Severity: Error

Explanation: USER-Access failed to establish a timer on the command output pipe after WAIT mode had been reset. Refer to the associated STRATUS specific message.

UA893-8712 Pipe read of NNN bytes failed

Severity: Error

Explanation: USER-Access failed to read data from the named pipe. Refer to the associated STRATUS specific message.

UA893-8713 Piped command completed before reading all data

Severity: Error

Explanation: USER-Access failed because the piped command completed before processing all of the data.

UA893-8714 Failed to set WRITE I/O timer on data pipe

Severity: Error

Explanation: USER-Access failed to set an I/O timer on the data pipe for writing. The timer is needed to prevent possible blocking on the write. Refer to the associated STRATUS specific message.

UA893-8715 Pipe write of NNN bytes failed

Severity: Error

Explanation: USER-Access failed to write data to the named pipe. Refer to the associated STRATUS specific message.

UA893-8716 Failed to execute piped command

Severity: Error

Explanation: USER-Access failed to execute the piped command. Refer the associated command execution error(s).

UA893-8717 Failed to execute piped command

Severity: Error

Explanation: USER-Access failed to execute the piped command. Refer the associated command execution error(s).

UA893-8903 Remote BATCH is not supported

Severity: Error

Explanation: Unsupported feature.

UA893-9101 Failed to rewind the magnetic tape

Severity: Error

Explanation: USER-Access failed to rewind the magnetic tape. Refer to the associated STRATUS specific message.

UA893-9102 Missing SOURCE name

Severity: Error

Explanation: USER-Access requires a port name as the SOURCE for an ATTACH.

UA893-9103 Failed to get the port id of port name ‘SSS’

Severity: Error

Explanation: USER-Access failed to get the corresponding port id for the specified port name. Refer to the associated STRATUS specific message.

UA893-9104 Failed to open port name ‘SSS’

Severity: Error

Explanation: USER-Access failed to open the port attached to the magnetic tape device. Refer to the associated STRATUS specific message.

UA893-9105 Failed to close the TAPE file

Severity: Error

Explanation: USER-Access failed to close the open file associated with the current port attachment. Before continuing to the next file on the tape, the current file must be closed. Refer to the associated STRATUS specific message.

UA893-9107 Failed to close TAPE file

Severity: Error

Explanation: USER-Access failed to close the open file associated with the current port attachment. Refer to the associated STRATUS specific message.

SI-4001 Invalid OPERATOR password

Severity: Warning

Explanation: This warning is issued from the CONTROL program for the Service Initiator. The password was not specified or it was invalid.

SI893-8001 Missing USERNAME

Severity: Error

Explanation: A USERNAME is required to login to the host.

SI893-8002 Invalid username

Severity: Error

Explanation: The username specified is not a valid username for the host.

SI893-8003 Invalid password

Severity: Error

Explanation: The password specified is not a valid password for the username and host.

SI893-8008 Failed to write command ‘SSS’

Severity: Error

Explanation: The Service Initiator failed to write a command to the new process started under the specified user id. The command could either be the SERVER command or some other command passed on the CONNECT command line. Refer to the associated STRATUS specific message.

SI893-8012 Failed to read from the LOGIN output pipe

Severity: Error

Explanation: The Service Initiator failed to read output returning from the SERVER process (i.e., USER001). Refer to the associated STRATUS specific message.

Index

A

abbreviating alias names.....	86
accepting input.....	77
ADD function.....	55
alias definition.....	82
alias function.....	18
alias input.....	85
alias name abbreviation.....	86
alias names.....	86
alias parameters.....	84
alias types.....	83
arithmetic operations.....	55
ASCII.....	vii
ASK command.....	116
asynchronous.....	vii

B

Boolean qualifier.....	15
breaking a connection.....	24
buffer.....	vii

C

changing prompts.....	16
checking command status.....	82
CHR function.....	56
CMP function.....	57
code conversion.....	vii
command format.....	vi
ASK.....	116
CONNECT.....	119
CONTINUE.....	122
DISCONNECT.....	123
ENCRYPT.....	125
EXIT.....	127
GOTO.....	129
HELP.....	130
INPUT.....	132
LOCAL.....	135
ON.....	137
OUTPUT.....	140
QUIT.....	142
RECEIVE.....	143
REMOTE.....	144
SEND.....	146
SET.....	147
SET ALIAS.....	149
SET GLOBAL.....	151
SET HOST.....	153
SET VARIABLE.....	155

SHOW.....	157
SHOW ALIAS.....	158
SHOW GLOBAL.....	159
SHOW HOSTS.....	160
SHOW QUALIFIER.....	161
SHOW VARIABLE.....	162
TEXT.....	163
TRANSLATE.....	165
command interrupt.....	32
command qualifiers.....	13
command status.....	82
commands.....	13, 115
configuration manager.....	vii
CONNECT command.....	118
CONTINUE command.....	122
creating aliases.....	82

