



eFTxx3
EFT™ for UNIX Systems

Release 5.4

Software Reference Manual

Revision Record

| Revision | Description |
|-------------------------|---|
| 01 (06/2001) | EFT manual released. |
| 02 (10/2009) | Miscellaneous updates. |
| 5.4 (07/2013) | Align to release versions- adding support for Linux |
| 5.4-01 (01/2014) | Minor corrections to port_min/port_max details |
| 5.4-02 (04/2014) | Minor corrections |

© 1999-2014 by Network Executive Software. Reproduction is prohibited without prior permission of Network Executive Software. Printed in U.S.A. All rights reserved.

You may submit written comments using the comment sheet at the back of this manual to:

Network Executive Software, Inc.
Publications Department
6420 Sycamore Lane, Suite 300
Maple Grove, MN 55369
USA

Comments may also be submitted over the Internet 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 eFTxx3 EFT software for UNIX systems. EFT is used in conjunction with TCP/IP allowing the end user to easily transfer files across the network.

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

The manual is organized as follows:

“Introduction” gives a basic description of EFT and includes a sample EFT session.

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

“UNIX Remote User's Guide” describes features of eFTxx3 as seen by a remote user. This includes executing commands on a UNIX host remotely and transferring files to and from a remote UNIX host.

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

“Advanced Local User's Guide” describes the advanced features of EFT on UNIX systems as seen by the local user.

“Command Descriptions,” “Host Independent Commands,” “General Alias Commands,” and “FTP Alias Commands” provide detailed descriptions of all commands available in eFTxx3 EFT.

“Installation Procedures” provides detailed instruction on product installation.

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 use of any product options or features that are 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, NetEx/IP, BFX, PFX, USER-Access, and eFT |
| Hewlett-Packard Company | HP, HP-UX |
| The Open Group | UNIX |
| International Business Machines | IBM, AIX, RISC 6000, RS/6000, System z |
| SUN Microsystems, Inc. | SUN, Solaris |
| Red Hat, Inc. | Red Hat |
| Linus Torvalds | Linux |

These references are made for informational purposes only.

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

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 <i>this font</i> . |
| user entry | <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. |

Contents

| | |
|---|------------|
| Revision Record | ii |
| Preface..... | iii |
| Notice to the Reader..... | iv |
| Corporation Trademarks and Products..... | iv |
| Document Conventions..... | v |
| Contents | vii |
| Figures..... | xvii |
| Tables..... | xvii |
| Introduction..... | 1 |
| EFT Overview..... | 1 |
| How EFT Works..... | 1 |
| eFT and UNIX | 2 |
| EFT and networking protocols..... | 2 |
| EFT and the ISO Model..... | 3 |
| Sample UNIX EFT Session | 4 |
| UNIX Local User's Guide | 11 |
| Introduction..... | 11 |
| Invoking EFT in UNIX | 11 |
| Local UNIX EFT Startup Files | 12 |
| Remote EFT Startup Files..... | 12 |
| Getting Started | 13 |
| EFT Commands and Command Qualifiers..... | 13 |
| Displaying the Valid Qualifiers for a Command | 13 |
| Displaying the Current Value of a Qualifier..... | 13 |
| Setting a Command Qualifier | 15 |
| Overriding a Command Qualifier | 15 |
| Online Help..... | 16 |
| Controlling EFT Input and Output..... | 16 |
| EFT Error Messages | 17 |
| Aliasing..... | 18 |
| Terminating an EFT Session..... | 19 |
| Establishing a Connection to a Remote Host..... | 19 |
| Using CONNECT to Establish a Connection | 19 |
| Using LOGIN to Establish a Connection..... | 20 |
| Exchanging Host Information on Connect | 21 |
| Establishing Multiple Host Connections | 22 |
| Disconnecting from a Host | 23 |
| 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 | 25 |
| Executing Remote Host Commands | 27 |

| | |
|--|-----------|
| Executing Local UNIX Commands | 28 |
| Issuing Local UNIX Host-Independent Commands | 30 |
| Editing Remote Files with a UNIX Editor | 31 |
| Interrupting a Command within UNIX EFT | 32 |
| Changing the Default UNIX SHELL | 32 |
| UNIX Remote User's Guide | 34 |
| Connecting into a UNIX Host | 34 |
| CONNECT Qualifiers Used by UNIX EFT | 34 |
| Remote UNIX EFT Startup Files | 34 |
| Transferring Files to a UNIX Host | 35 |
| Executing Remote UNIX Commands | 35 |
| Issuing Remote UNIX Host Independent Commands | 36 |
| File Handling Under UNIX EFT | 40 |
| UNIX File Transfer Qualifiers and Default Values | 40 |
| Definition of DIRECTORY Under UNIX EFT | 42 |
| UNIX File Specifications | 43 |
| UNIX File Specification Examples | 43 |
| File Transfer Examples from a Local UNIX Host | 43 |
| Example 1 | 43 |
| Example 2 | 44 |
| File Transfer Examples to a Remote UNIX Host | 44 |
| Example 1 | 44 |
| Example 2 | 44 |
| Source Wildcard Support for UNIX File Transfers | 44 |
| Destination Wildcard Support for UNIX File Transfers | 45 |
| Transfer Modes Supported Under UNIX EFT | 45 |
| Advanced Local User's Guide | 48 |
| Introduction | 48 |
| Special Characters | 48 |
| EFT String Substitution | 49 |
| String Variables | 50 |
| String Literals | 52 |
| String Functions | 53 |
| Arithmetic Operations | 56 |
| CHR Function | 57 |
| CMP Function | 58 |
| DATE Function | 59 |
| DEC and INC Functions | 60 |
| DFN and NDF Functions | 61 |
| ENCRYPT Function | 62 |
| ENV Function | 63 |
| EQS and NES Functions | 64 |
| EXT Function | 65 |
| INDEX Function | 66 |
| LEN Function | 67 |
| Logical Operations | 68 |
| LOWER and UPPER Functions | 69 |
| MSG Function | 70 |

| | |
|--|-----------|
| PARAMS Function..... | 71 |
| SLEEP Function..... | 72 |
| STATUS Function..... | 73 |
| TIME Function..... | 74 |
| Disabling String Substitution..... | 75 |
| Nested String Substitution..... | 75 |
| Developing EFT Scripts Using Input Files and Aliases..... | 76 |
| EFT Input Files..... | 76 |
| Echoing Input Scripts at the Terminal..... | 77 |
| Displaying Output and Accepting Input within a Script..... | 77 |
| Passing Parameters to a Script..... | 78 |
| Using String Functions within an EFT Script..... | 79 |
| Using EFT Labels and GOTOs..... | 80 |
| Using the ON (ERROR/INTERRUPT) Command..... | 81 |
| Checking Command Status..... | 82 |
| Creating EFT Aliases..... | 82 |
| EFT Aliases Vs Host Aliases..... | 83 |
| Creating Multicommand EFT Aliases..... | 84 |
| Passing Parameters to an Alias..... | 84 |
| Accepting Input within an EFT Alias..... | 85 |
| Abbreviating Alias Names..... | 86 |
| Defining Multiword Alias Names..... | 86 |
| Debugging an EFT Alias or Input Script..... | 87 |
| Error Message Formatting..... | 88 |
| EFT Code Conversion..... | 89 |
| EFT Data Verification..... | 89 |
| EFT Data Compression..... | 90 |
| Character Mode Compression..... | 91 |
| UNIX EFT SEARCH Keywords (SITE), (USER), and (NONE)..... | 91 |
| User-Definable HELP Files Under UNIX..... | 94 |
| Running EFT as a Batch Job Under UNIX..... | 95 |
| Running a UNIX Stand-Alone EFT Server..... | 96 |
| Advanced UNIX Transfer Modes..... | 97 |
| Command Descriptions..... | 99 |
| ASK Command..... | 100 |
| Description..... | 100 |
| Format..... | 100 |
| Examples..... | 101 |
| Related Topics..... | 101 |
| CONNECT Command..... | 102 |
| Description..... | 102 |
| Format..... | 103 |
| Host Dependencies..... | 104 |
| Examples..... | 104 |
| Related Topics..... | 105 |
| CONTINUE Command..... | 106 |
| Description..... | 106 |
| Format..... | 106 |
| Example..... | 106 |
| Related Topics..... | 106 |

| | |
|--------------------------------|-----|
| DISCONNECT Command | 107 |
| Description | 107 |
| Format | 107 |
| Examples | 107 |
| Related Topics | 107 |
| EXIT Command | 108 |
| Description | 108 |
| Format | 108 |
| Examples | 108 |
| Related Topics | 108 |
| GOTO Command | 109 |
| Description | 109 |
| Format | 109 |
| Examples | 109 |
| Related Topics | 109 |
| HELP Command | 110 |
| Description | 110 |
| Format | 110 |
| Examples | 110 |
| INPUT Command..... | 112 |
| Description | 112 |
| Format | 112 |
| Examples | 113 |
| Related Topics | 113 |
| LOCAL Command | 114 |
| Description | 114 |
| Format | 114 |
| Informational Qualifiers | 114 |
| Examples | 115 |
| Related Topics | 115 |
| ON Command..... | 116 |
| Description | 116 |
| ON ERRor | 116 |
| ON INTerrupt | 116 |
| ON LOCAl_error..... | 116 |
| ON REMote_error | 116 |
| Format | 116 |
| Examples | 117 |
| Related Topics | 117 |
| OUTPUT Command..... | 119 |
| Description | 119 |
| Format | 119 |
| Informational Qualifiers | 120 |
| Examples | 120 |
| Related Topics | 120 |
| QUIT Command..... | 121 |
| Description | 121 |
| Format | 121 |
| Examples | 121 |
| Related Topics | 121 |
| RECEIVE Command..... | 122 |

| | |
|--------------------------------|-----|
| Description..... | 122 |
| Format..... | 122 |
| Examples..... | 122 |
| Related Topics | 122 |
| REMOTE Command..... | 123 |
| Description..... | 123 |
| Format..... | 123 |
| Informational Qualifiers | 123 |
| Examples..... | 124 |
| Related Topics | 124 |
| SEND Command..... | 125 |
| Description..... | 125 |
| Format..... | 125 |
| Examples..... | 125 |
| Related Topics | 125 |
| SET Command..... | 126 |
| Description..... | 126 |
| Format..... | 126 |
| Examples..... | 126 |
| Related Topics | 127 |
| SET ALIAS Command | 128 |
| Description..... | 128 |
| Format..... | 128 |
| Host Dependencies | 128 |
| Examples..... | 128 |
| Related Topics | 129 |
| SET GLOBAL Command..... | 130 |
| Description..... | 130 |
| Format..... | 130 |
| Example | 130 |
| Related Topics | 130 |
| SET HOST Command | 132 |
| Description..... | 132 |
| Format..... | 132 |
| Examples..... | 132 |
| Related Topics | 132 |
| SET VARIABLE Command..... | 134 |
| Description..... | 134 |
| Format..... | 134 |
| Examples..... | 134 |
| Related Topics | 134 |
| SHOW Command | 135 |
| Description..... | 135 |
| Format..... | 135 |
| Examples..... | 135 |
| Related Topics | 135 |
| SHOW ALIAS Command | 136 |
| Description..... | 136 |
| Format..... | 136 |
| Examples..... | 136 |
| Related Topics | 136 |

| | |
|---------------------------------------|------------|
| SHOW GLOBAL Command..... | 137 |
| Description | 137 |
| Format | 137 |
| Examples | 137 |
| Related Topics..... | 137 |
| SHOW HOST Command | 138 |
| Description | 138 |
| Format | 138 |
| Examples | 138 |
| Related Topics..... | 138 |
| SHOW QUALIFIER Command..... | 139 |
| Description | 139 |
| Format | 139 |
| Examples | 139 |
| Related Topics..... | 139 |
| SHOW VARIABLE Command..... | 140 |
| Description | 140 |
| Format | 140 |
| Examples | 140 |
| Related Topics..... | 140 |
| TEXT Command | 141 |
| Description | 141 |
| Format | 141 |
| Examples | 141 |
| Related Topics..... | 141 |
| TRANSLATE Command | 142 |
| Description | 142 |
| Format | 143 |
| Examples | 143 |
| Host Independent Commands..... | 147 |
| General Alias Commands | 149 |
| ASSIGN Alias Command..... | 150 |
| Description | 150 |
| Format | 150 |
| Examples | 150 |
| Related Topics..... | 150 |
| DEBUGOFF Alias Command..... | 151 |
| Description | 151 |
| Format | 151 |
| Related Topics..... | 151 |
| DEBUGON Alias Command..... | 152 |
| Description | 152 |
| Format | 152 |
| Related Topics..... | 152 |
| EDIT Alias Command..... | 153 |
| Description | 153 |
| Format | 153 |
| Examples | 153 |
| Related Topics..... | 153 |

| | |
|--------------------------------|-----|
| ENCRYPT Alias Command | 154 |
| Description..... | 154 |
| Format..... | 154 |
| Examples..... | 154 |
| Example 1 | 154 |
| Example 2 | 154 |
| Example 3 | 155 |
| Related Topics | 155 |
| LDAP Alias Command | 156 |
| Description..... | 156 |
| Format..... | 156 |
| Examples..... | 156 |
| Related Topics | 156 |
| LEDIT Alias Command | 157 |
| Description..... | 157 |
| Format..... | 157 |
| Examples..... | 157 |
| Related Topics | 157 |
| LOGIN Alias Command | 158 |
| Description..... | 158 |
| Format..... | 158 |
| Examples..... | 158 |
| Related Topics | 158 |
| RDIR Alias Command | 159 |
| Description..... | 159 |
| Format..... | 159 |
| Examples..... | 159 |
| Related Topics | 159 |
| SET LOGIN Alias Command | 160 |
| Description..... | 160 |
| Format..... | 160 |
| Examples..... | 160 |
| Related Topics | 161 |
| SHOW LOGIN Alias Command | 162 |
| Description..... | 162 |
| Format..... | 162 |
| Examples..... | 162 |
| Related Topics | 162 |
| SLD Alias Command | 163 |
| Description..... | 163 |
| Format..... | 163 |
| Examples..... | 163 |
| Related Topics | 163 |
| SRD Alias Command | 164 |
| Description..... | 164 |
| Format..... | 164 |
| Examples..... | 164 |
| Related Topics | 164 |
| TEST Alias Command | 165 |
| Description..... | 165 |
| Format..... | 165 |

| | |
|---------------------------------|------------|
| Examples | 165 |
| Related Topics | 165 |
| FTP Alias Commands | 167 |
| ACCOUNT Alias Command..... | 168 |
| Description | 168 |
| Format | 168 |
| Examples | 168 |
| Related Topics | 168 |
| APPEND Alias Command..... | 169 |
| Description | 169 |
| Format | 169 |
| Examples | 169 |
| Related Topics | 169 |
| ASCII Alias Command..... | 170 |
| Description | 170 |
| Format | 170 |
| Examples | 170 |
| Related Topics | 170 |
| BIN Alias Command | 171 |
| Description | 171 |
| Format | 171 |
| Examples | 171 |
| Related Topics | 171 |
| BYE Alias Command | 172 |
| Description | 172 |
| Format | 172 |
| Examples | 172 |
| Related Topics | 172 |
| CD Alias Command | 173 |
| Description | 173 |
| Format | 173 |
| Examples | 173 |
| Related Topics | 173 |
| CLOSE Alias Command | 174 |
| Description | 174 |
| Format | 174 |
| Examples | 174 |
| Related Topics | 174 |
| DELETE Alias Command | 175 |
| Description | 175 |
| Format | 175 |
| Examples | 175 |
| Related Topics | 175 |
| DIR Alias Command | 176 |
| Description | 176 |
| Format | 176 |
| Examples | 176 |
| Related Topics | 176 |
| FTP Alias Command | 177 |
| Description | 177 |

| | |
|---------------------------|-----|
| Format..... | 177 |
| Examples..... | 177 |
| Related Topics | 177 |
| GET Alias Command..... | 178 |
| Description..... | 178 |
| Format..... | 178 |
| Examples..... | 178 |
| Related Topics | 178 |
| LCD Alias Command..... | 179 |
| Description..... | 179 |
| Format..... | 179 |
| Examples..... | 179 |
| Related Topic..... | 179 |
| LS Alias Command..... | 180 |
| Description..... | 180 |
| Format..... | 180 |
| Examples..... | 180 |
| Related Topics | 180 |
| LSMem Alias Command | 181 |
| Description..... | 181 |
| Format..... | 181 |
| Examples..... | 181 |
| Related Topics | 181 |
| MKDIR Alias Command | 182 |
| Description..... | 182 |
| Format..... | 182 |
| Examples..... | 182 |
| Related Topics | 182 |
| OPEN Alias Command | 183 |
| Description..... | 183 |
| Format..... | 183 |
| Examples..... | 183 |
| Related Topics | 183 |
| PUT Alias Command | 184 |
| Description..... | 184 |
| Format..... | 184 |
| Examples..... | 184 |
| Related Topics | 184 |
| PWD Alias Command..... | 185 |
| Description..... | 185 |
| Format..... | 185 |
| Examples..... | 185 |
| Related Topics | 185 |
| RENAME Alias Command..... | 186 |
| Description..... | 186 |
| Format..... | 186 |
| Examples..... | 186 |
| Related Topics | 186 |
| RM Alias Command | 187 |
| Description..... | 187 |
| Format..... | 187 |

| | |
|--|------------|
| Examples | 187 |
| Related Topics | 187 |
| RMDIR Alias Command | 188 |
| Description | 188 |
| Format | 188 |
| Examples | 188 |
| Related Topics | 188 |
| Installation Procedures | 189 |
| Prerequisites | 189 |
| Pre-Installation | 189 |
| Accessing the UNIX eFT software distribution | 189 |
| Getting the NESi Public Key | 189 |
| Importing the NESi Public Key | 189 |
| Verifying Signatures | 189 |
| Installation | 189 |
| Upgrading eFTxx3 | 190 |
| Removing eFTxx3 | 191 |
| Removing the NESi Public Key | 191 |
| Starting, Stopping & Verifying the eFT Installation | 191 |
| For Unix/Linux systems: | 191 |
| For AIX: | 191 |
| Completion and Testing of Installation | 191 |
| Appendix A. Updating the TCP/IP Network Control Files | 193 |
| Appendix B. Stopping the Service Initiator | 195 |
| Appendix C. Service Initiator Keywords | 197 |
| Appendix D. Product Configuration File | 200 |
| Appendix E. License Key File | 202 |
| Appendix F. EFT Error Messages for UNIX | 204 |
| Additional Descriptions | 213 |
| Index | 220 |

Figures

Figure 1. Diagram of an eFT Connection Sequence2

Figure 2. ISO Model Communication.....4

Figure 3. Nested String Substitution.....75

Figure 4. Sample Command Script.....96

Tables

Table 1. ISO Model.....3

Table 2. List of Functions54

Table 3. COMPRESS/EXPAND combinations91

Table 4. Combinations of CHARACTER mode compress/expand91

Table 5. Host Independent Commands147

Table 6. Error Messages205

Introduction

EFT Overview

The Network Executive Software (NESi) EFT™ software is a user interface to TCP/IP. It provides the ordinary user with a means to move and manipulate files across a network using simple, easily remembered commands. In addition, EFT provides extensive interactive help files so the user can become familiar with EFT.

EFT provides several advantages to network users. Among these are:

- **User-friendly** - Once EFT is installed, you can transfer files and exercise other EFT functions in very little time and with little training.
- **Tailorable** - The EFT interface can be tailored to meet your needs at the host and user levels. Default values can be set, aliases defined, etc., in site- and user-input files that are read by EFT when it is invoked.
- **Common Interface** - The EFT 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** - EFT 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.

How EFT Works

All EFT products, regardless of platform, follow the same pattern of function. This pattern is illustrated in Figure 1 on page 2. As the figure shows, the client (initiator) sends a request to the remote (or responding) Service Initiator (on a well-known, configured or default port 6900), including account and password information (1). The responding Service Initiator logs the user in and starts up a Service Module (2). The Service Module then offers a service (on one of the system or configured ephemeral ports) and notifies the Service Initiator (3). The Service Initiator then returns a message to the client (initiator) with the ephemeral port to connect to and disconnects to wait for another incoming request (4). The client (initiator) then connects directly to the Service Module (5) on the specified ephemeral port. The connection between the client (initiator) and the Service Module is maintained until terminated by a command or a system timeout. If the ephemeral port needs to be restricted there are optional 'server' parameters (port_min/port_max) which can be used to select a port range and these ports must be opened on the firewall.

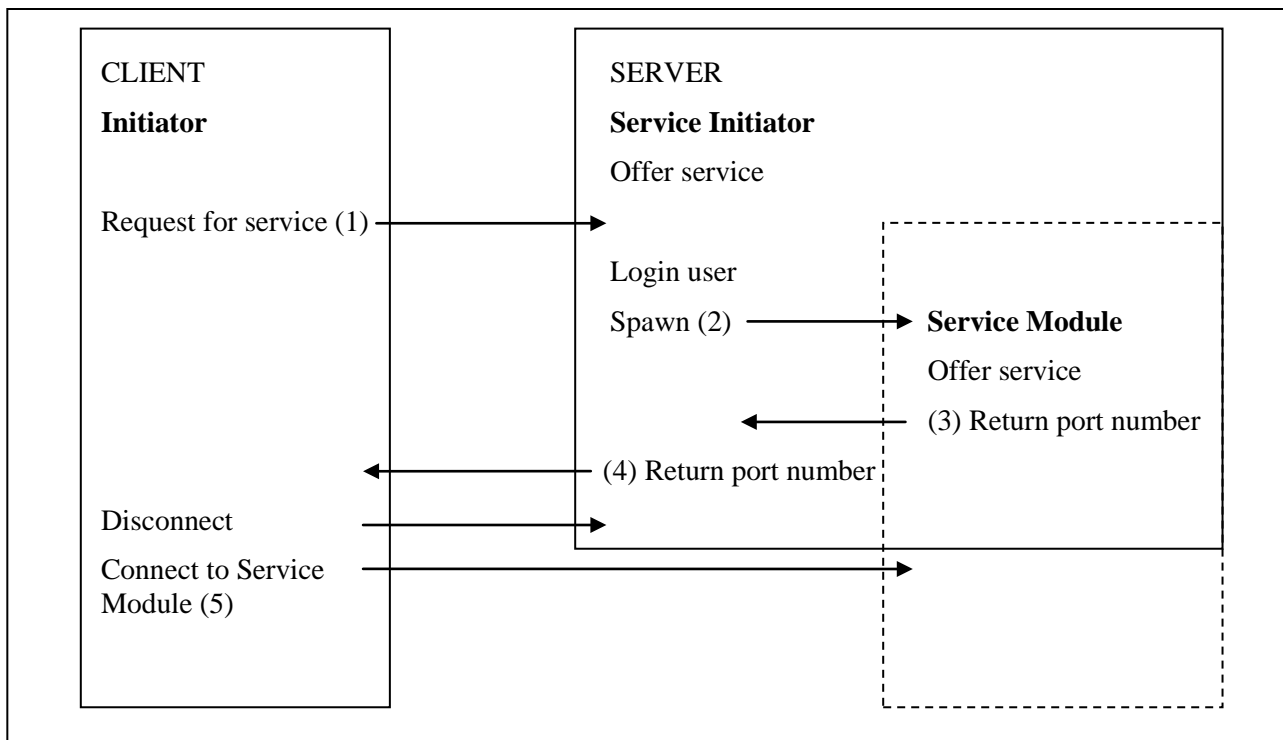


Figure 1. Diagram of an eFT Connection Sequence

eFT and UNIX

The spread of UNIX has allowed the UNIX interface to EFT to be standardized and presented as the eFTxx3 EFT product. This manual describes the EFT software for a UNIX host running the appropriate version of TCP/IP.

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

The user interface allows EFT to request services from other EFT hosts, to perform file transfers, and to submit remote commands. EFT will also accept requests from other EFT servers.

EFT and networking protocols

NESi's EFT is a software package that extends file transfer capabilities to the less technical end-user. EFT has easy-to-use commands that direct the networking software (TCP/IP) to make connections, transfer files, and carry out related activities.

EFT is used as a standard TCP/IP application to enable communications between two or more application programs (which may be running on different hosts) to communicate with each other at multimegabit speeds.

The following sections describe how EFT and the network protocol conform to the International Standards Organization (ISO) guidelines for open systems interconnection.

EFT and the ISO Model

In creating EFT, NESi 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. EFT forms the application (or user) layer and the presentation layer of the ISO model. Each of the seven layers interact 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.

| Table 1. ISO Model | |
|---------------------------|---|
| 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 transport layer 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 |

Although each layer physically interacts only with adjacent layers, each layer appears to communicate directly with the corresponding layer of the other model. Figure 2 on page 4 illustrates this concept.

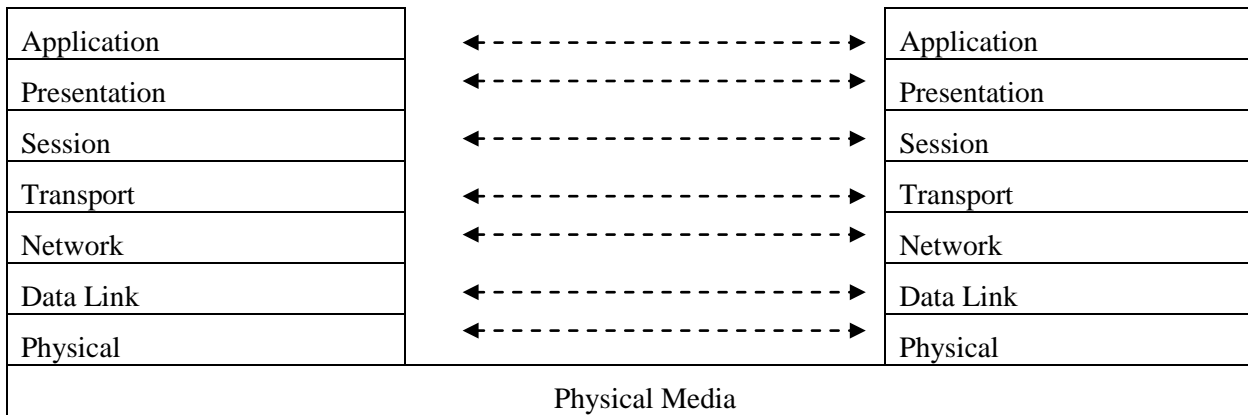


Figure 2. ISO Model Communication

Notice that the corresponding layers appear to communicate directly as indicated by the dotted lines, but actually 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.

Sample UNIX EFT Session

This section gives a very brief example of a few of the functions that can be accomplished during a EFT session. This sample session is meant to be only a simple introduction to EFT and how it may appear to the local UNIX user. The sections following this provide a more detailed look at the product and its features. Users that have never seen EFT 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 EFT, the user command is entered from the UNIX command line as:

```
$ eftip-client
EFT>
```

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

A connection to any host in the network that is running EFT can be made using the LOGIN command. The LOGIN command below establishes a connection with a Solaris host named 'sunrise'. LOGIN prompts the user for various 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 EFT prompt appears. Notice that in this session, EFT has been configured to prompt with the name of the remote host 'sunrise'.

```
EFT> login
Hostname? sunrise
Username? test1
Password?
Qualifiers?
EFT: Connected to Service Initiator on host 'sunrise'.
=====
Last login: Fri Oct  4 11:15:28 on pts/9
Sun Microsystems Inc.  SunOS 5.10      Generic January 2005
=====
EFT: Logged in as user 'test1'.
```

```
EFT: Connected to service '3000' on host 'sunrise'.
EFT: *
EFT: * Entered server.ua file
EFT: *
sunrise>
```

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

```
EFT: Failed to connect service 'EFT' on host 'sunrise' (UA-4105).
```

This is followed by either an appropriate network or remote system error message. For example, if a host by the name 'sunrise' does not exist in the network, for example, an error similar to the following would appear:

```
EFT: Failed to connect service 'EFT' on host 'sunrise' (UA-4105).
EFT: Host 'sunrise' does not exist in configuration (UA-804).
EFT: Invalid TCP host name 'sunrise' (EFT803-2501).
```

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

```
=====
Login incorrect
=====
EFT: Failed to connect service 'EFT' on host 'sunrise' (UA-4105).
EFT: Remote: Login failed (SI693-8011).
```

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 HOST command can be used to display all remote host connections held by this EFT 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.

```
sunrise> show host
EFT:
EFT: active --> (1) Host=sunrise   User=test1
EFT:
sunrise>
```

Once a connection is established, a SHOW REMOTE command can be issued to return useful information about the connection and the remote EFT host. From the list below, for example, it can be seen that the remote host character code is 'ASCII7', the default directory, is '/mnt/home2/test1', and the EFT version number is '5.2'.

```
sunrise> show remote
EFT:
EFT: * BLOCKsize ..... 16384
EFT: * COPYRight ..... COPYRIGHT (c) 1999-2009 - Network Executive Software,
EFT:   Inc.
EFT:   DIRectory ..... /mnt/home2/test1
EFT: * GATEway .....
EFT:   HOMEdir ..... /mnt/home2/test1
EFT: * HOST ..... sunrise
EFT: * HOSTCODE ..... ASCII7
EFT: * HOSTTYPE ..... UNIX
EFT: * NODE .....
EFT: * PID ..... 15995
EFT:   PREFix ..... Unix:
EFT: * PRODUCT ..... EFT693
EFT:   QUIet ..... off
```

```

EFT: * ROOTdir ..... /usr/nesi/eft/user/
EFT: * SERvice ..... 3000
EFT:  SHELL ..... /bin/bash
EFT: * STATus .....
EFT: * TRANSlate ..... Network
EFT: * USERname ..... test1
EFT: * VERsion ..... 5.2
EFT:
EFT: * Informational qualifier (cannot be modified).
EFT:
sunrise>

```

Similar information can also be displayed about the local Linux host by issuing the `SHOW LOCAL` command. Note here that the local character code is 'ASCII7', the current local directory is '/mnt/home2/test2', and the local version of EFT is '5.4'.

```

sunrise> show local
EFT:
EFT: * COPYRight ..... Copyright (C) 1999-2013, Network Executive Software,
EFT: Inc.
EFT:  DIRectory ..... /mnt/home2/test2
EFT: * GATEway .....
EFT:  HOMEdir ..... /mnt/home2/test2
EFT: * HOSTCODE ..... ASCII7
EFT: * HOSTTYPE ..... UNIX
EFT:  INTeractive ..... off
EFT: * LicExp ..... 20131026
EFT: * LicKey ..... CO3Z-YAAA-ABAA-CRCZ-DWYU-POYU
EFT: * LicNotOper ..... 20131026
EFT: * NETwork ..... TCPIP
EFT: * PID ..... 22462
EFT:  PREFix ..... Unix:
EFT: * PRODuct ..... EFT803
EFT:  QUIet ..... off
EFT: * ROOTdir ..... /usr/share/nesi/eftip/user
EFT:  SHELL ..... /bin/bash
EFT: * STATus .....
EFT: * USERname ..... test2
EFT: * VERsion ..... 5.4
EFT:
EFT: * Informational qualifier (cannot be modified).
EFT:
sunrise>

```

Once a connection is established to a remote host, users can issue commands to that host using the `REMOTE` command. The example below issues a `REMOTE 'ls -la'` 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 'UNIX' host.

```

sunrise> ls -l tst*
Unix: -rw-rw-r--  1 test1   nesi           2719 Oct 20   2009 tstcmp
Unix: -rw-rw-r--  1 test1   nesi           7741 Dec  3   2009 tstcrc
Unix: -rw-rw-r--  1 test1   nesi           7740 Nov 25   2009 tstfile
Unix: -rw-rw-r--  1 test1   nesi           8080 Sep 11   2009 tstfile1
Unix: -rw-rw-r--  1 test1   nesi           7741 Nov 20   2009 tstfile2
Unix: -rw-rw-r--  1 test1   nesi              0 Oct  4 09:31 tstfile9
Unix: -rw-rw-r--  1 test1   nesi           7741 Nov 23   2009 tstfile9.bak
sunrise>

```


A major feature of EFT 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. EFT simply maps DIRECTORY to the UNIX `'bin/ls -al'` command. Now network users need only learn one network-wide command set. This command set can be the EFT 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.

```
sunrise> remote directory tst*
Unix: -rw-rw-r--  1 test1  nesi      2719 Oct 20  2009 tstcmp
Unix: -rw-rw-r--  1 test1  nesi      7741 Dec  3  2009 tstcrc
Unix: -rw-rw-r--  1 test1  nesi      7740 Nov 25  2009 tstfile
Unix: -rw-rw-r--  1 test1  nesi      8080 Sep 11  2009 tstfile1
Unix: -rw-rw-r--  1 test1  nesi      7741 Nov 20  2009 tstfile2
Unix: -rw-rw-r--  1 test1  nesi         0 Oct  4 09:31 tstfile9
Unix: -rw-rw-r--  1 test1  nesi      7741 Nov 23  2009 tstfile9.bak
sunrise>
```

Local UNIX commands or local Host Independent Commands can also be executed from within EFT 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 **Unix**.

```
sunrise> local directory
Unix: total 228
Unix: drwxr-xr-t 11 test2 nesi  4096 2012-09-27 23:41 .
Unix: drwxr-xr-x 97 root  root  4096 2013-05-22 10:49 ..
Unix: -rw-----  1 test2 nesi  5215 2013-10-04 14:24 .bash_history
Unix: -rw-r--r--  1 test2 nesi   191 2008-05-14 09:29 .bash_profile
Unix: -rw-r--r--  1 test2 nesi  3265 2012-09-27 23:41 .bashrc
Unix: drwx-----  2 test2 nesi  4096 2008-05-16 15:59 Desktop
Unix: drwx-----  2 test2 nesi  4096 2008-05-16 15:57 .gconf
Unix: drwx-----  2 test2 nesi  4096 2008-05-16 15:58 .gconfd
Unix: -rw-----  1 test2 nesi   217 2008-05-16 15:50 .ICEauthority
Unix: drwxr-xr-x  4 test2 nesi  4096 2008-05-16 15:46 .kde
Unix: drwx-----  3 test2 nesi  4096 2008-05-16 15:46 .local
Unix: drwxr-xr-x  3 test2 nesi  4096 2008-05-16 15:46 .mcp
Unix: -rw-r--r--  1 test2 nesi 19260 2011-11-15 16:22 nct
Unix: drwxr-xr-x  2 test2 nesi  4096 2008-05-16 16:02 .qt
Unix: -rw-r--r--  1 test2 nesi  7740 2009-11-17 10:43 testfile
Unix: -rw-r--r--  1 test2 nesi   184 2009-11-13 09:25 ua.bck
Unix: -rw-----  1 test2 nesi  2353 2008-05-16 16:07 .viminfo
Unix: -rw-r--r--  1 test2 nesi   597 2008-05-14 09:29 .vimrc
Unix: drwxr-xr-x  2 test2 nesi  4096 2013-07-22 14:37 .vnc
Unix: -rw-----  1 test2 nesi    65 2009-09-17 09:35 .xauthUE85EK
Unix: -rw-r--r--  1 test2 nesi 12952 2008-05-16 15:50 xstart.log
Unix: -rw-r--r--  1 test2 nesi     0 2008-05-16 15:45 xstart.xrdbq.log1
sunrise>
```

To transfer a file from the local host to the remote host, the SEND command is used. The example below sends the file `xstart.log` from the current local directory `/mnt/home2/test2` on the UNIX host to the current remote directory (or TSO/E Prefix) TEST1 on an IBM zOS host. Since all EFT commands can be pre-defined 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 EFT uses the source file name to create a default destination file name when one isn't specified.

```

zos5> send xstart.log
EFT: Source                               Destination                               Size
EFT: -----                               -----                               -----
EFT: /mnt/home2/test2/xstart.log          TEST1.XSTART.LOG                        12952
zos5>

```

With EFT it is also very easy to transfer a group of files using a single command. The example below sends all of the local starting with xstart to the remote IBM zOS host. Per the user's request, the files are stored in a single IBM Partitioned Data Set (PDS) as individual members XSTART1 and XSTART2. If any file transfer errors were encountered, they would be displayed in place of the status line below.

```

zos5> send xstart* sam.pds(*)
EFT: Source                               Destination                               Size
EFT: -----                               -----                               -----
EFT: /mnt/home2/test2/xstart1            TEST1.SAM.PDS(XSTART1)                  12952
EFT: /mnt/home2/test2/xstart2            TEST1.SAM.PDS(XSTART2)                  11781
zos5>

```

A quick REMOTE DIRECTORY will act as a second verification that the files have indeed been transferred. Note the new files SAM.PDS and XSTART.LOG below:

```

zos5> remote directory
MVS: TEST1.SAM.PDS
MVS: TEST1.XSTART.LOG
zos5>

```

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 WRAP from the IBM zOS system to the local Linux host.

```

zos5> receive test1.wrap
EFT: Source                               Destination                               Size
EFT: -----                               -----                               -----
EFT: TEST1.WRAP                           /mnt/home2/test2/wrap                    254
zos5>

```

This transfer can too be verified by viewing a LOCAL DIRECTORY LISTING.

```

zos5> local dir
Unix: total 268
Unix: drwxr-xr-t 11 test2 nesi 4096 2013-10-04 15:27 .
Unix: drwxr-xr-x 97 root root 4096 2013-05-22 10:49 ..
Unix: -rw----- 1 test2 nesi 5215 2013-10-04 14:24 .bash_history
Unix: -rw-r--r-- 1 test2 nesi 191 2008-05-14 09:29 .bash_profile
Unix: -rw-r--r-- 1 test2 nesi 3265 2012-09-27 23:41 .bashrc
Unix: drwx----- 2 test2 nesi 4096 2008-05-16 15:59 Desktop
Unix: drwx----- 2 test2 nesi 4096 2008-05-16 15:57 .gconf
Unix: drwx----- 2 test2 nesi 4096 2008-05-16 15:58 .gconfd
Unix: -rw----- 1 test2 nesi 217 2008-05-16 15:50 .ICEauthority
Unix: drwxr-xr-x 4 test2 nesi 4096 2008-05-16 15:46 .kde
Unix: drwx----- 3 test2 nesi 4096 2008-05-16 15:46 .local
Unix: drwxr-xr-x 3 test2 nesi 4096 2008-05-16 15:46 .mcpop
Unix: -rw-r--r-- 1 test2 nesi 19260 2011-11-15 16:22 nct
Unix: drwxr-xr-x 3 test2 nesi 4096 2008-05-16 15:58 .openoffice.org2.0
Unix: drwxr-xr-x 2 test2 nesi 4096 2008-05-16 16:02 .qt
Unix: -rw-r--r-- 1 test2 nesi 7740 2009-11-17 10:43 testfile
Unix: -rw-r--r-- 1 test2 nesi 184 2009-11-13 09:25 ua.bck
Unix: -rw----- 1 test2 nesi 3566 2013-10-04 15:16 .viminfo
Unix: -rw-r--r-- 1 test2 nesi 597 2008-05-14 09:29 .vimrc

```

```
Unix: drwxr-xr-x  2 test2 nesi  4096 2013-07-22 14:37 .vnc
Unix: -rw-r--r--  1 test2 nesi   254 2013-10-04 15:27 wrap
Unix: -rw-----  1 test2 nesi    65 2009-09-17 09:35 .xauthUE85EK
Unix: -rw-r--r--  1 test2 nesi 12952 2008-05-16 15:50 xstart1
Unix: -rw-r--r--  1 test2 nesi 11781 2013-10-04 15:16 xstart2
Unix: -rw-r--r--  1 test2 nesi 12952 2013-10-04 15:24 xstart.log
zos5>
```

To force a disconnection from all remote hosts (in this case the IBM zOS 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 EFT session status that can be interpreted by the local Linux host. This status is especially useful when EFT is used within a batch job.

```
zos5> exit
[15:29:55 test2@suse1 ~ 0]$
```

To keep this sample session short, no more commands or features of EFT will be shown. However, since only a small fraction of EFT 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.

UNIX Local User's Guide

Introduction

This section is intended for UNIX users that would like an introduction to EFT and some of its features. This section explains how to invoke EFT from a UNIX terminal, what a EFT 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 the "Advanced Local User's Guide" on page 48 for a more in depth look into EFT. Users should also refer to the "Remote User's Guide" for the remote host in which a connection will be made for additional information about that host's environment.

Invoking EFT in UNIX

EFT is invoked using the following, general format:

```
eftip-client [input-file [argument1, argument2, ...]] [-keyword value]
or
eftntx-client [input-file [argument1, argument2, ...]] [-keyword value]
```

Where:

- eftip-client** or **eftntx-client** is the command to invoke EFT. (If you have the NetEx eFT product, use '**eftntx-client**'.) It is possible that this command may conflict with another UNIX command or symbol already set up at a particular site. If that is the case, setup an alias for whatever you want to use.
- input-file** is an optional EFT input or script file containing EFT commands that may be read and executed. When EFT completes execution of the input file the session terminates and the UNIX 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 EFT session. The following are valid keywords:
- BLOCK** specifies the size in bytes of the CONNECT block size (displayed as the "BLOCKsize" variable in "SHOW CONNECT"). The default value is 16384.
 - GLObal** specifies the size in bytes of the global variable environment. 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 EFT is invoked (taken from the shell environment variable "USER_HOME"). Changing this keyword's value redefines the location EFT uses to locate user startup files.
 - OUTput** specifies the name of an output file that is to receive the output from this session. This keyword is taken from the shell environment variable "OUTPUT".

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

EFT is invoked for interactive use by typing *eftip-client* or *eftntx-client* at the UNIX sysprompt:

```
$ eftip-client
EFT>
```

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

Local UNIX EFT Startup Files

When EFT is invoked, it attempts to read two startup files on the local host: a site startup file located in the EFT root or (SITE) directory called *sclient.ua*, and a user startup file located in the user’s login directory called *client.ua*. The site startup file is read first, and then the user startup file is read. Neither of the startup files is required.

The startup files consist of EFT commands. Typically a site administrator will create the site start-up 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 EFT aliases for displaying the local and remote directory listings, *ld* and *rd* respectively. It also sets the default EFT local prefix to be **MYHOST:**. After EFT 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 EFT input prompt appears.

It is possible to invoke EFT by declaring alternative startup files. This is done using the SEARCH qualifier on the command line when EFT is invoked. By default, SEARCH is defined as ‘(SITE) (USER)’. By implication, this reads *sclient.ua* from the local EFT (SITE) directory and then *client.ua* from the user’s login directory, 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 “UNIX EFT SEARCH Keywords (SITE), (USER), and (NONE)” on page 91 for more information.

Remote EFT Startup Files

In addition to the local startup files, there are equivalent remote startup files that the EFT Responder on the remote host returns to the local Initiator following a successful connection. By default, both a site and user startup are read, 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 an EFT 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.

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 EFT. This section will present some basic concepts that are an important foundation for understanding the details of EFT.

EFT Commands and Command Qualifiers

A EFT 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 99.

Several of the EFT 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 need not be concerned with overriding or redefining these values. The sophisticated user can use the qualifiers to modify commands, often making the commands more powerful for an application. There are two methods for changing the values of qualifiers:

1. A qualifier can be re-defined 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 and its new value.

Command qualifiers are similar to EFT 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 99. For instance, the minimum spelling of qualifier CREate is CRE.