D

DATE function.....	58
debugging.....	87
alias.....	87
input script.....	87
DEC function.....	59
default values for file transfer.....	39
defining aliases.....	18, 82
description of commands.....	115
developing scripts.....	75
DFN function.....	60
DIRECTORY definition.....	40
disabling string substitution.....	74
DISCONNECT command.....	24, 123
disconnecting from a host.....	24
display output.....	76
displaying aliases.....	18
displaying qualifier values.....	14
displaying qualifiers.....	13
DIV function.....	55

E

editing files.....	31
ENCRYPT command.....	125
ENCRYPT function.....	61
ENV function.....	62
EQ function.....	67
EQS function.....	63
error message formatting.....	88
error messages.....	17
executing local commands.....	28
executing remote host commands.....	27
EXIT command.....	127

exiting USER-Access	19
EXT function	64
extended alias definition	84

F

file handling	39
file specification.....	41
file transfer.....	24
file transfer defaults	39
file transfer examples.....	42
file transfer qualifiers.....	26, 39

functions

ADD function.....	55
CHR function	56
CMP function	57
DATE function.....	58
DEC function.....	59
DFN function.....	60
DIV function	55
ENCRYPT function	61
ENV function	62
EQ function	67
EQS function.....	63
EXT function.....	64
GE function	67
GT function	67
INC function.....	59
INDEX function	65
LE function.....	67
LEN function.....	66
LOWER function	68
LT function.....	67
MOD function	55
MSG function.....	69
MUL function.....	55
NDF function.....	60
NE function	67
NES function.....	63
PARAMS function	70
SLEEP function.....	71
STATUS function.....	72
SUB function.....	55
TIME function.....	73
UPPER function	68

G

GE function.....	67
GOTO command	129
GOTO instruction	80
GT function.....	67

H

header.....	vii
HELP command.....	130
help files	94
user defined	94

host.....	vii
host independent commands	30
host information exchange	21

I

INC function	59
INDEX function.....	65
information exchange.....	21
INPUT command	16, 132
input files.....	75
input scripts	75
Internet Protocol (IP).....	vii
interrupting a command	32
interrupting string substitution	74
invoking USER-Access.....	11
ISO	vii

K

keyboard interrupt.....	32
-------------------------	----

L

labels	80
LE function	67
LEN function.....	66
limit to number of host connections	23
LOCAL command.....	28, 135
local commands.....	28
local startup files	12
local user guide	11
logical operations	67
LOGIN command.....	21
LOWER function	68
LT function	67

M

message formatting	88
messages.....	17
MOD function	55
MSG function.....	69
MUL function	55
multicommand alias	84
multiple connections	23
multiword alias names.....	86
MVS filehandling.....	39

N

NDF function	60
NE function	67
NES function.....	63
nested string substitution	74
NETEX.....	vii
Network Configuration Table (NCT).....	vii
NONE	92

O

ON command	137
------------------	-----

ON ERROR instruction	81
ON INTERRUPT instruction	81
online help	15
Open Systems Interconnection (OSI)	vii
OUTPUT command.....	16, 140
overriding qualifier values.....	15

P

PARAMS function	70
passing parameters.....	77, 84
path	vii
positional parameters.....	77
prompts.....	16

Q

qualifier value.....	14
qualifier values	15
qualifiers.....	26
Boolean	15
command line.....	15
qualifiers for file transfer.....	39
QUIT command.....	19, 142
quitting USER-Access.....	19

R

RECEIVE command.....	25, 143
RECEIVE qualifiers	26
REMOTE command.....	27, 144
remote editing operations	31
remote host	19
remote host commands	27
remote startup files	13
remote user's guide.....	33

S

script input	76
script output.....	76
scripts.....	75
scripts with string functions.....	78
SEARCH keywords.....	92
(NONE).....	92
(SITE).....	92
(USER).....	92
SEND command.....	24, 146
SEND qualifiers.....	26
SET ALIAS command.....	18, 149
SET command	14, 147
SET GLOBAL command	151
SET HOST command.....	23, 153
SET VARIABLE command	155
setting qualifiers	14
SHOW ALIAS command.....	18, 158

SHOW command.....	14, 157
SHOW GLOBAL command.....	159
SHOW HOSTS command	23, 160
SHOW INPUT command	16
SHOW LOCAL command.....	21
SHOW OUTPUT command	16
SHOW QUALIFIER command	13, 161
SHOW REMOTE command.....	22
SHOW VARIABLE command.....	162
SITE.....	92
SLEEP function	71
special characters	47
!	47
#	47
*	47
{	48
}	48
wildcard	43
special characters:	48
startup files.....	12
local	12
remote	13
STATUS function.....	72, 82
string functions.....	52
within scripts.....	78
string substitution.....	74, 78
disabling	74
nested.....	74
SUB function	55

T

TCP/IP	vii
temporary qualifier changes.....	14
TEXT command.....	76, 163
TIME function	73
transfer modes.....	45
transferring files	24
TRANSLATE command.....	164
types of aliases	83

U

UPPER function.....	68
USER	92
user defined help files	94
USER-Access HELP command	15, 130
USER-Access input and output.....	16
USER-Access invocation.....	11

W

wildcard characters	43
---------------------------	----