Displaying the Valid Qualifiers for a Command

A list of valid qualifiers for a particular EFT 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:

```
EFT> show qualifier input
EFT:
EFT:  CONTInue ..... continue on error (on/off)
EFT:  ECHO ..... echo input to terminal (on/off)
EFT:  IGNore ..... ignore input failure due to missing file (on/off)
EFT:  PROMPT2 ..... secondary prompt for input continuation
EFT:  PROMpt ..... prompt string for USER input
EFT:  SEARch ..... search path for default INPUT commands
EFT:  VERify ..... verify string/alias substitution (on/off)
EFT:
EFT>
```

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:

```

zos5> show send
EFT:
EFT:  BLOCKsize ..... 3120
EFT:  CHEcksum ..... off
EFT:  COMPress ..... off
EFT:  CRC ..... off
EFT:  CREate ..... new
EFT:  DDname .....
EFT:  DElete_on_error ... off
EFT:  DEVICE_Wait .....
EFT: * DIRectory:LOCal ... /usr/share/nesi/eftip/si
EFT: * DIRectory:REMote .. TEST1
EFT:  DISPosition ..... CATALOG
EFT:  EXPIration .....
EFT:  EXPand ..... off
EFT:  FLOW ..... off
EFT:  LABELTYPE .....
EFT:  LABELnumber .....
EFT:  MAXRECORD ..... 10000
EFT:  METHod ..... lzw
EFT:  MODE ..... character
EFT:  PARTialrecord ..... on
EFT:  PDSE ..... off
EFT:  QUIet ..... off
EFT:  RECFORmat ..... VB
EFT:  RECLENgth ..... 251
EFT:  REFER .....
EFT:  RELEase ..... on
EFT:  RETAIN ..... off
EFT:  RETENTION .....
EFT:  RNT ..... off
EFT:  RNT_BUFAalloc ..... 262144
EFT:  RNT_INTerval ..... 60
EFT:  RNT_TIMEout ..... 1200
EFT:  SPACe ..... *16k
EFT:  STATistics ..... on
EFT:  TAB ..... 0
EFT: * TSOPREfix:REMote .. TEST1
EFT:  UNIT .....
EFT:  VOLume .....
EFT:  WRAP ..... on
EFT:
EFT: * Informational qualifier (cannot be modified).
EFT:

```

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:

```
zos5> show input prompt
EFT: PROMpt ..... {dfn(host:remote, host:remote, "EFT")} {"> "}
zos5>
```

Setting a Command Qualifier

Use the `SET` command to re-define the value of a qualifier for a command for the duration of the EFT 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:

```
zos5> set receive mode stream
```

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

```
zos5> show receive mode
EFT: 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:

```
zos5> set 'command' 'qualifier' on
```

or

```
zos5> set 'command' 'qualifier'
```

For the `INPUT` qualifier `ECHO`, this would be:

```
zos5> set input echo on
```

or

```
zos5> set input echo
```

For Boolean qualifiers, a missing value is interpreted by EFT 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' (2^{10}) or 'M' (2^{20}) multiplier. For example, to set the `CONNECT BLOCKSIZE` qualifier to 16 kilobytes, the following may be entered:

```
zos5> set connect blocksize 16k
zos5> show connect blocksize
EFT: BLOCKsize ..... 16384
```

Note that the value of any qualifier can be overridden by a qualifier specified on the command line.

Overriding a Command Qualifier

The qualifiers that can be defined with the `SET` command (noninformational 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:

```
EFT> 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:

```
EFT> show receive mode
EFT: 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.

```
EFT> send -quiet on sourcefile
EFT> send -quiet sourcefile
```

EFT interprets the missing Boolean value to be ON, even if the default value is OFF.

Online Help

Built into EFT 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 EFT help display, use the HELP command as follows:

```
EFT> 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 EFT command LOCAL. To get additional help on the LOCAL command, one would type:

```
EFT> help local
```

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

```
EFT> 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 99 for more details.

Controlling EFT Input and Output

The EFT commands INPUT and OUTPUT, along with their respective qualifiers, control a majority of the user-oriented input and output within EFT. By setting various qualifiers, users can change the EFT prompt, tell EFT 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 EFT I/O.

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

```
EFT> show input
EFT:
EFT:  CONTInue ..... off
EFT:  ECHO ..... on
EFT:  IGNore ..... off
EFT:  PROMPT2 ..... More>>
EFT:  PROMpt ..... {dfn(host:remote, host:remote, "EFT")}{"> "}
EFT:  SEArch .....
EFT:  VERify ..... off
EFT:
```

Each of these qualifiers is explained in detail in “Command Descriptions” on page 99 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 EFT prompt from **EFT>** to **MY-PROMPT:** type the following:

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

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

```
EFT> set input continue on
```

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

```
EFT> show output
EFT:
EFT:  COLumns ..... 94
EFT:  CREate ..... replace
EFT:  * DESTination .....
EFT:  FORmat ..... {msg("text")} ({msg("facility")}-{msg("code")}).
EFT:  HOLD ..... off
EFT:  LINES ..... 47
EFT:  PREFix ..... EFT:
EFT:  QUIet ..... off
EFT:  RESize ..... on
EFT:  TRUNcate ..... off
EFT:
EFT:  * Informational qualifier (cannot be modified).
EFT:
```

Each of these qualifiers is explained in detail in “Command Descriptions” on page 99 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 EFT to pause every 24 lines (the current value of the LINES qualifier), turn on the HOLD qualifier with the following command:

```
EFT> set output hold on
```

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

```
EFT> set output lines 20
```

The OUTPUT command itself can be used to capture the results of a EFT 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:

```
EFT> set input echo on
EFT> 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 “Advanced Local User’s Guide” on page 48 of this manual. This facility is particularly useful as a means of providing information to NESi’s technical support personnel regarding questions and problems.

EFT Error Messages

EFT provides a friendly user interface across many different host types. This includes error messages that are easy to understand. Error messages returned by EFT consist of at least a EFT 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:

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

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

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

```
zos5> send badfile
EFT: Failure during CHARACTER mode send (UA-5001).
EFT: Failed to access file '/usr/share/nesi/eftip/si/badfile' (EFT803-8302).
EFT: No such file or directory (UNIX-2).
zos5>
```

The first error code (UA-5001) indicates that this is a general EFT error (UA) with an error number of 5001. The second error code (UA803-8302) says the error is from EFT (UA), but generated by the EFT product number 803 (or more exactly eFT803). The actual error number is 8302. The last error code (UNIX-2) indicates the error is generated by the operating system (UNIX or whatever the operating system name might be), with the operating system error number of 2. The EFT error messages are listed in “Appendix F. EFT Error Messages for UNIX” on page 204 and in similar appendices in other EFT manuals. The general EFT 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., UA803 is product eFT803). Refer to the manuals for the operating system for system-specific 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 EFT 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,” 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 common set for a user or group of users. An alias is nothing more than a new name for a EFT command or set of commands. Aliases are useful for creating “shorthand” commands for complex or frequently used EFT command sequences. The simplest aliases are one for one translations of an alias name and a EFT 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.

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

Now, instead of typing HELP to obtain help information, the user can just type ‘?’ at the EFT prompt. The commands are considered equivalent by EFT. 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:

```
EFT> 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 “Developing EFT Scripts Using Input Files and Aliases” on page 76.

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

```
EFT> edit myfile
```

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

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

```
EFT> show alias edit
EFT: EDit ..... receive -mode character {1} edit.tmp
EFT:      local -interactive myeditor edit.tmp
EFT:      send -mode character -create replace edit.tmp {1}
EFT:      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 EFT input or script file, or within a site or user startup file which are read automatically when EFT is invoked. Refer to “Developing EFT Scripts Using Input Files and Aliases” on page 76 for a detailed description of aliasing.

Terminating an EFT Session

To end an interactive EFT session type EXIT:

```
EFT> exit
```

EXIT will disconnect all connections to remote hosts and terminate the current EFT 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 99 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 EFT Initiator on the local host and the EFT Responder on the remote host. There are two ways to make a connection to a remote host, the CONNECT command and the LOGIN alias.

Using CONNECT to Establish a Connection

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

| Command | Parameters |
|---------|-----------------------------------|
| CONnect | host userid password [parameters] |

Where:

- host** is the name of a remote host as defined in the local host's database (either in the local hosts file, accessible through DNS, or described in the NCT file for NetEx/IP environments).
- 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.
- parameters** indicates additional parameters that may be required by the remote host at login time.

Below is an example CONNECT where the host name is 'zos5', the userid is 'test1', and the password is 'test1':

```
EFT> connect zos5 test1 test1
EFT: Connected to service 'EFT' on host 'zos5'.
=====
ICH70001I TEST1      LAST ACCESS AT 18:26:31 ON TUESDAY, NOVEMBER 19, 2013
IKJ56455I TEST1 LOGON IN PROGRESS AT 18:38:59 ON NOVEMBER 19, 2013
IKJ56951I NO BROADCAST MESSAGES
READY
=====
EFT: Logged in as user 'test1'.
EFT:
EFT: Welcome to MVS eFT version 5.4.6
EFT:
zos5>
```

Following a successful CONNECT, EFT 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 EFT Responder (Service Initiator or service 'USER'). 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 EFT but may be to the user logging in. Next is an EFT message indicating that a successful login occurred. 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" describing the CONNECT command will also assist in the use of this command and its qualifiers.

NOTE: 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 alias 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 alias. LOGIN is 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:

```
EFT> login zos5
Username? test1
Password?
Qualifiers?
EFT: Connected to service 'EFT' on host 'zos5'.
=====
```

```
ICH70001I TEST1      LAST ACCESS AT 18:38:59 ON TUESDAY, NOVEMBER 19, 2013
IKJ56455I TEST1 LOGON IN PROGRESS AT 18:41:31 ON NOVEMBER 19, 2013
IKJ56951I NO BROADCAST MESSAGES
READY
```

```
=====
EFT: Logged in as user 'test1'.
EFT:
EFT: Welcome to MVS eFT version 5.4.6
EFT:
zos5>
```

Notice that LOGIN prompts the user for appropriate login information and that the password was not printed to the terminal. (Whenever possible EFT 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 EFT, 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 EFT 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 EFT Responder (the remote server) sends information about itself to the Initiator (the local client) and visa 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:

```
EFT> show local
EFT:
EFT: * COPYRight ..... Copyright (C) 1999-2013, Network Executive Software, Inc.
EFT: DIRectory ..... /home/test1
EFT: * GATEWay .....
EFT: HOMEdir ..... /home/test1
EFT: * HOSTCODE ..... ASCII7
EFT: * HOSTTYPE ..... UNIX
EFT: INTeRactive ..... off
EFT: * LicExp ..... 20141231
EFT: * LicKey ..... CXAJ-YAC2-ABAA-DTNT-CJ2Z-5GSL
EFT: * LicNotOper ..... 20150331
EFT: * NETWork ..... TCPIP
EFT: * PID ..... 13454
EFT: PRERFix ..... Unix:
EFT: * PRORduct ..... EFT803
EFT: QUIet ..... off
EFT: * ROORdir ..... /usr/share/nesi/eftip/user
EFT: SHELL ..... /bin/bash
EFT: * STATus .....
EFT: * USERname ..... test1
EFT: * VERsion ..... 5.4
EFT:
EFT: * Informational qualifier (cannot be modified).
EFT:
```

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:

```
zos5> show remote
EFT:
EFT: * BLOCKsize ..... 16384
EFT: * COPYRight ..... COPYRIGHT (c) 1999-2012 - Network Executive Software, Inc. Mpls. M
EFT: N
EFT: DIRectory ..... TEST1
EFT: * GATEway .....
EFT: HOMEdir ..... TEST1
EFT: * HOST ..... zos5
EFT: * HOSTCODE ..... EBCDIC
EFT: * HOSTENV ..... TSO FOREGROUND
EFT: * HOSTOS ..... z/OS 1.8
EFT: * HOSTTYPE ..... MVS
EFT: * LicExp ..... 20131231
EFT: * LicKey ..... CRJY-YAC2-AAA4-4CTW-VNHR-GDFV
EFT: * LicNotOper ..... 20140331
EFT: * PID ..... OXF592808DF4B0
EFT: PREFix ..... MVS:
EFT: * PRODUct ..... EFT213
EFT: QUIet ..... off
EFT: * ROOTdir ..... EFT.EFT1.TCP.TEXT
EFT: * SERvice ..... EFT
EFT: * STATus .....
EFT: * TRANSLate ..... Network
EFT: TSOPREfix ..... TEST1
EFT: * USERname ..... TEST1
EFT: * VERsion ..... 5.4.6
EFT:
EFT: * Informational qualifier (cannot be modified).
EFT:
zos5>
```

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

An EFT 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 'zos5'. This first connection can be verified by invoking the SHOW HOST command:

```
zos5> show host
EFT:
EFT: active --> (1) Host=zos5      User=TEST1
EFT:
zos5>
```

SHOW HOST 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 "Using LOGIN to Establish a Connection" on page 20. For example, to connect to a host named 'suse1', the following command sequence is used:


```

zos5> login suse1
Username? test2
Password?
Qualifiers?
EFT: Connected to Service Initiator on host 'suse1'.
=====
Last login: Mon Nov 18 16:17:18 CST 2013 from suse1 on pts/25
exec /usr/bin/ftip-server -si -service 0 -port_min 3100 -port_max 3150
=====
EFT: Logged in as user 'test2'.
EFT: Connected to service '3100' on host 'suse1'.
suse1>

```

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

```

suse1> show host
EFT:
EFT:          (1) Host=zos5      User=TEST1
EFT: active --> (2) Host=suse1   User=test2
EFT:
suse1>

```

Notice that 'suse1' 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 'zos5'. The SHOW REMOTE command also will display the remote environment for host 'suse1' since it is now active. The connection to host 'zos5' remains but is in an idle state. To make it the active connection, the SET HOST command is used as:

```

suse1> set host zos5
zos5>
or
suse1> set host 1
zos5>

```

Now a look at the host display will show that 'zos5' is the active host:

```

zos5> show host
EFT:
EFT: active --> (1) Host=zos5      User=TEST1
EFT:          (2) Host=suse1      User=test2
EFT:
zos5>

```

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:

```

zos5> disconnect
EFT: Disconnected from host zos5.
EFT>

```

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

```
EFT> show host
EFT:
EFT:          (2) Host=susel      User=test2
EFT:
```

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 'suse1' active:

```
EFT> set host suse1
```

SHOW HOST will now indicate the change:

```
susel> show host
EFT:
EFT: active --> (2) Host=susel      User=test2
EFT:
susel>
```

An alternative way to disconnect from an active host is to exit the EFT 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 EFT 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, EFT 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:

```
EFT> send src_spec
```

where **src_spec** is the name of an existing file on the local host. EFT takes care of mapping, the local file name to a valid remote file specification in all but a few instances. If EFT 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 the file `src_spec` and renames it `new_file` on the remote host:

```
EFT> 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 UNIX ” on page 40. 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 | <code>src_spec [dest_spec] [qualifiers]</code> |

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, EFT 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:

```
EFT> receive src_spec
```

where **src_spec** is the name of a file that currently resides on the remote host. In the same way as it handles SEND, EFT 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:

```
EFT> 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 UNIX ” on page 40 of this manual.

Send and Receive Qualifiers

EFT 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.

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.:

```
susel> show qualifier send
EFT:
EFT:  CHEcksum ..... file transfer 16-bit CHECKSUM (on/off)
EFT:  COMPress ..... compress the source data (on/off)
EFT:  CRC ..... file transfer 32-bit CRC (on/off)
EFT:  CREate ..... file create options: NEW REPlace APPend BACKup
EFT:  DELeTe ..... DELeTe
EFT:  DELeTe_on_error ... delete destination file if transfer error (on/off)
EFT:  EXPand ..... expand the destination data (on/off)
EFT:  FLOW ..... file transfer flow control (on/off)
EFT:  MAXRECORD ..... maximum record size (in bytes): 1-100000
EFT:  METHod ..... method used to compress/expand data: LZW RLE
EFT:  MODE ..... file transfer mode: CHAracter STream RECORD
EFT:  BACKup REStore COpy V1Char
EFT:  PARTialrecord ..... partial record mode (on/off)
EFT:  QUIet ..... inhibit file transfer displays (on/off)
EFT:  RNT ..... enable resilient network transfer (on/off)
EFT:  RNT_BUFAlloc ..... resilient network transfer buffer size: 1-1048576
EFT:  RNT_INTerval ..... resilient network transfer retry interval (seconds): 1-600
EFT:  RNT_TIMEout ..... resilient network transfer timeout (seconds): 1-32767
EFT:  SPACe ..... allocated file space (in bytes)
EFT:  TAB ..... tab expansion column spacing (0=off): 0-999
EFT:
susel>
```

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:

```
susel> show send
EFT:
EFT:  CHEcksum ..... off
EFT:  COMPress ..... off
EFT:  CRC ..... off
EFT:  CREate ..... replace
EFT:  DELeTe_on_error ... off
EFT:  * DIRectory:LOCAl ... /home/test1
EFT:  * DIRectory:REMote .. /mnt/home2/test2
EFT:  EXPand ..... off
EFT:  FLOW ..... off
EFT:  MAXRECORD ..... 10000
EFT:  METHod ..... lzw
EFT:  MODE ..... character
EFT:  PARTialrecord ..... on
EFT:  QUIet ..... off
EFT:  RNT ..... off
EFT:  RNT_BUFAlloc ..... 262144
EFT:  RNT_INTerval ..... 60
EFT:  RNT_TIMEout ..... 1200
```

```

EFT:  SPaCe ..... 0
EFT:  TAB ..... 0
EFT:
EFT:  * Informational qualifier (cannot be modified).
EFT:
suse1>

```

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 noninformational 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:

```
EFT> 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:

```
EFT> receive sourcefile -create replace
```

For a complete list of valid RECEIVE qualifiers, refer to the "File Handling Under UNIX" section on page 40 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 99 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

EFT 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 an EFT 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:

- qualifiers** represents optional REMOTE qualifiers that may be added to the command line to override the default values. Their default values are defined by the remote host.
- command** may be either a valid command on the remote host, an alias command defined using SET REMOTE ALIAS, or one of the predefined host independent commands.

EFT 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 EFT session by typing the following:

```

suse1> remote date
Unix: Wed Nov 20 15:07:33 CST 2013
suse1>

```

The results are displayed in the remote host's format with the exception of the optional host prefix that precedes each line of output (suse1>). 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, EFT defines a set of commands (implemented as aliases) that exist on all hosts.¹ 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:

```
suse1> show remote alias
EFT:
EFT: CANcel ..... /usr/bin/cancel
EFT: COPY ..... /bin/cp
EFT: DElete ..... /bin/rm
EFT: DIfference ..... /bin/diff
EFT: DIRectory ..... /bin/ls -al
EFT: HELp ..... /usr/bin/man
EFT: PRInt ..... /usr/bin/lp
EFT: QUEue ..... /usr/bin/lpstat -t
EFT: REName ..... /bin/mv
EFT: STatus ..... /bin/ps -ef
EFT: TYpe ..... /bin/cat
EFT: WHO ..... /usr/bin/who
EFT:
suse1>
```

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. EFT 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:

```
EFT> 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 'ls -la', an alternative to the above would be:

```
EFT> remote ls -la
```

The commands would give identical results since the host independent command DIRectory would be mapped to the native command 'ls -la' 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 UNIX Commands

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

| Command | Parameters |
|---------|------------------------|
| LOCAL | [qualifiers] [command] |

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

Where:

- qualifiers** represents optional LOCAL qualifiers that may be added to the command line to override the default values. Qualifiers must appear before the command parameter.
- command** can be a valid UNIX command, an alias command defined using SET LOCAL ALIAS, or one of the predefined host independent commands (e.g. DIRECTORY, TYPE, WHO, etc.). EFT performs translation on any alias prior to passing the command string to UNIX.

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

Note: Various UNIX distributions' utilities are slightly different. The examples in this manual will show the output from SUSE Linux Enterprise Server 10 utilities.

The following is an example of the LOCAL command being used within UNIX EFT to obtain a list of what is happening on the system. This command is 'ps -a':

```
susel> local ps -a
Unix:  PID TTY          TIME CMD
Unix:  7307 pts/14        00:00:00 script
Unix:  7308 pts/14        00:00:00 script
Unix: 11581 pts/12        00:00:00 bash
Unix: 12019 pts/11        00:00:00 svcinit
Unix: 16001 pts/9         00:00:00 tail
Unix: 16638 pts/2         00:00:00 eftip-client
Unix: 16988 pts/2         00:00:00 ps
Unix: 19828 pts/4         00:00:00 vim
Unix: 23654 pts/5         00:00:00 bash
Unix: 27090 pts/11        00:00:00 bash
susel>
```

The prefix Unix: indicates that the results are being returned from the UNIX host.³ Using LOCAL from within an EFT session, it is also possible to invoke a compiler, send user messages, execute script files, etc. For example, to send the file MOV.FIL to the printer, the BSD command⁴ would be:

```
EFT> local lpr mov.fil
```

To execute a local command under UNIX, EFT forks a shell sub-process then issues the command under it. Therefore, any command that is issued within the sub-process that changes the user's environment (e.g., local cd) will have no effect on the parent process or EFT. However, shell scripts can be executed that modify the sub-process environment and then issue commands making use of those changes. The most likely item to want to modify is the local directory default. EFT makes this possible with the SET LOCAL command:

```
EFT> set local directory /nsc/temp/sub
```

A display of the local directory will verify the change:

```
EFT> show local directory
EFT: DIRectory ..... /nsc/temp/sub
```

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

³ The AT&T UNIX equivalent is *ps -ef*

⁴ The AT&T equivalent is *lp mov.fil*.

```
EFT> local pwd
Unix: /nsc/temp/sub
```

Some UNIX 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 execute a program called *test* that prompts for a file name, the INTERACTIVE flag would be set as:

```
EFT> set local interactive on
```

The INTERACTIVE qualifier tells EFT to treat the terminal as standard input. Normally standard input is redirected to the NULL device.

The LOCAL command also gives the user the ability to enter an interactive UNIX session, keeping the EFT 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.

```
EFT> local
$
```

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

```
$ exit
EFT>
```

EXIT returns the user back to the EFT 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 EFT, establish a remote connection, and then return to UNIX for further activity. When a file or remote job is requested, the user simply returns to EFT where the remote host is actively waiting.

For more information on the LOCAL command and its qualifiers, refer to “Command Descriptions” on page 99 of this manual.

Issuing Local UNIX Host-Independent Commands

As on the remote host, the local EFT 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 UNIX EFT on BSD, issue the SHOW LOCAL ALIAS command:

```
EFT> show local alias
EFT:
EFT: CAnCel ..... /usr/ucb/lprm
EFT: COpy ..... /bin/cp
EFT: DElete ..... /bin/rm
EFT: DIfference ..... /bin/diff
EFT: DIrectory ..... /bin/ls -al
EFT: HELp ..... /usr/ucb/man
EFT: PRInt ..... /usr/ucb/lpr
EFT: QUEue ..... /usr/ucb/lpq
EFT: REName ..... /bin/mv
EFT: STatus ..... /bin/ps -xac
EFT: TYPE ..... /bin/cat
EFT: WHO ..... /bin/who
EFT:
```

For UNIX, the host-independent commands are:


```

EFT> show local alias
EFT:
EFT: CANCEL ..... /usr/bin/cancel
EFT: COPY ..... /bin/cp
EFT: DELETE ..... /bin/rm
EFT: DIFFERENCE ..... /bin/diff
EFT: DIRECTORY ..... /bin/ls -al
EFT: HELP ..... /usr/bin/man
EFT: PRINT ..... /usr/bin/lp
EFT: QUEUE ..... /usr/bin/lpstat -t
EFT: RENAME ..... /bin/mv
EFT: STATUS ..... /bin/ps -ef
EFT: TYPE ..... /bin/cat
EFT: WHO ..... /bin/who
EFT:

```

The host independent commands are in the left column and the UNIX command translations are in the right column. Users can issue host independent commands as if they were commands native to UNIX.

The following is an example of a LOCAL command that invokes a host independent command called TYPE. TYPE translates to the UNIX command *cat* which types out the contents of a file:

```

EFT> local type hello.file
Unix: * * * * *
Unix: H E L L O
Unix: * * * * *

```

Notice that the output from TYPE is equivalent to the output from *cat*:

```

EFT> local cat hello.file
Unix: * * * * *
Unix: H E L L O
Unix: * * * * *

```

Local UNIX EFT users can also create their own local aliases using the SET LOCAL ALIAS command. For example, to create a local alias called CURRENTDIR that shows the current default directory (i.e. *pwd*), issue the following command:

```

EFT> set local alias currentdir pwd

```

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

```

EFT> show local alias currentdir
EFT: CURRENTDIR ..... pwd

```

This new alias is equivalent to the UNIX command *pwd* 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 EFT sessions, edit them into a local EFT startup file.

Refer to “Command Descriptions” on page 99 of this manual for the list of valid LOCAL qualifiers and an example of their use. Also see the “Advanced Local User’s Guide” on page 48 for further discussion on host aliases and aliases in general.

Editing Remote Files with a UNIX Editor

Using a predefined alias called EDIT, users can invoke their favorite UNIX editor to edit files that reside on the remote host in which they are connected. The EDIT alias is typically defined in the EFT 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 a UNIX editor:

```

EFT> show alias edit
EFT: EDit ..... receive -mode char -cre repl {1} edit.tmp
EFT:                                ledit edit.tmp
EFT:                                ask -prom "Update remote (Yes/No)? " -def "Y" yn
EFT:                                set var S send -mode char -cre repl edit.tmp {1}
EFT:                                {cmp(yn,"Yes",S)}
EFT:                                local delete edit.tmp

```

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

```
EFT> edit rfile
```

In the example, the remote file *rfile* would automatically be transferred to the UNIX system and the local 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 over-writing 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 48 for more information on developing multicommand aliases.

Interrupting a Command within UNIX EFT

To interrupt or terminate a command from executing once it has started, the standard interrupt sequence should be invoked. For UNIX users, this is typically defined to be either the DELEte key or the <CTRL>-<C> key sequence (hold the <CTRL> key (Control) and press the C key. Since UNIX allows the interrupt sequence to be redefined, it is up to the user to know its definition. Issuing an interrupt from the keyboard will cause any EFT, 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 interrupt will result in the EFT 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 48.)

To terminate a EFT session that appears to have hung for some reason, hit the interrupt sequence three times in a row. Three consecutive interrupts causes EFT to cleanup and exit.

Changing the Default UNIX SHELL

EFT allows the user to select which SHELL to invoke when executing local commands on a UNIX system. For example, to set the SHELL qualifier on a local UNIX host to */bin/csh*, one would type:

```
EFT> set local shell /bin/csh
```

The SHELL qualifier could be changed to the standard (Bourne) shell, 'C' shell, or any third party shell. For example, to tell EFT to invoke myshell when executing LOCAL commands, type:

```
EFT> set local shell myshell
```

The shell, *myshell*, as defined, would always be taken from the user's working directory. Be sure that a sufficient path is given to the file or an error will result.

UNIX Remote User's Guide

This section is intended for users who are currently running the EFT Initiator from any local host, regardless of operating system, and would like to establish a network connection into a UNIX host. The information provided here and in "File Handling Under UNIX" on page 40 should be enough to help a non-UNIX user start being productive in a very short period of time. UNIX users should also reference these sections to become comfortable with how EFT operates in the UNIX environment.

Connecting into a UNIX Host

A EFT network connection is established into a UNIX environment by means of a process known as the Service Initiator. The task of the Service Initiator is to process UNIX login attempts from remote EFT users (Initiators) and, if successful, start-up a server process (Responder) that will then communicate with the remote user for the duration of the EFT 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 pass through the UNIX login utility to ensure that proper security is checked. 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 UNIX EFT

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. None of the CONNECT qualifiers are of special importance to UNIX.

The following CONNECT qualifiers are not used by UNIX EFT at connect/login time:

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

Note: Your site may have customized UNIX eFT to use one or more of these additional qualifiers.

The CONNECT command as described in "Command Descriptions" on page 99 lists all applicable CONNECT qualifiers.

Remote UNIX EFT Startup Files

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

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

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

- (NONE) do not process any responder (or server) startup files.
- (SITE) implies a EFT site startup file called *sserver.ua* located in the remote UNIX site directory. This is explained in more detail in “UNIX EFT SEARCH Keywords (SITE), (USER), and (NONE)” on page 91.
- (USER) implies a EFT user-level startup file called *server.ua* that is located in the user’s login directory. This is explained in more detail in “UNIX EFT SEARCH Keywords (SITE), (USER), and (NONE)” on page 91.

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

```
CONNECT
DISCONNECT
LOCAL
RECEIVE
REMOTE
SEND
```

Transferring Files to a UNIX Host

The file transfer capabilities of EFT are provided by two commands, SEND and RECEIVE. The SEND command provides file transfer from the local host to a remote UNIX host. The RECEIVE command transfers files from a remote UNIX 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 UNIX system, refer to “File Handling Under UNIX ” on page 40.

Executing Remote UNIX Commands

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

The following is an example of the REMOTE command being used within UNIX EFT to check the status of the current remote UNIX system. The BSD command is *ps -xac*⁵:

```
EFT> remote ps -xac
Unix:  PID  TT  STAT  TIME  COMMAND
Unix:   52  ?   I     0:04  sendmail
Unix:   58  ?   I     0:12  svcinit
Unix:   59  ?   I     0:03  bfxjs
Unix:  168  co  IV    0:02  csh
Unix: 3918  lp  S     0:00  user
Unix: 3922  lp  R     0:08  ps
```

⁵ The AT&T command is *ps -ef*.

```
Unix:      882  p4  IW      0:01  client
```

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

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

To execute a remote command under UNIX, EFT forks a `SHELL` subprocess then issues the command under it. Therefore, any command that is issued within the sub-process that changes the user's environment will have no effect on the parent process or EFT. However, shell scripts can often be executed that modify the sub-process environment and then issue commands making use of those changes.

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

```
EFT> set remote directory /root/remdir/two
```

A display of the remote directory will verify the change:

```
EFT:> show remote directory
EFT: DIRectory ..... /root/remdir/two
```

The new directory value will be used as the default directory for all subsequent `REMOTE` commands since EFT makes this change to the parent process, not a sub-process. Therefore, the UNIX command `pwd` results in the following value:

```
EFT> remote pwd
EFT: /root/remdir/two
```

For more information on the `REMOTE` command, refer to "Command Descriptions" on page 99 of this manual.

Issuing Remote UNIX Host Independent Commands

Although executing UNIX commands from a remote system may be very useful, many times remote users are not familiar with the UNIX command set. Therefore, EFT 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. To display the list of host independent commands defined for BSD UNIX EFT, issue the `SHOW REMOTE ALIAS` command:

```
EFT> show remote alias
EFT:
EFT: CANcel ..... /usr/ucb/lprm
EFT: COPY ..... /bin/cp
EFT: DELeTe ..... /bin/rm
EFT: DIFFerence ..... /bin/diff
EFT: DIRectory ..... /bin/ls -al
EFT: HELP ..... /usr/ucb/man
EFT: PRInt ..... /usr/ucb/lpr
EFT: QUEue ..... /usr/ucb/lpq
EFT: REName ..... /bin/mv
EFT: STatus ..... /bin/ps -xac
EFT: TYPE ..... /bin/cat
```

```
EFT: WHO ..... /bin/who
EFT:
```

For the equivalent UNIX commands, the results would be as follows:

```
EFT> show remote alias
EFT:
EFT: CANcel ..... /usr/bin/cancel
EFT: COpy ..... /bin/cp
EFT: DElete ..... /bin/rm
EFT: DIfference ..... /bin/diff
EFT: DIRectory ..... /bin/ls -al
EFT: HELp ..... /usr/bin/man
EFT: PRInt ..... /usr/bin/lp
EFT: QUEue ..... /usr/bin/lpstat -t
EFT: REName ..... /bin/mv
EFT: STATus ..... /bin/ps -ef
EFT: TYPE ..... /bin/cat
EFT: WHO ..... /bin/who
EFT:
```

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

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

- CANcel** cancels a specified job entry in the default UNIX print queue. Use the QUEUE alias to display the queue and determine the job entry number. The format is:
 CANCEL entry
- COpy** copy a UNIX file to another file name or UNIX directory. The format is:
 COPY file_spec1 file_spec2
- DElete** delete a file or set of files on the UNIX 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 UNIX files and list their differences. The format is:
 DIFFERENCE file_spec1 file_spec2
- DIRectory** display a listing of all the files in the given UNIX directory. The wildcard character '*' can be used to list only select files if desired. The format is:
 DIRECTORY [directory_name | file_spec]
- HELp** obtain help on a UNIX topic. The format is:
 HELP [unix-topic]
- PRInt** print a UNIX file or set of files to a UNIX printer. The format is:
 PRINT file_spec
- QUEue** display the current entries in the default UNIX print queue. The format is:
 QUEUE

REName chance the name of the specified UNIX 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 UNIX host. No parameters are required. The format is:

```
STATUS
```

SUBmit there is no equivalent UNIX command for SUBMIT

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

```
TYPE file_spec
```

WHO display a listing of users that are currently logged on the UNIX 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 directly to the UNIX command *who* which displays a list of user's currently logged onto the system:

```
EFT> remote who
Unix:  admin   console   June 1   05:32
Unix:  guest   ttya      June 7   06:18
Unix:  sam     ttypl     June 1   16:69
```

The output from WHO is equivalent to the output that would be seen from the UNIX command *who*.

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 UNIX directory listing (i.e. *ls -al*), issue the following command:

```
EFT> set remote alias listfiles ls -al
```

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

```
EFT> show remote alias listfiles
EFT: LISTFILES ..... ls -al
```

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 EFT sessions, edit them into a remote EFT startup file.

Refer to "Advanced Local User's Guide" on page 48 for further discussion on host aliases and aliases in general.

File Handling Under UNIX EFT

This section is intended to address UNIX file handling issues as they relate to EFT file transfer commands SEND and RECEIVE. Both local and remote UNIX EFT users should use this section as a reference for transferring files to and from a UNIX 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. The destination file is the new 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 for the most part tied directly to the Destination Host since that is where files get created.

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

UNIX 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 UNIX system is the Destination Host. That is, when the local host is a UNIX system, the qualifiers listed below pertain to the RECEIVE command (local UNIX is the Destination Host). When the remote host is a UNIX system, the qualifiers listed below are valid for the SEND command (remote UNIX is the Destination Host).

- CHEcksum** (BOOLEAN) this optional qualifier indicates whether or not eFT should perform a checksum as part of the file transfer. This value should be set to either ON or OFF. When CHECKSUM is enabled, a 16-bit CHECKSUM is calculated by the sender along with a block sequence number. These are verified by the receiver. If the CRC qualifier is also ON, only the CRC will be performed. The default for this value is OFF.
- COMPRESS** (BOOLEAN) this optional qualifier indicates whether or not EFT should perform compression as part of the file transfer. This value should be set to either ON or OFF. When COMPRESS is enabled, the data is compressed by the sender. The compression can be done by differing methods specified by the file transfer qualifier -METHOD (specified by the sender). The default for this value is OFF. For more information on the compression algorithms, refer to “EFT Data Compression” on page 90.
- CRC** (BOOLEAN) indicates whether or not EFT should perform a 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 UNIX 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 UNIX, the “normal” thing to do is replace an existing file if a file by the same name currently exists. Therefore, the default CREATE option on UNIX is REPLACE. Other options are APPEND, BACKUP,

DELETE, and NEW. NEW returns an error if a file by the same name already exists. BACKUP renames an existing file with a '.bak' extension and creates a new file. APPEND appends the source file to the destination file if it exists, or creates a new file if it does not. DELETE is identical to REPLACE, except that it deletes an existing file by the same name instead overwriting it, and then creates a new one.

- DELEte_on_error** (BOOLEAN) tells eFT how to respond if an error is encountered during the file transfer. This value should be set to either ON or OFF. ON tells eFT to delete the destination file if an error is encountered during the file encountered during the file transfer so a transfer so a partial file not left around. The default is OFF.
- EXPand** (BOOLEAN) this optional qualifier indicates whether or not EFT should perform data expansion (decompression) as part of the file transfer. This value should be set to either ON or OFF. When EXPAND is enabled, the data is decompressed by the receiver. There are differing methods of decompression and the receiver will automatically use the same method as was used to compress the data. If the file was not compressed and this qualifier is set ON, EFT will not decompress the file while returning an informational message. The default is OFF. For more information on the compression algorithms, refer to "EFT Data Compression" on page 90.
- FLOW** (BOOLEAN) indicates whether or not EFT should enable file transfer flow control. When FLOW is on, every 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 unnecessary transfers causing retry conditions when there are delays at the remote host (or unusual read timeouts for NetEx/IP environments) 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 EFT receiver, a significant penalty in performance is paid when FLOW is enabled.
- 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.
- METHod** (STRING) this optional qualifier specifies the data compression method that EFT should use to compress the data when the -COMPRESS qualifier is ON. This value can be set to either LZW or RLE. LZW specifies Lempel-Ziv-Welch compression, and RLE specifies the run-length-encoding method. The default is method is LZW. For more information on the compression algorithms, refer to "EFT Data Compression" on page 90.
- MODE** (STRING) the current file transfer mode. EFT for UNIX supports the following modes: CHARACTER, STREAM, RECORD, BACKUP, RESTORE, and COPY. The value of MODE is used internally by EFT 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. For more information, refer to "Transfer Modes Supported Under UNIX" on page 45.
- PARTialrecord** (BOOLEAN) with PARTialrecord enabled, records of length greater than the 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.

- RNT** (BOOLEAN) indicates whether or not the Resilient Network Transfer option is enabled for this file transfer. This option provides the ability for the sender to resume a file transfer operation from the point of interruption. If RNT is enabled, the network connection between an EFT client/server pair is re-established following a network outage (e.g. a dropped connection, a checksum or CRC failure, etc.), and the file transfer operation is resumed. This value should be set to either ON or OFF. The default is OFF.
- RNT_BUFalloc** (INTEGER) specifies the size of the RNT buffer. This parameter indicates the number of bytes retained by the sending host, in order to reestablish a network connection and restart a file transfer from the point of interruption. The default is 262144 bytes.
- RNT_INTerval** (INTEGER) specifies the number of seconds between attempts to re-establish the EFT session during RNT processing. The default is 60 seconds.
- RNT_TIMEout** (INTEGER) specifies the number of seconds to elapse before abandoning attempts to re-establish the EFT session during RNT processing. The default is 1200 seconds.
- SPACE** (INTEGER) the number of bytes which EFT 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 (*), EFT 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 EFT 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.
- TAB** (BOOLEAN) used to expand tabs to blanks. A non-zero value specifies the tab stop column width. This qualifier is only used during CHARACTER mode transfers. The default for this qualifier is 0, which disables tab expansion.

Definition of DIRECTORY Under UNIX EFT

When EFT is invoked from either the Initiator or Responder side, the session begins with a default definition for the DIRECTORY qualifier for both LOCAL and REMOTE. The definition of this qualifier under UNIX is the user's current default 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 EFT.

To change the default value of DIRECTORY, the EFT commands SET LOCAL DIRECTORY and SET REMOTE DIRECTORY are used. For example, if the remote host is a UNIX system and the current value of the remote DIRECTORY is /root/my/oldfiles, a new value of /sys/guest/newfiles can be established as:

```
EFT> set remote directory /sys/guest/newfiles
```

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

```
EFT> show remote directory
EFT: DIRectory ..... /sys/guest/newfiles
```

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

UNIX File Specifications

Below is a very brief discussion of file specification syntax on UNIX operating systems. This is provided as an aid to the occasional UNIX user who is interested in transferring files to or from a UNIX system yet is unsure of file specification syntax.

A UNIX file specification is called a pathname. A pathname describes the path UNIX takes to get from a starting point to a filename. A pathname begins with the root directory (referred to as '/') and includes every directory name between the starting point and the file name. Slashes separate the names within a pathname specification.

A file specification under UNIX has the format:

```
/DIR/DIR/ ... /NAME.EXT
```

Where:

DIR is a directory name that can consist of multiple subdirectories each separated by a slash (/).

NAME the file name portion consisting of alphanumerics, underscore (_), or period (.).

EXT the file extension in EFT is the portion of the file specification following the last period. EFT distinguishes extensions consisting of alphanumerics and underscore. This portion generally gives indication of what type of data is in the file.

An example UNIX pathname is:

```
/usr/smith/doc/manual.specs
```

The slash (/) at the beginning of the pathname refers to the root directory. The system begins its search here. Next, it looks for a file system (think of it as a directory) called 'usr'. Once found, UNIX searches for a directory called 'smith' within the file system. It then searches for subdirectory 'doc'. From this it can locate the file 'manual.specs'.

A typical EFT user would set his UNIX default directory, 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 and extension portion of the pathname in most cases. Also note that UNIX file names are case sensitive. This means that 'file' and 'File' are not equivalent names.

UNIX File Specification Examples

Some example UNIX file specifications are listed below to help non-UNIX users set a better understanding of their appearance.

```
/usr/barry/com/pag  
/unix/smith/sources/saved/file.c  
/myprog.file.o  
/etc/x  
/ETC/X  
/giant/y_man_doc/z/abcfile.text
```

When the path portion is missing, the current location is used as the default.

File Transfer Examples from a Local UNIX Host

Example 1

To send the file *alpha.for* from the current default local directory (/nsc/smith), to a remote host, the following command would be issued:

```
EFT> send alpha.for
EFT: SOURCE                DESTINATION                SIZE
EFT: -----                -----                -----
EFT: /nsc/smith/alpha.for   alpha.for                   54909
```

Notice the entire source filename is displayed. 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.exe* from */mary/joe/exe*, to the remote host with the new name of *test1.sav*, issue the following.

```
EFT> send /mary/joe/exe/test1.exe test1.sav -mode stream
EFT: SOURCE                DESTINATION                SIZE
EFT: -----                -----                -----
EFT: /mary/joe/exe/test1.exe test1.sav                   228512
```

The MODE qualifier was set to STREAM because it was known that the file being transferred was a non-record oriented binary file.

File Transfer Examples to a Remote UNIX Host

Example 1

To send a local file called *myfile.tmp* from a non-UNIX system to a Unix host, forcing a CRC on the transfer, the following command is issued:

```
EFT> send myfile.tmp -crc
EFT: SOURCE                DESTINATION                SIZE
EFT: -----                -----                -----
EFT: myfile.tmp            /jones/tmp/myfile.tmp      1922
```

Example 2

To receive a file called *login.txt* from a remote UNIX host, issue the command:

```
EFT> receive login.txt
EFT: SOURCE                DESTINATION                SIZE
EFT: -----                -----                -----
EFT: /sys/jones/login.txt   login.txt                   2046
```

Source Wildcard Support for UNIX File Transfers

Wildcarding is valid on the source file specification for both the SEND and RECEIVE commands. Two EFT wildcard characters have been defined in an attempt to standardize the wildcarding for all hosts which can support it. These are:

- * matches zero or more characters. For example, **def* matches the strings *abcdef*, *cdef*, *def*.
- ? matches exactly one character. For example, *?def* matches the strings *adef*, *bdef*, *cdef*, but does not match *abcdef* or *def*.

In addition to the EFT wildcard characters, one can also make use of the UNIX wildcarding capabilities where the two do not conflict.

An example use of source wildcarding for UNIX EFT appears below. The example demonstrates sending all the files in the default directory that start with the letter B and have exactly three character extensions:

```
EFT> send b*.*??
EFT: SOURCE          DESTINATION          SIZE
EFT: -----
EFT: /tmp/black.ftn   black.ftn           82293
EFT: /tmp/blue.sam   blue.sam            167461
EFT: /tmp/brown.doc  brown.doc           4365
```

Destination Wildcard Support for UNIX File Transfers

Destination wildcarding is also available on UNIX EFT. Destination wildcarding makes it possible to transfer a set of files from one system to a UNIX host, modifying the file names as part of the process. The single character ‘*’ is used to make this happen.

When a ‘*’ is seen as part of the destination file specification, EFT 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 UNIX host, renaming the files with an extension of FOR, destination wildcarding would be used as:

```
EFT> send *.ftn *.for
EFT: SOURCE          DESTINATION          SIZE
EFT: -----
EFT: abcdef.ftn      /temp/abcdef.for     33114
EFT: sample.ftn      /temp/sample.for     67261
EFT: test.ftn        /temp/test.for       4277
```

In this example, EFT 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:

```
EFT> send *.ftn x*x.*sav
EFT: SOURCE          DESTINATION          SIZE
EFT: -----
EFT: abcdef.ftn      /temp/xabcdefx.ftnsav 33114
EFT: sample.ftn      /temp/xsamplex.ftnsav 67261
EFT: test.ftn        /tempfxtestx.ftnsav  4277
```

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

Transfer Modes Supported Under UNIX EFT

UNIX EFT supports seven modes of file transfer: BACKUP, CHARACTER, COPY, RECORD, RESTORE, and STREAM. 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 UNIX.

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

BACKUP mode is designed to allow UNIX files to be backed up on 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 under EFT for UNIX is more advanced than in other systems; it saves full file characteristics as part of the backup header.

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. An error will generally result if an attempt to transfer binary data is made.

COPY mode is designed for peer to peer file transfers. In COPY mode, UNIX files can be moved from one UNIX system to another very efficiently. EFT keeps track of all file characteristics and restores them on file creation. Any type of file (text or binary) can be transferred very fast in COPY mode since file access is done as efficiently as possible, without individual records having to be manipulated. The data in COPY mode is transferred as an unstructured stream of bytes.

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.

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. 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.

Advanced Local User's Guide

Introduction

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

The majority of this section discusses how to develop a custom EFT 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 EFT batch jobs.

Special Characters

Several characters have special meaning to EFT when it is parsing a command line. The position of the character within a line is a determining factor on how EFT 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, EFT ignores the line. (An alternate comment character is the '#'). Comments are generally used within EFT alias definitions and input files to make them more readable for the user. The following are example EFT comment lines:

```
EFT> * This is a comment and is ignored.
```

- # This is identical to the '*' character as described above.

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

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

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

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

```
EFT> send -quiet source destination
```

To tell EFT 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 EFT 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:

```
EFT> 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 EFT to take them literally and skip any string processing that would normally be done. For example:

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

Finally, the exclamation point is used to tell EFT not to do any alias processing on a given command. Since an alias may have the same name as a EFT command, an exclamation point immediately preceding a command tells EFT 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, EFT uses the command as it appears without processing it as an alias. Each of the following lines in the example below tell EFT to use the command even if an alias by the same name has been defined

```
EFT> !text ignore alias processing on text
EFT> local !dir
EFT> 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:

```
EFT> text Your name is {name}
```

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

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

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

```
EFT> 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:

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

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

EFT String Substitution

EFT string substitution gives users the ability to write complex aliases and input scripts. String substitution can take place anywhere within a EFT 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 EFT TEXT command produces the following results:

```
EFT> text {" "}.
EFT: .
```

```
EFT> text {"Box"}.
EFT: Box.
```

```
EFT> text {" Big Box "}.
EFT: Big Box .
```

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

```
EFT> text {hostname}.
EFT: BLUESKY.
```

```
EFT> text {a}.
EFT: Sample string.
```

```
EFT> text {day}.
EFT> 28.
```

String function refers to one of the EFT defined functions that may accept 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:

```
EFT> text {date()}.
EFT: Sun Apr 7, 2013.
```

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

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

Although the TEXT command was used in all of the examples above, string substitution can be performed anywhere within a EFT command line, whether it is part of another EFT 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 EFT command.

String Variables

EFT variable names can be from one to twenty alphanumeric characters long, including underscores or 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 reference 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 EFT 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 EFT internal storage and can eventually lead to a “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:

```
EFT> set variable username smith
```

and,

```
EFT> 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:

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

and,

```
EFT> show global days
EFT: 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:

```
EFT> set variable username
```

and,

```
EFT> 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 EFT 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:

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

The real significance of string variables is the ability to use them within EFT aliases and input scripts. To reference the value of a variable, enclose the variable name in braces within a EFT 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:

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

and,

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

The braces around the variable name tell EFT to replace it with its assigned value.

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

```
EFT> set variable hostname alpha
EFT> 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 EFT 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:

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

By default, when a variable is enclosed in braces (without the ‘:global’ syntax), EFT looks for a local variable by that name. If one is found, its value is returned. If one is not found, EFT 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 EFT not to look for a local variable but instead look immediately for a global variable of that name.

EFT carries this special syntax one step further in allowing the substitution of EFT 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:

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

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

```
EFT> 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 75). The following command produces an equivalent result:

```
EFT> 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 EFT input buffer.

Below is an example of a string literal used with string substitution and the string function LOWER() (described on page 69) to define the EFT prompt as `command?` with two leading and two trailing spaces.

```
EFT> 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:

```
EFT> 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:

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

String Functions

EFT 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 either a numeric constant, 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
```

EFT 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 EFT scripts (input files or aliases). Often it is desirable to perform a particular EFT comment 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 EFT 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 EFT 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 EFT strings and string variables as described in the section entitled “String Variables” on page 50. Table 2 on page 54 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. | 56 |
| CHR | Returns a single character represented by the specified number in the local host machine’s native character set (ASCII or EBCDIC). | 57 |
| CMP | Compares two strings. Allows for partial string match by specifying the required characters in upper case. | 58 |
| DATE | Returns the system date of the local host | 59 |
| DEC | Subtracts one from a numeric string expression and returns the result. | 60 |
| DFN | Tests if a variable is defined. | 61 |
| DIV | Divides the first numeric string expression specified by the second and returns the result. | 56 |
| ENCRYPT | Takes a user password, encrypts that password and returns an alphanumeric string that represents the encrypted password. | 62 |
| 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. | 63 |
| EQ | Tests if the first number specified is equal to the second number specified. | 68 |
| EQS | Tests if the first string specified is equal to the second string specified. | 64 |
| EXT | Extracts and returns a bounded sequence of characters from a string. | 65 |
| GE | Tests if the first number specified is greater than or equal to the second. | 68 |
| GT | Tests if the first number specified is greater than the second | 68 |
| INC | Adds one to a numeric string expression and returns the result. | 60 |
| INDEX | Returns the position of the second string specified within the first string. The function returns zero if the second string is not found. | 66 |

| Table 2. List of Functions | | |
|-----------------------------------|---|-------------|
| Function | Description | Page |
| LE | Tests if the first number specified is less than or equal to the second. | 68 |
| LEN | Returns the count of characters that make up the specified string. | 67 |
| LOWER | Returns the lower case equivalent of a string expression, all characters but upper case are left untouched. | 69 |
| LT | Tests if the first number specified is less than the second. | 68 |
| MOD | Returns the remainder of the division of the first numeric string expression specified by the second. | 56 |
| MUL | Multiplies two numeric string expressions and returns the result. | 56 |
| MSG | Returns the information of the last message. MSG is most often used to tailor the EFT output message format. | 70 |
| NDF | Tests if a variable is not defined. | 61 |
| NE | Tests if the first number specified is not equal to the second. | 68 |
| NES | Tests if the first string specified is not equal to the second. | 64 |
| 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. | 71 |
| SLEEP | Causes EFT to pause or “sleep” for a specified number of seconds. This process is not interruptible. This function results in a null string. | 72 |
| STATUS | Returns the single status character of the previous command: S = Success, E = Error. | 73 |
| SUB | Subtracts the second numeric string expression specified from the first and returns the result. | 56 |
| TIME | Returns the system time of the local host. | 74 |
| UPPER | Returns the upper case equivalent of a string expression, all characters but lower case are left untouched. | 69 |

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:

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

Ask the user to enter three numbers:

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

Find the square of the first number:

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

Find the total sum of the three numbers:

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

Divide the third number by the first number:

```
EFT> text {num3}/{num1} = {div(num3,num1)}
EFT: 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 using ASCII:

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

To display the quote character on an EBCDIC host:

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

CMP Function

The CMP function compares two strings. 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 *key* argument.

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”:

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

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

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

0 = WWW MMM DD, YYYY

1 = YYMMDD

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

Examples: Display the system date:

```
User> text Today is {date(0)}  
User: Today is Thu Mar 14, 2013
```

Display the system date in YearMonthDay format:

```
User> text Today is {dated(1)}  
User: Today is 130314
```

DEC and INC Functions

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

INC add 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”:

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

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

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

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

If COUNT was defined, EFT would respond with YES.

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

```
EFT> 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":

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

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.

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

Execute the alias with no parameters:

```
EFT> print
EFT: no parameter
```

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

```
EFT> print this text
EFT: this
```

ENCRYPT Function

The purpose of this function is to encrypt host passwords which later will be used by EFT to establish host connections. This approach eliminates the security risk of having readable (clear-text) passwords stored in files. For additional usage information, please refer to “ENCRYPT Alias Command” page 154.

Format:

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

Where:

- password** Specifies the case-sensitive 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 EFT CONNECT commands.
- username** Optionally specifies the case-sensitive username associated with the local EFT process that will issue the CONNECT command. **This is optional. Is omitted the encrypted password can be used by anyone.** This username is used as a secondary encryption key for the specified password. When EFT is later run it queries the operating system for the username running the current process. EFT then uses this username as one of its keys in decrypting the password. A value of '*' (single asterisk) tells the EFT ENCRYPT function to use the current username running the EFT process as the secondary key. Note that the *effective* username is used on UNIX. You must be running as the same user which will later run EFT to issue the CONNECT command. **The value for “username” above must be entered in uppercase in order to match the username value later returned by MVS/zOS/Tandem and VMS.**

Example: Encrypt the password ‘COBRA’ using the UNIX username ‘myers’ as the local username for secondary encryption. Use the EFT TEXT command to display the encrypted results:

```
User> text {encrypt("COBRA", "myers")}  
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 specified variable is not defined, a null string is returned.

Format:

| |
|----------------------------|
| <code>env(variable)</code> |
|----------------------------|

Where:

variable a local host environment variable. This variable may be upper/lower case sensitive. The definition of 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:

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

To return the host environment variable “HOME”:

```
EFT> text {env("HOME")}  
EFT: SYS$SYSDEVICE:[ROOT]
```

EQS and NES Functions

The EQS function tests if the first string specified is equal to the second. The NES function tests if the first string specified is not equal to the second.

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:

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

In this example, since the username specified was not “guest”, the user was prompted for a password.

Now, set the Username variable “usr” to the text “guest”. Then, if the Username is *not* “guest”, ask for a Password:

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

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”, set the password to “netex”:

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

EXT Function

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

Format:

```
ext(string, number1, number 2)
```

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.

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

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

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

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

Display only the Hours and Minutes of the system time:

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

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

```
EFT> set input prompt {} User {ext(version:local,5,9)}>  
User 5.1.0>
```

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:

| |
|--------------------------------------|
| <code>index(string1, string2)</code> |
|--------------------------------------|

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

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

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

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

```
EFT> text {index("JanFebMarAprMayJunJlyAugSepOctNovDec","Mar")}
EFT: 7
```

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

```
EFT> text {index("abcdef", "cat")}
EFT: 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”:

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

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

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

```
EFT> text Year: {ext(date(),sub(len(date()),3),len(date()))}  
EFT: Year: 2013
```

A simpler form of the example above is:

```
EFT> text Year: {ext(date(),-3,0)}  
EFT: Year: 2013
```

Logical Operations

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

- EQ** test if the first number specified is equal to the second.
- NE** test if the first number specified is not equal to the second.
- 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:

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

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

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

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

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

LOWER and UPPER Functions

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

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

Format:

```
upper(string)
lower(string)
```

Where:

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

Examples: Display the lower case equivalent of “ABCdef”:

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

Display the upper case equivalent of “ABCdef”:

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

Display the system date in lower case:

```
EFT> text Today is: {lower(date())}
EFT: Today is: thu mar 14, 2013
```

To display the Weekday in upper case:

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

To set a predefined EFT variable “password” to its lower case equivalent:

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

MSG Function

The MSG function returns the information of the last message. MSG is most often used to tailor the EFT 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 (in NetEx only environments), UA, EFTxxx (where xxx represents the host product code), SIxxx, 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 NETEX 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 (in NetEx only environments), UA, EFTxxx (where xxx represents the host product code), SIxxx, or the operating system mnemonic for the host generating the error. The default is UA.

Examples: Display the facility of the last error message:

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

Display the text of the last error message from EFT:

```
EFT> text The Last Error Message was: {msg("t")}
```

Display the code of the last error message from EFT:

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

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

```
EFT> 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:

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

Execute the alias with no parameters:

```
EFT> print
```

```
EFT:
```

Now execute the alias with the parameters: this text string

```
EFT> print this text string
```

```
EFT: text string
```

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

```
EFT> print this "is a" test
```

```
EFT: "is a" test
```

SLEEP Function

The SLEEP function causes EFT 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 EFT session for ten seconds:

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

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

```
EFT> 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 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 a previous failure. If any of these commands fail, an error status is set. A status return specified by EXIT or QUIT (e.g., EXIT ERROR) will override the previous command status.

Format:

| |
|----------|
| status() |
|----------|

Examples: To display the output of the status function:

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

To display the status of the last command:

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

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

```
EFT> 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:

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

TIME Function

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

Format:

| |
|--------------------------|
| <code>time([num])</code> |
|--------------------------|

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. Default is 0. Specifying a value other than 0 or 1 will return a null value.

Examples: Output the time without the colon separator:

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

Output the time:

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

Redefine the input prompt to prompt with the system time:

```
EFT> 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 EFT sees `{string}` on a command line, it immediately tries to perform string substitution on `string`. To tell EFT 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:

```
EFT> set alias put send {} {sourcefile} {dfile}
```

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

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

If the empty `{}` had not been included during the definition above, EFT 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'):

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

The resulting alias definition looks like:

```
EFT> show alias put
EFT: 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 3 is a representation of nested substitution.

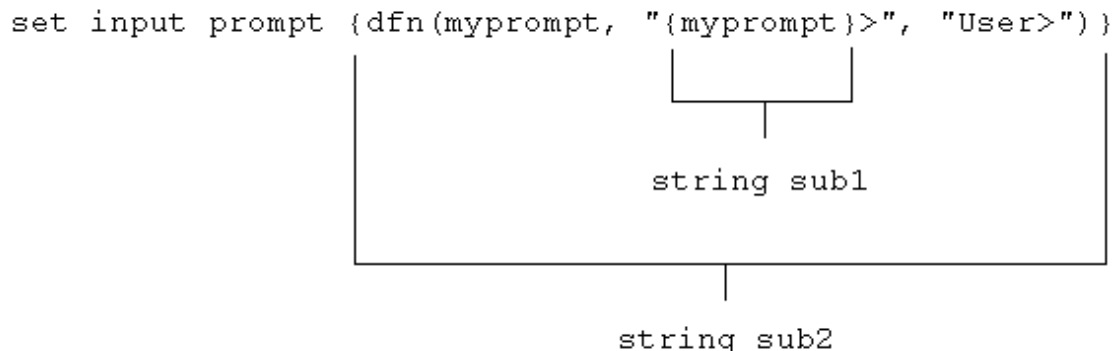


Figure 3. Nested String Substitution

The example above sets the EFT 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, EFT 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, EFT processes the outermost string substitution syntax (labeled string sub2 above).

All string substitution processing is performed by EFT 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 EFT 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.

```
EFT> 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 non-quoted string `ENTER YOUR NAME`. The outer set of quotes then is applied to that string and the remainder of the command is processed.

Developing EFT Scripts Using Input Files and Aliases

EFT 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 EFT interface to be even more simple or familiar for its users. This “customization” is generally done by more sophisticated EFT users then handed back to the general user base. This section addresses the areas important to developing EFT scripts.

A script can be in the form of an input file or an alias, EFT 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 EFT command line processing, most notably, the sections on “Special Characters” on page 48 and “EFT String Substitution” on page 49.

EFT Input Files

EFT 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 EFT input file or input script is a file that contains a list of EFT 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 EFT to read and execute the EFT commands in the input file given. For example:

```
EFT> input hostdir
```

This command tells EFT to look for an input script with the file specification *HOSTDIR*, and then read it line by line, executing commands along the way. If the file specification is not found, the INPUT SEARCH qualifier is used to locate the file (see item 2).

2. Using the INPUT qualifier SEARCH.

When EFT 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, EFT determines if it is a EFT command. If not, it looks at the INPUT qualifier SEARCH. If that qualifier is defined, EFT uses the SEARCH path to find an input file by the name of the command it read. If an input file is found, EFT reads and executes it. If no input file is found, EFT 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 99), and then type HOSTDIR on the command line:

```
EFT> hostdir
```

Typing HOSTDIR here gives the appearance that HOSTDIR is actually a EFT command. The INPUT SEARCH qualifier can also contain SEARCH keywords (SITE) and (USER). Refer to “UNIX EFT SEARCH Keywords (SITE), (USER), and (NONE)” on page 91 for more information.

3. On the EFT command line.

The third option is to specify the input file on the EFT command line when it is invoked. This option is most often used when running EFT within a batch job. It simply tells EFT to read and execute all of the commands within the input script and then exit EFT. For more information, see “Running EFT as a Batch Job Under UNIX” on page 95.

Echoing Input Scripts at the Terminal

In order to have EFT 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” on page 99 for more details on these qualifiers. The section “Debugging an EFT 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 EFT 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 EFT 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:

```
EFT> transfer
Transfer being made to what host? BLUESKY
User 10 on host BLUESKY? guest
Password for user guest? _____
File to be transferred? tmpfile
*****
File 'tmpfile' has been transferred to BLUESKY.
*****
EFT>
```

Passing Parameters to a Script

Parameters may be passed to a EFT 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:

```
EFT> 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, EFT performs string substitution. After string substitution the command lines appear as:

```

EFT> 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:

```

EFT> 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 an EFT 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 EFT string functions. (See the section entitled “String Functions” on page 53 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:

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

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 EFT Labels and GOTOS

To make EFT 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 or other special characters) 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 EFT 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 FILEx 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 EFT 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 EFT 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 EFT. 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, EFT 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" 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 | Parameters |
|---------|--------------------|
| ON | exception [action] |

where

exception is any one of the following:

- ERRor** on EFT 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 single valid EFT command or alias, the most likely of which being:
- CONTinue** ignore the exception
- EXit** exit the current EFT session
- GOTO** GOTO a specified label
- INput** input a specified EFT 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 anyone of the exceptions listed above and perform a predefined action. It is most useful within EFT scripts for tailoring exception handling. ON commands generally appear at the beginning of a EFT 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, EFT terminates all input levels (if within nested input scripts), and returns to the interactive level. If the INPUT CONTINUE qualifier was set, EFT 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 EFT error occurs. Without an ON ERROR specified, EFT terminates all input levels (for nested input scripts), and begins processing at the interactive level. If the INPUT CONTINUE qualifier was set, EFT 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, EFT just ignores host errors and continues processing the next command.

For further details about the ON command, see "Command Descriptions" on page 99.

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). Status() returns either an "S" for Success, or an "E" for Error. Since it is a string function, the result is a string. Status() can be combined with any other string function or EFT 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 EFT Aliases

Aliasing is simply another form of EFT scripting. Although the previous sections discussed scripting in the context of INPUT files, everything described applies to EFT aliases. Aliases can be thought of as EFT 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 EFT 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 an EFT 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:

```
EFT> set alias fetch receive
```

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

```
EFT> show alias fetch
EFT: 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 EFT 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:

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

and,

```
EFT> 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:

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

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

```
EFT> show alias fetch
EFT: 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:

```
EFT> fetch sourcefile
```

and,

```
EFT> 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:

```
EFT> set alias fetch
```

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

EFT Aliases Vs Host Aliases

There are two types of aliases, Host Aliases and EFT aliases. Host aliases are either Local aliases or Remote aliases. That is, their definitions translate to either Local host commands or Remote host commands. EFT aliases translate to EFT 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, EFT aliases are defined using SET ALIAS. The format is:

| Command | Parameters |
|-----------|------------------------|
| SET ALIAS | alias_name EFT_command |

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

| Command | Parameters |
|-----------------|-----------------------------|
| SET LOCAL ALIAS | loc_alias_name host_command |

and

| Command | Parameters |
|------------------|-----------------------------|
| 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 EFT aliases and Host aliases is that EFT aliases may have multicommand definitions (see the section entitled “Creating Multicommand EFT Aliases” on page 84), 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 EFT aliases. The topics apply to host aliases as well unless specifically designated as “EFT Aliases”. For more information on host aliases, refer to the Local or Remote User’s Guide.

Creating Multicommand EFT Aliases

An alias can translate to a single EFT command as shown above where `FETCH` was mapped to `RECEIVE`. EFT 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 EFT 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:

```
EFT> 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:

```
EFT> show alias fetch
EFT: FETCH ..... connect hosta guest netex
EFT:         receive sourcefile
EFT:         disconnect
```

Notice that EFT 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. It will function correctly and not be a constant burden.

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 EFT commands. EFT 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:

```
EFT> 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:

```
EFT> show alias fetch
EFT: FETCH ..... connect hosta {1} {2}
EFT:             receive {3}
EFT:             disconnect
```

To invoke the alias with the same parameters that were embedded in the alias earlier, a user would type:

```
EFT> fetch guest netex source file
```

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:

```
EFT> connect hosta guest netex
EFT> receive sourcefile
EFT> disconnect
```

Now refer back to the empty '{}' used in the definition of FETCH earlier. This special notation is used to tell EFT 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 "EFT String Substitution" on page 49). If parameter substitution is not turned off, EFT 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 75 for further details.

Accepting Input within an EFT 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:

```
EFT> 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 EFT 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 EFT 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 EFT not to echo the response back to the terminal. It should be noted that some versions of EFT cannot support this feature due to operating system limitations. For more information, refer to the *ASK* command in the “Command Descriptions” on page 99.) 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:

```
EFT> fetch
User Id?  guest
Password? _____
File Name? sourcefile
EFT:      < connect results >
EFT:      < receive results >
EFT:      < 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 “EFT Input Files” on page 76.

Abbreviating Alias Names

To make life more simple 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:

```
EFT> set alias FETch receive
```

Now the definition can be displayed:

```
EFT> show alias fetch
EFT: 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 (e.g., *ABc* and *ABd*), the first alphabetical alias is processed (*AB* would execute *ABc*).

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:

```
EFT> set alias "FETch A FIle" receive
```

Its definition can then be displayed as:

```
EFT> show alias "fetch a file"
```



```
EFT: FETch A File ..... receive
```

And it can be invoked as:

```
EFT> 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 EFT commands such as SHOW HOST, SET ALIAS, etc.

Debugging an EFT 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 EFT when the alias is invoked. To tell EFT to display each command in its “string substituted” form, set the VERIFY qualifier of the INPUT command:

```
EFT> set input verify on
```

Now every command issued from the command line gets re-displayed 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:

```
EFT: FETCH ..... connect hosta {1} {2}
EFT:          receive {3}
EFT:          disconnect
```

To debug or verify each command as it gets executed, the INPUT VERIFY qualifier is turned on and FETCH is invoked with parameters:

```
EFT> set input verify on
EFT> fetch guest "fast netex" sourcefile
EFT: connect hosta guest fast netex
EFT: ----- connect results -----
EFT: receive sourcefile
EFT: ----- receive results -----
EFT: disconnect
EFT: ----- disconnect results -----
EFT>
```

Another debugging tool is the CONTINUE qualifier of the INPUT command. By default, EFT 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 EFT to keep processing an alias or input script even if an error condition occurs, set the CONTINUE qualifier of the INPUT command:

```
EFT> set input continue on
```

Now when the FETCH alias (or any alias) is invoked EFT 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 VERIFY and CONTINUE are initialized to off by EFT. 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. See “EFT Input Files” on page 76 for more details.

Error Message Formatting

EFT 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 | EFT host independent message. |
| EFTxxx | EFT host dependent message where xxx represents the product number of the host generating the error (e.g. EFT263, EFT213, etc.) |
| SI | EFT Service Initiator (SI) host independent message. |
| Sixxx | EFT Service Initiator (SI) host dependent message. |
| MUXxxx | EFT Multiplex Server (MUX) host independent message. |
| MUXxxx | EFT Multiplex Server (MUX) host dependent message. |
| NETEX | A NETEX generated error (only in NetEx environments). |
| 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 EFT message are extracted using the string function msg() (described in the section on “String Functions”). The default message format can be displayed as:

```
EFT> show output format
EFT: FORmat ..... {msg("text")} ({msg("facility")}-{msg("code")})
```

With this format defined, a simple “Invalid command” error would generate the following:

```
EFT> badcommand
EFT: Invalid command 'badcommand' (UA-4708).
```

The user can modify the 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 EFT 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 {}

```
EFT> set output format {} {msg("severity")}:{msg("facility")}-{msg("code")}
```

With this message format, the same “Invalid command” error would generate the following display:

```
EFT> badcommand
EFT: 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.

EFT Code Conversion

In NetEx environments, NetEx is ordinarily responsible for performing code conversion between ASCII and EBCDIC computer systems. EFT provides an alternative code conversion facility intended for environments where TCP/IP is the networking protocol or for those sites requiring more flexibility than that offered within NETEX. EFT 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 conversions separately.
- It allows EFT to offer optional data verification facilities (CRC) for character file transfer as well as bit-stream transfer. Refer to “EFT Data Verification” on page 89 for more information.

All EFT code conversion is controlled by means of the TRANSLATE command. Refer to the TRANSLATE command as described in “Command Descriptions” on page 99. Using TRANSLATE, a site administrator can define code conversion tables, review the current tables, and enable or disable the EFT 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, site administrator can define the tables and enable EFT code conversion as part of the CONNECT/LOGIN process to particular systems.

EFT code conversion is enabled by default in TCP/IP environments. When EFT code conversion is enabled, it replaces NetEx/IP code conversion in all communications between systems in NetEx/IP environments.

To protect the EFT 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 EFT 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.

EFT Data Verification

EFT 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, EFT 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 re-send an aborted file a specified number of times. For readability, this alias uses the ASSIGN and TEST aliases that are shipped with EFT.

*

```

* 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, EFT code conversion is used (refer to “EFT 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, EFT 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 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.

EFT Data Compression

Support has been added to EFT 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 (RLE)

```

The compression method currently supported is RLE. The RLE method uses a simple Run Length Encoding algorithm that counts strings of repeated characters (usually spaces or nulls). This method provides compression ratios of typically 80% to 95%. 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 EFT 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 EFT 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 |

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 (eg. ASCII to EBCDIC) there are some subtle problems caused by the fact that only the EFT 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:

| EFT 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 |
| 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 |

UNIX EFT SEARCH Keywords (SITE), (USER), and

(NONE)

This section assumes the reader has a working knowledge of the EFT commands as well as a general understanding of the SEARCH qualifier for each of them.

The EFT commands CONNECT, INPUT, HELP, and TRANSLATE have SEARCH qualifiers associated with them. The SEARCH qualifiers generally instruct EFT 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 EFT site or root directory on the local or remote host. (SITE) is where system wide EFT files are stored. The actual value for (SITE) can be determined by displaying the LOCAL or REMOTE qualifier called ROOTDIR:

```
EFT> show local rootdir
```

or

```
EFT> show remote rootdir
```

(USER) This keyword refers to the user’s login or home directory (or equivalent) on the local or remote host. Often (USER) is where users keep personal files (such as startup files) that are intended for tailoring EFT to personal taste. The actual value for (USER) can be found in the LOCAL and REMOTE informational qualifier HOMEDIR, as shown below:

```
EFT> show local homedir
```

or

```
EFT> show remote homedir
```

(NONE) This keyword simply tells EFT 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 EFT not to look for any files.

The keywords (SITE) and (USER) can be used as they appear above, or with file names appended to them. Appending a file name to (SITE) or (USER) tells EFT to look in the specified location for the given file name. When specified without a file name, the keywords have implied file names that EFT attempts to locate. The rest of this section addresses each EFT 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 EFT startup files on the remote host following a successful connection. The default definition for SEARCH is the string “(SITE) (USER)”. (SITE), when declared without a file name, always refers to the file SSERVER.UA on the remote host in the (SITE) directory. When (USER) is declared without a file name, it implies a file by the name of SERVER.UA in the (USER) directory. (NONE) tells EFT 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 UA.NEW located in the EFT root directory, issue the following command:

```
EFT> set connect search (SITE) (USER) (SITE)UA.NEW
```

The user should be reminded that the order here is important. EFT uses the SEARCH definition from left to right. In the example above, (SITE) would be searched first, then (USER), and finally (SITE)UA.NEW.

HELP

The HELP SEARCH qualifier is used to locate EFT help files on the local host. The default definition for SEARCH is (SITE). (SITE), when declared without a file name, always implies the file *userhelp.ua* in the (SITE) directory. (USER) has no implied file name 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.

For example, issue the following command to define the HELP SEARCH qualifier to look for a user created help file called *myhelp.hlp* located in the user's login directory, and the default EFT help file:

```
EFT> set help search (USER)myhelp.hlp (SITE)
```

INPUT

The SEARCH qualifier for the INPUT command is used to locate user defined EFT input scripts. There are no implied file names associated with (SITE) or (USER) for this command, which means specifying (SITE) or (USER) without an accompanying file name is equivalent to defining it as (NONE). The user can, however, append a file name onto (SITE) or (USER) instructing EFT to locate the specified file upon an INPUT request. In fact, the appended file name may even include an asterisk '*' which EFT replaces with the name of the input file specified.

For example, the INPUT SEARCH qualifier can be defined to search all the files in the root directory that have an extension of '.ua':

```
EFT> set input search (SITE)*.ua
```

Now any input requests (via the INPUT command or implied), will only require a file name. For example, if a EFT input script by the name of *myjob.ua* is located in the root directory, it can be invoked as:

```
EFT> input myjob
```

or

```
EFT> myjob
```

The first example only uses the INPUT SEARCH path if a file with a file specification of *myjob* is not found first.

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 file name, always refers to a file in the root directory on the local host with the name {HOSTCODE:REMOTE}.ua, 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 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 *unxtoibm.ua* located in the user's login directory:

```
EFT> set translate search (USER)unxtoibm.ua
```

User-Definable HELP Files Under UNIX

A site has the ability to create its own HELP files that EFT 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 EFT to look for additional help files. By default, the HELP SEARCH qualifier is defined as:

```
EFT> show help search
EFT: SEARCh ..... (SITE)
```

(SITE) is a special SEARCH keyword used by EFT to indicate the default help file *userhelp.ua* from the local EFT root directory. This help file contains all EFT commands, qualifiers, examples, etc. To instruct EFT to also look in a site or user defined help file, for example, */root/uahelp/myalias.ua*, type the following:

```
EFT> set help search /root/uahelp/myalias.ua (SITE)
```

The order of the SEARCH list is important. In the example above, EFT will look first in the *myalias.ua* help file upon any help request. EFT then 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 EFT default one, in the root directory. The (SITE) keyword allows for appending- file names as shown below:

```
EFT> set help search (SITE)myalias.ua (SITE)
```

This example instructs EFT to first look for any help in the file *myalias.ua* in the (SITE) directory, followed by the EFT default file *userhelp.ua* from the same directory. Refer to “UNIX EFT SEARCH Keywords (SITE), (USER), and (NONE)” on page 91 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 EFT 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 EFT help file as a guide.

Running EFT as a Batch Job Under UNIX

Since the EFT Initiator is a UNIX 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 EFT script file that performs the required operations (e.g. connect to a remote host, transfer a series of files, and exit). Then, edit a UNIX command file that contains the following line:

```
$ user input_ua
```

Where *input_ua* is the name of the EFT input script created earlier. If positional parameters were used within the input script, these would be appended to the line above. It is also important that the symbol *user* is defined for the batch process, to invoke EFT.

Once the EFT input script and the UNIX command file have been created, use the UNIX Background Processing syntax to execute the command file. Since this will not be an interactive session, EFT knows not to prompt for input during the batch run.

The following is a sample UNIX command file that has EFT commands embedded in it. This common script can be run in the background as:

```
$ sample &
```

Read through the sample in Figure 4 on page 96, taking special note of error catching and checking by both UNIX and EFT. The main point is that each EFT command returns a status which can then be passed through to UNIX for special processing

```

# SAMPLE:
# Sample UNIX Script File that invokes EFT with the
# EFT commands included right in the script (command
# procedure). The script connects to a remote host, sends some
# files, and exits with status. The status from EFT
# is then passed to the command procedure itself and evaluated.
#
# Output the date UNIX has and invoke the EFT Initiator
# as 'USER'.
#
date
user <</
# Note the above, '<</', this tells UNIX to use this script
# file as input to EFT until it sees a line consisting
# only of '/'.
#
# 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'. If normal
# exit, return a success to the command procedure.
#
text Today is {date()} - {time()}.
login sun guest netex
/
set st = $status
if ($st == 0) then
echo ..... EFT Job Completed Successfully!
else
echo ..... Error from EFT: $st .....
endif
# Following the '/' above, any UNIX commands could have been used
# to interpret and act on the exit status returned from USER-
# Access.
# Here we just saved the UNIX status, tested for success, and
# printed a success or failure message.

```

Figure 4. Sample Command Script

Running a UNIX Stand-Alone EFT Server

The EFT Server (or Responder) is typically invoked by the Service Initiator following a successful login request under UNIX EFT. 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 background processing, when login is not crucial, or when the server is invoked from some non-privileged user account.

The EFT Stand-Alone Server (or Responder) can be invoked from a UNIX command line (interactively or in batch), with the instruction:

```
eftip-server/efntx-server [-keyword value]
```

Where:

| | |
|-----------------------|---|
| eftip-server | |
| eftntx-server | is the command to invoke the EFT Stand-Alone Server. UASERVER should be defined at installation time. If it is not, contact the site administrator. |
| -keyword value | (optional) specifies optional command line keywords that may be given to affect the operation of the EFT session. The following are valid keywords: |
| -BLOCKsize | specifies an alternative default blocksize value in which to offer. The default is 32768. |
| -HOMEdir | specifies the name of the user's "login" or "home" directory when EFT is invoked. Changing this keyword's value redefines the location EFT uses to locate user startup files. |
| -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. |
| -PREfix | an alternative server prefix string that precedes au server displays. The default is "Server: ". |
| -ROOTdir | specifies the name of the installed EFT root directory containing site specific server help and startup files. There is generally no reason to modify this keyword. |
| -SEArch | specifies the search path EFT 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 service name to listen for TCP/IP environments (or offer for NetEx/IP environments). The default is "EFT". |

As mentioned earlier, the Stand-Alone Server would generally be run as part of a UNIX batch job. Using the exit status returned by each EFT command, special processing can be built into a batch job as seen in the section entitled "Running EFT as a Batch Job Under UNIX". A Stand-Alone Server could be run in the same manner as described there.

Advanced UNIX Transfer Modes

As discussed in "File Handling Under UNIX " on page 40, several file transfer modes are supported: CHARACTER, RECORD, STREAM, BACKUP, RESTORE, and COPY. Not all of these modes are supported by all EFT 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 EFT 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 (8 bits on many machines) versus word addressability (60 bits on say a CYBER), there will tend to be some slight variations.

CHARACTER mode is used to transfer text files, such as program source files. EFT assumes the file is record-oriented and provides character code conversion between the ASCII character set and the remote host's native character set.

In CHARACTER mode, UNIX EFT attempts to access the file a block at a time (bypassing record at-a-time overhead) and pack blocks into one of two formats. A linefeed terminated record format as on UNIX systems (format <LF>) or a carriage-return, linefeed terminated record format (format <CR-LF>) is used.

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, EFT 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 system with 8-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 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 UNIX hosts, COPY mode is recommended rather than STREAM, because COPY mode copies the file attributes as well as the data.

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. For UNIX hosts it involves accessing the file in block I/O mode, copying the file as a bitstream, and copying the file's characteristics. This mode is provided as a means to make a backup copy of a file on another host or as part of the COPY mode for transferring files UNIX to UNIX. Record lengths and total file length are preserved so that the file can be restored as it originally existed even when the file is copied to a host which has different addressability limitations.

RESTORE mode is the counterpart of BACKUP mode. It is used to restore a file which has previously been copied in BACKUP mode. Again block I/O mode is used to create the file on the destination UNIX 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 UNIX to UNIX.

COPY mode is a special means of copying files between peer hosts. On UNIX host systems, COPY mode allows copying of most any sequential disk file from one UNIX to another. It is efficient because it essentially uses BACKUP mode on the originating host and RESTORE mode on the destination host. It also provides for full wildcarding capability because one can now copy a complete directory branch with a single command even when the directory contains files of character and/or binary data.

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 HOST
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 User Guide 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. ASK terminates as soon as the user hits a carriage return or upon expiration of the TIMEOUT qualifier value. If a user responds without typing anything other than a carriage return or if the ASK request times out, the variable gets defined to nothing 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 nothing.

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 | Qualifiers | Parameters |
|---------|---|------------------|
| ASK | [-DEFault string] [-PROMpt string] [-SECure on] off [-TIMEout seconds] | [var1 [var2...]] |

Where:

- ASK** (required) the verb for this command.
- DEFault** (optional) the default string passed to ASK if the user does not provide one or if the ASK command "times out". 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 EFT 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) the number of seconds to allow the user to respond to the ASK command before timing out. A value of zero means to wait forever. This is the default. The minimum spelling is -TIM.

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:

```
EFT> ask -prompt "Name? " -default Ed name
Name? Joe Smith
```

The user's response here *Joe Smith* is read into variable *name*. Now using standard EFT string substitution syntax, you can display the value of *name* with the TEXT command:

```
EFT> text Hello {name}.
EFT: Hello Joe Smith.
```

Alternatively, you can display the value of variable *name* as:

```
EFT> show variable name
EFT: NAME ..... Joe Smith
```

If you wanted to prompt for the variables *day* and *date*, you could type:

```
EFT> 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 Table (the NCT for NETEX environments). For TCP/IP environments, the host name must be known via DNS or the local host lookup file.

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 offer available. You can set or display the service name with the SET CONNECT SERVICE and SHOW CONNECT SERVICE command respectively. The recommended way for connecting to a remote host is to use the LOGIN alias. This alias prompts the user for a hostname, username, and password, 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 passed to accomplish this. Some systems provide a default EFT 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. EFT 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. (For NETEX sites, the status of your site really determines the number of connections allowed based on the NETEX configuration and current number of NETEX sessions). 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, EFT displays intermediate CONNECT messages informing you of its current connect status. If a connection cannot immediately be established due to NETEX errors 3501 (“Service not offered on specified host”) or 3502 (“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 | Qualifiers | Parameters |
|---------|--|----------------------------------|
| CONnect | [-ACCcount code] [-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.
- ACCcount** (optional) login account code that may be used by the host in which you are attempting to connect.
- APPlication** (optional) identify the login application on the remote host.
- BLOCKsize** (optional) local host's maximum block size in bytes. This size gets sent to the remote host on a CONNECT and a new negotiated block size gets returned. The REMOTE "information only" BLOCKSIZE qualifier contains this new value.
- COMmand** (optional) startup command file name that may be used by the host in which you are attempting to connect.
- INTerval** (optional) connect retry interval in seconds used for connection retries when connecting to a remote host. The default is to retry every 5 seconds.
- 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.
- PROFile** (optional) startup profile file name that may be used by the remote host during a connect sequence.
- PROJect** (optional) login project code that may be used by the remote host during a connect.
- QUIet** (optional) tells EFT whether or not to display intermediate connect messages. This value should be set to either ON or OFF. The default is OFF.
- SCRipt** (optional) script file name that is used by some remote hosts during the connect and login process.

- SEArch** (optional) describes the server startup files to be read during connect time. Refer to “UNIX EFT SEARCH Keywords (SITE), (USER), and (NONE)” on page 91. The default is “(SITE)(USER)”.
- SECondary** (optional) secondary login password that can be used by the remote host during connect.
- SERvice** (optional) service name that EFT tries to connect to on the remote host during a connect. This service name is EFT by default.
- 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 EFT. The default is to time out after 2 minutes.
- USERname** (optional) default login name of the user attempting to connect. USERNAME is usually overridden on the command line of the CONNECT command.
- 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.
- host** (required) name of the remote host to which you want to connect.
- username** (optional) your login user name on the remote host.
- password** (optional) your login password on the remote host.
- 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 local network database (either in the local hosts file, accessible through DNS, or described in the NCT file)) with a user name of **smith** and a password of **allen**, type:

```
EFT> connect sun smith allen
EFT: Connected to Service Initiator on host `SUN'
=====
                Welcome to SUN
                *      *      *
=====
EFT: Logged in as user `smith'.
EFT: 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, type:

```
EFT> connect -blocksize 4096 -timeout 10 -
More>> vax jones jane
EFT: Connected to Service Initiator on host `VAX'
=====
                Welcome to VAX
                *      *      *
=====
```

```
=====
EFT: Logged in as user `jones`.
EFT: 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:

```
EFT> set connect blocksize 4096
EFT> set connect timeout 10
EFT> connect vax jones jane
EFT: Connected to Service Initiator on host `VAX`
=====
                Welcome to VAX
                *      *      *
=====
```

```
=====
EFT: Logged in as user `jones`.
EFT: 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 EFT session where as the first approach would only affect the connect being issued.

Related Topics

- DISCONNECT command
- SET HOST command
- SHOW HOST 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 | Qualifiers | Parameters |
|----------|------------|------------|
| 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 EFT script - this script
* continues should any EFT 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

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 HOST 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 0 connections takes place following an EXIT or QUIT from a EFT session.

Format

| Command | Qualifiers | Parameters |
|------------|--------------------------------|------------|
| DISconnect | [-QUIet ON] OFF] | |

Where:

DISconnect (required) the verb for this command. The minimum spelling is DIS.

-QUIet (optional) tells EFT 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 HOST:

```
EFT> show host
EFT:          (1) Host=vx1      User=scott
EFT: active --> (2) Host=IBM    User=meyers
```

A DISCONNECT at this point terminates the current “active” connection (host *IBM*) as seen below:

```
EFT> disconnect
EFT: Disconnected from host IBM.
EFT> show host
EFT:          (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:

```
EFT> set host vx1
EFT> disconnect -quiet
```

When the QUIet qualifier is used, the message confirming the disconnect is not displayed.

Related Topics

- CONNECT command
- SET HOST command
- SHOW HOST command

EXIT Command

Description

The EXIT command causes EFT to exit to the previous input level. 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 EFT to terminate. If issued from an input script as part of a non-interactive EFT session (e.g., a batch job), the session terminates.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| 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 EFT 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 EFT to immediately abort. If status is not specified, an exit status of Success is assumed.

Examples

If you desire to leave the EFT session and return to your local system's command line interpreter, you would type:

```
EFT> exit
$
```

At this point you should receive the prompt you normally receive from your system's command line interpreter (e.g. \$).

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 EFT 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 EFT to continue processing at the given label. The label must be the first item to appear on the EFT 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 | Qualifiers | Parameters |
|---------|------------|------------|
| 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
- ON

HELP Command

Description

The EFT help facility gives you on-line access to EFT topics including host specific qualifier information (both locally and remotely), formats and descriptions of all EFT 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- EFT 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 | Qualifiers | Parameters |
|---------|------------------|--------------------|
| HELp | [-SEArch string] | [topic [subtopic]] |

Where:

- HELP** (required) the verb for this command. The minimum spelling is HEL.
- SEArch** (optional) allows the user to search alternate paths for EFT help files. For more information, refer to “UNIX EFT SEARCH Keywords (SITE), (USER), and (NONE)” on page 91. The default is (SITE). The minimum spelling is -SEA.
- topic** (optional) name of a EFT 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:

```
EFT> 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:

```
EFT> help send
```

Now, depending on the subtopics of SEND available, you might type:

```
EFT> help send example
```

which would display a sample SEND command.

The SEARCH qualifier for the HELP command is used to define where the EFT help files exist:

```
EFT> show help search
EFT:
EFT: SEArch ..... (SITE)
EFT:
```


The default is "(SITE)", when "(SITE)" is not succeeded by a file name the file "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:

```
EFT> show alias login
EFT:
EFT: LOGin ..... ask -prompt "Host? " host
EFT:                ask -prompt "Username? " user
EFT:                ask -sec -prompt "Password? " pass
EFT:                !connect {host} {user} {pass}
EFT:
```

The HELP command does not find the text for the LOGIN alias unless the user help file has been included on the HELP SEARCH path, in this case the help file "alias.hlp" is in the (SITE) location:

```
EFT> help login
EFT: Help is not available for 'login' (UA-4301).
```

Now add to the HELP SEARCH path "(SITE)alias.hlp":

```
EFT> set help search {search:help} (SITE)alias.hlp
EFT> show help search
EFT:
EFT: SEArch ..... (SITE) (SITE)alias.hlp
EFT:
EFT> help login
EFT:
EFT: FORMAT
EFT:
EFT:     LOGin
EFT:
EFT: DESCRIPTION
EFT:
EFT:     The LOGin alias is used to prompt user's for the
EFT:     necessary LOGin information.
EFT:
```

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 EFT to take its commands from the specified input file on the local host. This file may contain any number of EFT commands. These commands can be structured in such a way that a sophisticated user could create predefined EFT procedures that can be used by beginning EFT users. These procedures can prompt users for input, give them instructions, and issue EFT commands for them.

Format

| Command | Qualifiers | Parameters |
|---------|---|----------------------|
| 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 EFT how to respond to an error encountered when processing input files. This value should be set to either ON or OFF. ON tells EFT 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 default is OFF. The minimum spelling is -CONT.
- ECHO** (optional) tells EFT 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 default is OFF.
- PROMpt** (optional) the string used as the EFT command prompt. The default is EFT>. The minimum spelling is -PROM.
- PROMPT2** (optional) a secondary EFT command prompt string used for command continuation. The default is More>>.
- SEArch** (optional) search path used for default INPUT commands (the location of input files). SEARCH is only used if EFT cannot locate the source file specified on the command line. A SEARCH path is a space-separated list of UNIX file specifications. If SEARCH is defined, EFT will use it in an attempt to locate input files. Refer to “UNIX EFT SEARCH Keywords (SITE), (USER), and (NONE)” on page 91 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 default is OFF. The minimum spelling is -VER.
- source** (optional) the file specification for the Input file on the local host. EFT 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 EFT commands from a file named *setup.si* located in the current local working directory, you would type:

```
EFT> input -echo on setup.si
```

EFT will then attempt to execute all of the commands in *setup.si* before displaying the interactive EFT> 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:

```
EFT> 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:

```
EFT> 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 login directory:

```
EFT> input myalias
```

The “input myalias” command uses the SEARch path to find the EFT 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, check for input file. Setting the search path as in the previous example:

```
EFT> set input search (USER)*.ua
```

The “myalias.ua” script file can be executed by issuing the command:

```
EFT> myalias
```

Command processing checks for an alias by the name “myalias”, then a EFT 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 or 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 EFT> prompt. All of your remote host connections will be intact.

Format

| Command | Qualifiers | Parameters |
|---------|---|------------|
| LOCa1 | [-INteractive ON] OFF] [-PREfix string] [-QUIet OFF] ON] [-SHELL string] | [command] |

Where:

- LOCa1** (required) the verb for this common. The minimum spelling is LOC.
- INteractive** (optional) 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** (optional) 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 EFT or remote output. You can define this qualifier to be null if no prefix is desired. The minimum spellings is -PREF.
- QUIet** (optional) forces local command output not to be displayed. The default is OFF. The minimum spelling is -QUI.
- SHELL** (optional) the UNIX command shell used when executing LOCAL commands. The minimum spelling is -SH
- 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 UNIX environment. With the exception of DIRECTORY, these qualifiers cannot be modified by the user.

- DIRectory** (string) the current working directory on the local UNIX host. This qualifier can be modified by using the SET LOCAL DIRECTORY command.
- HOSTCODE** (string) the native host character code.
- HOSTTYPE** (string) Operating system type.
- PID** (integer) Process ID.

- PRODUCT** (string) NESi Product number.
- STATUS** (string) exit status of the last LOCAL command.
- VERSION** (string) EFT version number.

Examples

To display all users currently logged into your local UNIX host you would execute the UNIX *who* command from EFT as:

```
EFT> local who
Unix: admin      console   Jun 1  05:32
Unix: quest     ttya      Jun 7  06:18
Unix: sam       ttypl     Jun 1  16:59
```

The prefix **Unix:** tells you results are being returned from the local UNIX host.

One could force the prefix to be something else by changing the value of the PREFIX qualifier. For example:

```
EFT> local -prefix "HOSTA: " who
```

This command would cause the prefix **HOSTA:** to display before each line of output instead of the default **Unix:** .

To drop into the local system's command line interpreter without losing remote host connections, you can simply type:

```
EFT> local
$
```

The local system's prompt should appear (e.g., \$). At this point you can interact with your local system in the usual manner until you again want to return to your EFT 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.).

```
$ exit
EFT>
```

Related Topics

- REMOTE command
- SET command

ON Command

Description

The ON command allows users to catch any one of the exceptions:

- ERRor** on EFT 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 EFT error occurs. Without an ON ERRor specified, EFT terminates all input levels (for nested input scripts), and begins processing at the interactive level. If INPut CONTInue is ON, EFT 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, EFT terminates all input levels (for nested input scripts), and begins processing at the interactive level. If INPut CONTInue is ON, EFT 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 VMS command or DCL script exits with an unsuccessful 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 | Qualifiers | Parameters |
|---------|------------|--------------------|
| ON | | exception [action] |

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 EFT command or alias.

Examples

The following is a short EFT script that immediately exits should any error occur:

```
* Sample EFT 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 EFT command EXit.

The following is a short EFT script that immediately continues execution should any keyboard interrupt occur:

```
* Sample EFT 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 EFT 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 EFT 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
INput

OUTPUT Command

Description

The OUTPUT command instructs EFT 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 | Qualifiers | Parameters |
|---------|---|----------------------------|
| OUTput | <code>[-COLumns integer]</code> <code>[-CREate APPend]</code> <code> DELeTe </code> <code> NEW </code> <code> REPLace </code> <code>[-FORMat string]</code> <code>[-HOLD ON]</code> <code> OFF </code> <code>[-LINes integer]</code> <code>[-PREFix string]</code> <code>[-QUIet OFF]</code> <code> ON </code> <code>[-RESize OFF]</code> <code> ON </code> <code>[-TRUNcate OFF]</code> <code> ON </code> | <code>[destination]</code> |

Where:

- OUTput** (required) the verb for this command.
- COLumns** (optional) the maximum number of columns per terminal page of output. This represents the maximum number of characters across a page or terminal screen.
- CREate** (optional) describes how to create the output file on the local system. The valid values are APPend, DELeTE, NEW, and REPLace. The default is REPLACE.
- FORMat** (optional) all EFT messages are displayed using the format string defined by this qualifier. The 'msg()' string function is used to construct an appropriate format for EFT messages.
- HOLD** (optional) 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.
- LINes** (optional) the maximum number of lines per terminal page of output.
- PREFix** (optional) the prefix string displayed before each line of EFT output. The default is EFT:.
- QUIet** (optional) when this qualifier is ON, no EFT output is displayed to the user's terminal. If an output destination file exists, the output is still captured to the destination file. The default is OFF.
- RESsize** (optional) when this qualifier is ON, EFT will resize the terminal window to the window size. The default is ON.

- TRUNcate** (optional) works in conjunction with the COLumns qualifier. If any EFT output lines are longer than the COLumns value, the lines are truncated when this qualifier is ON. The default is OFF.
- destination** (optional) a local file specification that will receive the captured EFT 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 EFT output to a file on the local host named session, you would type:

```
EFT> output session
```

Now every line of EFT output that appears on the screen will also be sent to the output file until you close the file with another output command:

```
EFT> output
```

To tell EFT to hold the screen every time a full screen of output is displayed, type the following:

```
EFT> set output hold on
```

Related Topics

INPUT command

QUIT Command

Description

The QUIT command causes EFT 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 EFT to terminate. If issued from an input script as part of a non-interactive EFT session (e.g., a batch job), the session terminates.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| 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 EFT 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 EFT to immediately abort. If status is not specified, an exit status of Success is assumed.

Examples

If you desire to leave EFT and return to your local system's command line interpreter, you would type:

```
EFT> quit
$
```

At this point you should receive the prompt you normally receive from your system's command line interpreter (e.g. \$).

If you desire to leave EFT and return an Error status to your local system, you would type:

```
EFT> quit error
$
```

At this point you should receive the prompt you normally receive from your system's command line interpreter (e.g. \$).

Related Topics

- DISCONNECT command
- EXIT 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 EFT wildcard characters * and ? as well as host specific wildcard characters where the two do not conflict. See the discussion on EFT wildcarding in “Source Wildcard Support for UNIX File Transfers” on page 44 for further details.

Format

| Command | Qualifiers | Parameters |
|---------|---------------|----------------------|
| 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 UNIX ” on page 40 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 UNIX ” on page 40 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 EFT 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 to you.

There is no interactive mode for the REMOTE command. You must specify a remote host command 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 | Qualifiers | Parameters |
|---------|---|------------|
| REMOte | [-PREFIX string] [-QUIet OFF] ON] | 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 EFT 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 local 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. Only the DIRectory qualifier can be modified by the user.

- 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) EFT product number.
- SERvice** (string) the name of the 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 EFT version number

Examples

To display all users currently logged into the remote UNIX host you would execute the UNIX *who* command from EFT as:

```
EFT> remote who
Unix:  admin   console   June 1   05:32
Unix:  guest   ttya      June 7   06:18
Unix:  sam     ttypl     June 1   16:09
```

The prefix `UNIX:` indicates results are being returned from the remote UNIX host.

Assume the remote hose is running UNIX. To display what it thinks is the current date, type:

```
EFT> 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 EFT wildcard characters * and ? as well as host specific wildcard characters where the two do not conflict. See “Source Wildcard Support for UNIX File Transfers” on page 44 and “File Handling Under UNIX ” on page 40 for more details.

Format

| Command | Qualifiers | Parameters |
|---------|---------------|----------------------|
| 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 UNIX host, refer to “File Handling Under UNIX ” on page 40.
- 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 UNIX ” on page 40 for examples of the SEND command under UNIX. 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 EFT so novice users can issue commands without being concerned with switches on the command line. Once a user becomes more familiar with EFT and wants to perform more complex tasks, he can set up commands with 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 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, EFT expects any embedded quotes (that is, within the value) to be “escaped”. There must be 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 | Qualifiers | Parameters |
|---------|------------|---------------------------|
| SET | | command qualifier [value] |

Where:

- SET** (required) the verb for this command.
- command** (required) name of a EFT 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:

```
EFT> set input prompt >>
>> show input prompt
EFT: 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
EFT: DIRectory ..... DRA2: [TEMP]
```


Related Topics

SHOW command
SHOW QUALIFIER

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 EFT users.

You can define three kinds of aliases - EFT, local, and remote. EFT alias definitions are made up of other EFT commands and are invoked simply by typing the alias command name in response to the EFT 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 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 EFT aliases using the EFT escape character ! at the end of the line. Multicommand aliases are discussed further in the "Creating Multicommand EFT Aliases" on page 84.

Format

| Command | Qualifiers | Parameters |
|-----------------------------------|------------|------------------------|
| SET [LOCAl] ALias REMOte | | alias [definition] [!] |

Where:

- SET** (required) the verb for this command.
- LOCAl or REMote** (optional) entering either LOCAL or REMOTE here tells EFT to create a local or remote alias command instead of a EFT 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 EFT alias called *hc* to be the EFT command *HELP COMMANDS*, you would type:

```
EFT> set alias hc help commands
```

Now, to display the EFT command summary, you would just type `hc`:

```
EFT> hc
```

which is equivalent to typing:

```
EFT> 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:

```
EFT> set local alias ? whoami
```

To invoke your new alias, all you need to do it type:

```
EFT> local ?
```

which is equivalent to typing:

```
EFT> local whoami
```

To create a multicommand EFT alias named `STatus` that displays the current `LOCAL` and `REMOTE` qualifier defaults, you would type:

```
EFT> set alias STatus show local !  
More>> show remote
```

To execute your new alias command you would type:

```
EFT> 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 the variable within EFT 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, EFT expects any embedded quotes (that is, within the value) to be “escaped”. There must be 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 global variables that can be defined at one time is limited.

Format

| Command | Qualifiers | Parameters |
|------------|------------|--------------|
| 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. EFT 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 non-blank character.

Example

The following sample script assigns a numeric value to the global “num”:

```
EFT> set global num 1
EFT> text The global num = {num:global}
EFT: The global num = 1
EFT> show global num
EFT:
EFT: NUM ..... 1
EFT:
```

Related Topics

- SHOW GLOBAL
- SET VARIABLE
- SHOW VARIABLE

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 HOST command.

Format

| Command | Qualifiers | Parameters |
|----------|------------|------------|
| 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:

```
EFT> connect vaxa scott john
EFT> connect sun01 meyers ed
```

Typing the SHOW HOST command then would result in:

```
EFT> show host
EFT:          (1) Host=vaxa      User=scott
EFT: active --> (2) Host=sun01   User=meyers
```

We can use SET HOST here to “re-activate” the first connection (to host vaxa) in one of two ways:

```
EFT> set host 1
```

or

```
EFT> set host vaxa
```

The result would be the same:

```
EFT> show host
EFT: active --> (1) Host=vaxa      User=scott
EFT:          (2) Host=sun01   User=meyers
```

Now any EFT command that requires a remote connection (e.g. SEND and RECEIVE), would communicate with host *vaxa*.

Related Topics

- CONNECT command
- DISCONNECT command
- SHOW HOST 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, EFT expects any embedded quotes (that is, within the value) to be “escaped”. There must be 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 the string “{num}” is equivalent to “”.

Format

| Command | Qualifiers | Parameters |
|--------------|------------|--------------|
| 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. EFT 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 non-blank character.

Examples

The following sample script assigns a numeric value to the variable 'num':

```
EFT> set variable num 1
EFT> text The variable num = {num}
EFT: The variable num = 1
EFT> show variable num
EFT:
EFT: NUM ..... 1
EFT:
```

Related Topics

- ASK
- SHOW VARIABLE
- SET GLOBAL
- SHOW GLOBAL

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 | Qualifiers | Parameters |
|---------|------------|---------------------|
| SHow | | command [qualifier] |

Where:

- SHow** (required) the verb for this command. The minimum spelling is SH.
- command** (required) the name of a EFT 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:

```
susel> show input
EFT:
EFT:  CONTInue ..... off
EFT:  ECHO ..... off
EFT:  IGNore ..... off
EFT:  PROMPT2 ..... More>>
EFT:  PROMpt ..... {dfn(host:remote, host:remote, "EFT")}{"> "}
EFT:  SEArch .....
EFT:  VERify ..... off
EFT:
susel>
```

If you are only interested in the default value of qualifier PROMPT2, you would type:

```
EFT> show input prompt2
EFT: 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 remote alias*. If you do not specify either local or remote EFT aliases will be displayed. The *alias* parameter is optional.

Format

| Command | Qualifiers | Parameters |
|------------------------------------|------------|------------|
| SHow [LOCAl] ALias REMOte | | |

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 EFT to display a local or remote alias definition rather than a EFT 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 should type:

```
EFT> show remote alias who
EFT: WHO ..... whoami
```

To see all EFT alias command definitions you would type:

```
EFT> show alias
EFT: suse1 ..... connect suse1 test1 testtest
EFT: SL ..... show local
EFT: ? ..... 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 none is specified, then all global variables and their values are displayed.

Format

| Command | Qualifiers | Parameters |
|-------------|------------|------------|
| 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. EFT variable names must be alphanumeric and no longer than 20 characters.

Examples

The following commands define two variables, “first” and “last”:

```
EFT> set global first john
EFT> set global last doe
```

The following SHOW GLOBAL command displays the values of the global variables “first” and “last”:

```
EFT> show global first
EFT:
EFT: FIRST ..... john
EFT:
EFT> show global
EFT:
EFT: FIRST ..... john
EFT: LAST ..... doe
EFT:
```

Related Topics

- SET GLOBAL
- SET VARIABLE
- SHOW VARIABLE

SHOW HOST Command

Description

SHOW HOST displays all remote hosts currently connected to the local host in this EFT session. Connections are established with the CONNECT command. The list displayed by SHOW HOST 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 | Qualifiers | Parameters |
|-----------|------------|------------|
| Show HOst | | |

Where:

SHo (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:

```
EFT> connect zos5 scott john  
EFT> connect sunrise meyers ed
```

Typing the SHOW HOST command then would result in:

```
EFT> show host  
EFT:          (1) Host=zos5      User=scott  
EFT: active --> (2) Host=sunrise  User=meyers
```

Related Topics

- CONNECT command
- DISCONNECT command
- SET HOST command

SHOW QUALIFIER Command

Description

SHOW QUALIFIER lists all of the valid qualifiers for the specified EFT 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 | Qualifiers | Parameters |
|----------------|------------|------------|
| 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 EFT 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:

```
EFT> show qualifier input
EFT: CONTInue .. continue on error
EFT: ECHO ..... echo input to screen
EFT: PROMpt .... prompt for EFT input
EFT: PROMPT2 ... EFT continuation prompt
EFT: SEARCH .... search path for default INPUT
EFT: VERify .... verify string substitution
```

Related Topics

- SHOW command
- SET command

SHOW VARIABLE Command

Description

The SHOW VARIABLE command displays the currently assigned value of the variable specified. If none is specified then all variables and their values are displayed.

Format

| Command | Qualifiers | Parameters |
|---------------|------------|------------|
| 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. EFT variable names must be alphanumeric and no longer than 20 characters.

Examples

Example #1: The following commands define two variables, “first” and “last”:

```
EFT> set variable first john  
EFT> set variable last doe
```

The following SHOW VARIABLE commands display the values of the variables “first” and “last”:

```
EFT> show variable first  
EFT:  
EFT: FIRST ..... john  
EFT:  
EFT> show variable  
EFT:  
EFT: FIRST ..... john  
EFT: LAST ..... doe  
EFT:
```

Related Topics

- SET VARIABLE
- SET GLOBAL
- SHOW GLOBAL

TEXT Command

Description

Command TEXT writes a string of text to the user's terminal and/or output file. TEXT is usually used within EFT input files for interaction with a user. The string parameter may contain string substitution syntax.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| 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.

Examples

To display a simple line of text on the screen:

```
EFT> text this is a line of text to echo
EFT: 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 EFT variable called *NAME* defined to be *Paul*, you can display its value within a text string as:

```
EFT> text Is your name {NAME}?
EFT: Is your name Paul?
```

Related Topics

- ASK command
- INPUT command

TRANSLATE Command

Description

The TRANSLATE command is used to specify and display EFT code conversion tables, to enable and disable EFT code conversion, and to indicate whether or not EFT code conversion is currently enabled.

The TRANSLATE command followed by an action lets the user display and control the use of the EFT translation tables.

In a NETEX environment, NETEX code conversion is done by default unless the user explicitly turns on EFT translation with the TRANSLATE ON command. In a TCP/IP environment, EFT translation is on by default.

EFT translation can be turned off (i.e., NETEX code conversion is enabled) with the command TRANSLATE OFF. The EFT translation tables can be displayed using the TRANSLATE command followed by the actions DISPLAY or FULL. The RESET action reinitializes the EFT translation tables.

The TRANSLATE command is also used to tailor the default EFT translation tables. The initial EFT translation tables are identical to the NETEX translation tables until modifications are made to the EFT tables by using the TRANSLATE common followed by a native character code and a remote character code.

The initial invocation of TRANSLATE (that is, any TRANSLATE command) loads the EFT tables with the NETEX defaults and then uses the SEARCH qualifier to make changes to the EFT translation tables. The TRANSLATE RESET command reinitializes the EFT tables with the NETEX defaults and uses the SEARCH qualifier again, to make changes to the EFT tables.

When EFT translation is enabled with the command TRANSLATE ON, the EFT 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 EFT 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 EFT protocol:

- upper case alphabetic (A-Z)
- digits (0-9)
- space
- equal sign
- null

Format

| Command | Qualifiers | Parameters |
|-----------|--|--|
| 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 EFT code conversion table. The default is OFF. The minimum spelling is -IN.
- OUT_only** (optional) is used to modify only the outgoing EFT 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 UNIX system the default search path is "(SITE)". Each time a translate command is issued EFT looks in the (SITE) directory for the file "{HOSTCODE:remote}.ua". For additional information, refer to "UNIX EFT SEARCH Keywords (SITE), (USER), and (NONE)" on page 91. 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 (in NETEX environments)
 - FUIl** display entire translate table
 - OFF** disable EFT translation
 - ON** enable EFT 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 ASCII7 and the remote host has a HOSTCODE value of EBCDIC. A translation table is setup to convert the EBCDIC cent sign (0x4A) and solid bar (0x4F) (which are invalid ASCII7 characters) to the ASCII7 left bracket (0x5B) and right bracket (0x5D), with the following EFT commands:

```
EFT> translate 0x5B 0x4A (left bracket <--> cent sign)
EFT> translate 0x5D 0x4D (right bracket <--> solid bar)
```

EFT translation is enabled with the following command:

```
EFT> translate on
```

This translation effects both the incoming and outgoing tables. The EBCDIC cent sign (0x4A) is converted into a ASCII7 left bracket (0x5B) on its way in, and the ASCII7 left bracket (0x5B) is converted into and EBCDIC cent sign (0x4A) on its way out.

Example #2: Suppose the native character set is ASCII7 and the remote host has a HOSTCODE value of EBCDIC.

```
EFT> connect MVS user pw -quiet
```

The current TRANSLATE SEARCH path is:

```
EFT> show translate search
EFT:
EFT: SEArch ..... (SITE)
EFT:
```

EFT translation is turned on with the following command:

```
EFT> translate on
```

At this point EFT uses the SEARCH path to look for a EFT script file using the remote HOSTCODE, in this case EBCDIC. For a UNIX system the SEARCH path tells EFT to look for the file “EBCDIC.UA” in the (SITE) directory. The file “EBCDIC.UA” could contain the lines from the above example:

```
* Sample EBCDIC translation table for
* an ASCII7 host
*
translate 0x5B 0x4A (left bracket <--> cent sign)
translate 0x5D 0x4F (right bracket <--> solid bar)
```

Now every time a connection is active to an EBCDIC host and the EFT translation tables are initialized, the table is automatically loaded.

The translate table can be displayed with the command:

```
EFT> 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}.UA” in the (SITE) directory, the user would rather load a table from the user’s login directory. The current search path is:

```
EFT> show translate search
EFT:
EFT: SEArch ..... (SITE)
EFT:
```

The search path can be changed with the command:

```
EFT> set translate search (USER)translate.ua
EFT> show translate search
EFT:
EFT: SEArch ..... (USER)translate.ua
EFT:
```

The search path for the TRANSLATE command now implies, look for the file “TRANSLATE.UA” in the user’s login directory. The following command turns on translation and initializes the translate tables by reading the file “TRANSLATE.UA” in the user’s login directory:

```
EFT> translate on
```

Example #4: Suppose a Swedish UNIX 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 UNIX default HOSTCODE file “ASCII8.UA” in the (SITE) directory contains the following commands:

```
* EFT TRANSLATE code conversion defaults
```

```

*
*      Native: ASCII7
*      Remote: ASCII8
*
TRANS 0x5D 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 EFT translation tables are initialized with the first invocation of TRANSLATE:

```

EFT> translate
EFT:
EFT: EFT translation is currently disabled.
EFT:
EFT: Loaded the SWEDISH translation tables.

```

Translation is turned on with the following command:

```

EFT> translate on

```

All file transfers and remote command executions that follow, perform translations as defined by the EFT tables.

Host Independent Commands

This section contains descriptions of the built-in host independent commands. In Table 5, the first column lists the UNIX host-independent command (defined as a local alias command), and the second column lists the corresponding UNIX system command that gets executed whenever the UNIX local alias command is issued.

| UNIX Host-Independent Command | UNIX System Command | Description |
|--------------------------------------|----------------------------|--------------------|
| CANcel | /usr/bin/cancel | |
| COPY | /bin/cp | |
| DELeTe | /bin/rm | |
| DIFference | /bin/diff | |
| DIRectory | /bin/ls -al | |
| HELP | /usr/bin/man | |
| PRInt | /usr/bin/lp | |
| QUEue | /usr/bin/lpstat -t | |
| REName | /bin/mv | |
| STAtus | /bin/ps -ef | |
| TYPe | /bin/cat | |
| WHO | /bin/who | |

Table 5. Host Independent Commands

General Alias Commands

This section contains descriptions of the following general alias commands:

- ASsign
- DEBUGOFF
- DEBUGON
- EDit
- ENCrypt
- LDir
- LEDit
- LOgin
- RDir
- SET LOgin
- SHow LOgin
- SLD
- SRD
- TESTt

ASSIGN Alias Command

Description

Assign the results of a string function to the named variable. Function parameters are separated by blanks. Enclose string literal parameters in double quotes.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-----------------------------------|
| ASsign | | variable function [parameters...] |

Where:

ASsign (required) the verb for this command. The minimum spelling is AS.

variable (required) the name of a variable.

function (required) the string function to be applied.

parameters (optional) additional parameters for the string function.

Examples

```
EFT> assign sum add 1 2
EFT> show variable sum
EFT: SUM ..... 3
```

Related Topics

DEBUGOFF Alias Command

Description

An alias for turning off the debug parameters. This command should be used specifically under the direction of Network Executive Software, Inc. Support personnel. This will stop the printing of debug messages (for the client only) which were turned on by the DEBUGON Alias Command.

Note: This alias and script are only to be used under direction from Network Executive Software, Inc. Support. Set Debug and Show Debug commands are not described in the manual as they should ONLY be used under the supervision of Network Executive Software, Inc. Support.

Format

| Command | Qualifiers | Parameters |
|----------|------------|------------|
| DEBUGOFF | | |

Where:

DEBUGOFF (required) the verb for this command. The minimum spelling is DEBUGOFF. The debug settings will resemble the following:

```
CONnect ..... off
COUnt ..... 4096
CRC ..... off
DIScard ..... off
FILE ..... off
HEX ..... off
HSM ..... off
INPut ..... off
INTerval ..... 0
LOCal ..... off
LOG ..... off
MEMory ..... off
MESsages ..... off
MVS_DIAGNS ..... off
PARse ..... off
PROToCol ..... off
RECeive ..... off
REMote ..... off
SENd ..... off
TCP_READ ..... off
TCP_WRITE ..... off
```

Related Topics

DEBUGON Alias Command

DEBUGON Alias Command

Description

An alias for turning on the debug parameters. This command should be used specifically under the direction of Network Executive Software, Inc. Support personnel. This will produce an overabundance of messages (for the client only) which can be used to aid in diagnosing problems encountered by users. (Due to the volume of output, we recommend that your terminal be in logging mode so the output can be sent to Network Executive Software, Inc. Support for analysis.)

Note: This alias and script are only to be used under direction from Network Executive Software, Inc. Support. Set Debug and Show Debug commands are not described in the manual as they should ONLY be used under the supervision of Network Executive Software, Inc. Support.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| DEBUGON | | |

Where:

DEBUGON (required) the verb for this command. The minimum spelling is DEBUGON. The debug settings will resemble the following:

```
CONnect ..... on
COUnt ..... 4096
CRC ..... on
DIScard ..... off
FILE ..... on
HEX ..... off
HSM ..... off
INPut ..... on
INTerval ..... 0
LOCal ..... on
LOG ..... on
MEMory ..... off
MESSages ..... on
MVS_DIAGNS ..... on
PARse ..... on
PROToCol ..... on
RECeive ..... on
REMote ..... on
SENd ..... on
TCP_READ ..... on
TCP_WRITE ..... on
```

Related Topics

DEBUGOFF Alias Command

EDIT Alias Command

Description

Edit a remote file with the user's local full screen editor.

First receive the remote file to a local temporary file 'edit.tmp' file. Invoke the local full screen editor (using the LEDIT alias). When the user exits the editor, issue an "Update remote? " prompt to determine if the remote file should be updated with the edit changes. If the response is 'yes' (the default), send the temporary file back to the remote host replacing the original file. Then delete the 'edit.tmp' file.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-----------------|
| EDit | | remote_filename |

Where:

EDit (required) the verb for this command. The minimum spelling is ED.

remote_filename (required) the name of the remote file to be edited.

Examples

```
EFT> edit test.txt
```

Related Topics

ENCRYPT Alias Command

Description

The ENCRYPT alias command can be used to encrypt a user's password so users can avoid using clear-text passwords in EFT scripts and jobs.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-----------------------|
| ENCRypt | | [password] [username] |

Where:

ENCRypt (required) the verb for this command. The minimum spelling is ENC.

password (required) the password to encrypt.

username (optional) the name of the associated user. The username is used as an additional key to improve password encryption. **This is optional. Is omitted the encrypted password can be used by anyone. The value for "username" above must be entered in uppercase in order to match the username value later returned by MVS/zOS/Tandem and VMS.**

Examples

Example 1

For example, the ENCRYPT alias could be used to encrypt the same password 'COBRA' with the same secondary key 'myers' shown previously:

```
EFT> encrypt
Enter password? COBRA (password does not display)
Enter optional username (or '*'')? myers
EFT: The encrypted password is *249eece8e4203b189
```

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 resulting encrypted password is stored in a global variable PW for later reference.

Example 2

Encrypting Passwords Stored in a EFT Input Script File

Suppose a job running under the local UNIX username 'nscjones' inputs the EFT script 'zos5.ua' during program execution, and the script 'zos5.ua' contains the following line:

```
CONNECT zos5 admin7 secret
```

To avoid storing the password 'secret' in readable form in the script file, the password is encrypted by invoking the EFT client and using the ENCRYPT alias:

```
User> encrypt secret nscjones
User: The encrypted password is *26f17e2a4c9c65c56
```

Username 'nscjones' is specified because that is the local UNIX username under which the EFT job that uses the connect/login information will run. Using a local text editor, modify the input script 'zos5.ua' to look like:

```
CONNECT zos5 admin7 *26f17e2a4c9c65c56
```

Example 3

Using EFT 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 EFT CONNECT command to be later referenced by an EFT script. We can use the EFT OUTPUT command to generate the script file 'zos5.ua' to connect to the host 'zos5' as user 'admin7' with the password 'secret' (as shown in example #1):

```
User> encrypt secret nscjones
User: The encrypted password is *26f17e2a4c9c65c56
User> set output prefix
User> output zos5.ua
User> text CONNECT zos5 admin7 {pw}
User> output
```

The resulting file zos5.ua will contain the following line:

```
CONNECT zos5 admin7 *26f17e2a4c9c65c56
```

Related Topics

ENCRYPT string function

LDAP Alias Command

Description

Shorthand alias for LOCAL DIRECTORY.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-------------|
| LDir | | [directory] |

Where:

LDir (required) the verb for this command. The minimum spelling is LD.

directory (optional) the name of the local directory. If this argument does not exist the currently local directory is used.

Examples

```
EFT> ldir
Unix: total 74
Unix: drwxrwxrwt    7 sys      sys      358 May 17 03:30 .
Unix: drwxr-xr-x   33 root     root     1024 May 15 08:21 ..
Unix: drwxrwxr-x    2 root     root      104 May 15 13:12 .X11-pipe
Unix: drwxrwxr-x    2 root     root      104 May 15 13:12 .X11-unix
Unix: drwxrwxrwx    2 root     root      107 May 15 08:21 .pcmcia
Unix: drwxrwxrwt    2 root     root      207 May 15 08:21 .rpc_door
Unix: -rw-rw-r--    1 root     sys      7336 May 15 08:21 ps_data
```

Related Topics

LEDIT Alias Command

Description

Invoke the local full screen editor for the specified 'local_file'.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| LEdit | | local_file |

Where:

LEdit (required) the verb for this command. The minimum spelling is LED.

local_file (required) the name of the local file to be edited.

Examples

```
EFT> ledit test.txt
```

Related Topics

LOGIN Alias Command

Description

The LOGIN alias command is used to establish a connection to a remote host on the network. The host name specified must exist in the local network hosts database (either in the local hosts file, accessible through DNS, or in NetEx environments, the NETEX NCT file). The LOGIN alias command issues the EFT CONNECT command to establish the connection.

The LOGIN alias prompts for login information and issues a connect command to the specified host. Any missing parameters are prompted for. The password is captured in 'secure' mode so it doesn't display on the user's terminal.

Format

| Command | Qualifiers | Parameters |
|---------|------------|--|
| LOGIn | | [host] [username] [password] [qualifiers...] |

Where:

- LOGIN** (required) the verb for this command. The minimum spelling is LOG.
- host** (prompt) the name of the remote host computer to which the connection should be established
- username** (prompt) the name of the user to use for the login on the remote host computer. If this parameter is not specified, the login alias will prompt the user to enter a username.
- password** (prompt) the password to use for the username login sequence on the remote host computer. If this parameter is not specified, the login alias will prompt the user to enter the password.
- qualifiers** (prompt) additional qualifiers for the CONNECT command.

Examples

To establish a EFT session using the LOGIN alias command, issue the following commands:

```
EFT> login
Hostname? sunrise
Username? smith
Password? *****
Qualifiers?
sunrise>
```

Related Topics

- FTP Alias Command
- OPEN Alias Command
- CONNECT Command

RDIR Alias Command

Description

Shorthand alias for REMOTE DIRECTORY.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-------------|
| RDir | | [directory] |

Where:

RDir (required) the verb for this command. The minimum spelling is RD.

directory (optional) the name of the remote directory. If this argument does not exist, the current remote directory is used.

Examples

```
sunrise> rdir
Unix: total 658
Unix: drwxrwxrwt 7 sys sys 410 May 17 03:30 .
Unix: drwxr-xr-x 26 root root 1024 May 8 10:12 ..
Unix: drwxrwxr-x 2 root root 104 May 8 10:12 .X11-pipe
Unix: drwxrwxr-x 2 root root 104 May 8 10:12 .X11-unix
Unix: drwxrwxrwx 2 root root 107 May 8 10:12 .pcmcia
Unix: drwxrwxrwx 2 root other 69 May 8 10:14 .removable
Unix: drwxrwxrwt 2 root root 207 May 8 10:11 .rpc_door
Unix: -rw-rw-r-- 1 root sys 5584 May 8 10:11 ps_data
sunrise>
```

Related Topics

SET LOGIN Alias Command

Description

An alias of the SET CONNECT command.

Format

| Command | Qualifiers | Parameters |
|-----------|------------|-------------------|
| SEt Login | | qualifier [value] |

Where:

SEt LOGIN (required) the verb for this command. The minimum spelling is SE LO.

qualifier (required) the CONNECT command qualifier to modify.

value (optional) the value for the CONNECT qualifier, if necessary.

Examples

```
susel> show login
EFT:
EFT:  ACCount .....
EFT:  APPLication .....
EFT:  BLOCKsize ..... 16384
EFT:  COMmand .....
EFT:  INTerval ..... 5
EFT:  PASSword .....
EFT:  PROfile .....
EFT:  PROJect .....
EFT:  QUIet ..... off
EFT:  SCRipt .....
EFT:  SEArch ..... (SITE) (USER)
EFT:  SECondary .....
EFT:  SERvice ..... EFT
EFT:  SITE .....
EFT:  TIMEout ..... 120
EFT:  USERname .....
EFT:  VERBose ..... on
EFT:
susel> set login quiet on
susel> show login
EFT:
EFT:  ACCount .....
EFT:  APPLication .....
EFT:  BLOCKsize ..... 16384
EFT:  COMmand .....
EFT:  INTerval ..... 5
EFT:  PASSword .....
EFT:  PROfile .....
EFT:  PROJect .....
EFT:  QUIet ..... on
EFT:  SCRipt .....
EFT:  SEArch ..... (SITE) (USER)
```

```
EFT:  SECondary .....
EFT:  SERvice ..... EFT
EFT:  SITE .....
EFT:  TIMEout ..... 120
EFT:  USERname .....
EFT:  VERBose ..... on
EFT:
susel> sunrise>
```

Related Topics

CONNECT command
SET command

SHOW LOGIN Alias Command

Description

An alias for the SHOW CONNECT command.

Format

| Command | Qualifiers | Parameters |
|-------------|------------|------------|
| SHow LLogin | | |

Where:

SHow LOGIN (required) the verb for this command. The minimum spelling is SH LO.

Examples

```
susel> show login
EFT:
EFT:  ACCount .....
EFT:  APPLication .....
EFT:  BLOCKsize ..... 16384
EFT:  COMmand .....
EFT:  INTerval ..... 5
EFT:  PASSword .....
EFT:  PROFile .....
EFT:  PROJect .....
EFT:  QUIet ..... off
EFT:  SCRipt .....
EFT:  SEArch ..... (SITE) (USER)
EFT:  SECondary .....
EFT:  SERvice ..... EFT
EFT:  SITE .....
EFT:  TIMEout ..... 120
EFT:  USERname .....
EFT:  VERBose ..... on
EFT:
susel>
```

Related Topics

CONNECT command
SHOW command

SLD Alias Command

Description

A shorthand alias for the SET LOCAL DIRECTORY command.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-------------|
| SLD | | [directory] |

Where:

SLD (required) the verb for this command. The minimum spelling is SLD.

directory (optional) the name of the local directory.

Examples

```
susel> show local dir
EFT: DIRectory ..... /home/test1
susel> sld /var
susel> show local dir
EFT: DIRectory ..... /var
susel>
```

Related Topics

SET ALIAS command
SHOW ALIAS command

SRD Alias Command

Description

A shorthand alias for SET REMOTE DIRECTORY.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-------------|
| SRD | | [directory] |

Where:

SRD (required) the verb for this command. The minimum spelling is SRD.

directory (optional) the name of the remote directory.

Examples

```
susel> show remote directory
EFT: DIRectory ..... /etc
susel> srd /dev
susel> show remote directory
EFT: DIRectory ..... /dev
susel>
```

Related Topics

SET ALIAS command
SHOW ALIAS command

TEST Alias Command

Description

The TEST alias can be used with any of the numeric compare functions (EQ, NE, LE, LT, GE, GT) and the string compare functions (EQS, NES, CMP). All of these functions require two parameters. Function parameters CANNOT contain embedded blanks. String literal parameters are enclosed in double quotes.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-------------------------------|
| TEST | | param1 function param2 action |

Where:

TEST (required) the verb for this command. The minimum spelling is TES.

param1 (required) a string function, string variable, or string literal.

function (required) the string function to be performed.

param2 (required) a string function, string variable, or string literal.

action (required) the action to be taken. A EFT command or alias.

Examples

If the current year is 2013, print a text message:

```
EFT> TEST EXT(1),1,2 EQ 13 TEXT Welcome to 2013.
```

Related Topics

FTP Alias Commands

This section contains descriptions of the following FTP alias commands:

ACCOUNT
APPEND
ASCII
BIN
BYE
CD
CLOSE
DELETE
DIR
FTP
GET
LCD
LS
LSMem
MKDIR
OPEN
PUT
PWD
RENAME
RM
RMDIR

ACCOUNT Alias Command

Description

The ACCOUNT alias command issues ACCOUNT as a system command on the remote host computer to which there is an active connection. Command availability dependent on remote host type

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| ACCOUNT | | [string] |

Where:

ACCOUNT (required) the verb for this command. The minimum spelling is ACCOUNT.

string (optional) a string of text to be passed as input to the remote ACCOUNT system command. The format of the parameters for this command is dependent on the type of the remote host system. Refer to the remote user's guide section of the EFT manual for the remote system to which the command is being issued.

Examples

To issue the ACCOUNT command on the remote system:

```
EFT> ftp sunrise
User: smith
Password: *****
susel> account
```

Related Topics

- FTP Alias Command
- LOGIN Alias Command
- CONNECT Command

APPEND Alias Command

Description

The APPEND alias command sends the source file from the local host to the current active remote host and appends it to the 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 appended file is assumed to exist in the remote default directory.

If the destination parameter is not specified, a file by the same name as the source file name will be appended to in the remote default directory. If the destination parameter is specified and that remote file does not exist, the command will return an error.

The source file name may include the EFT wildcard characters * and ? as well as host specific wildcard characters where the two do not conflict. See “Source Wildcard Support for File Transfer” on [page](#) and “UNIX File Handling” on [page](#) for more details.

Format

| Command | Qualifiers | Parameters |
|---------|------------|----------------------|
| APPEND | | source [destination] |

Where:

APPEND (required) the verb for this command. The minimum spelling is APPEND.

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 file that is being appended to on the remote host.

Examples

To append the file *alpha.txt* from the current default local directory (C:\GUEST\SMITH\) to a file on the remote host, issue the following command:

```
susel> append alpha.txt
EFT: SOURCE                               DESTINATION                               SIZE
EFT: -----                               -----                               -----
EFT: C:\GUEST\SMITH\ALPHA.TXT             ALPHA.TXT                                 54909
```

Notice the entire source filename is displayed. The resulting destination file specification depends on the remote host to which the connection is made. If no destination name is specified, the source name is used to construct the destination name of a file 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.

Related Topics

FTP Alias Command
SEND Command

ASCII Alias Command

Description

The ASCII alias command sets the default SEND and RECEIVE transfer mode to CHARACTER.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| ASCII | | |

Where:

ASCII (required) the verb for this command. The minimum spelling is ASCII.

There are no qualifiers or parameters with this command.

Examples

To receive the ASCII file *alpha.txt* from host NT2 using the FTP aliases, issue the following commands:

```
EFT> ftp sunrise
User: smith
Password: *****
EFT: Connected to Service Initiator on host 'sunrise'.
=====
Last login: Fri May 18 10:21:51 from localhost
Sun Microsystems Inc. SunOS 5.7 Generic October 1998
=====
EFT: Logged in as user 'smith'.
EFT: Connected to service '33243' on host 'sunrise'.
suse1> ascii
suse1> get alpha.txt
EFT: SOURCE DESTINATION SIZE
EFT: -----
EFT: /home/guest/alpha.txt /home/smith/alpha.txt 54909
suse1> bye
```

Related Topics

- FTP Alias Command
- GET Alias Command
- PUT Alias Command
- APPEND Alias Command
- SEND Command
- RECEIVE Command

BIN Alias Command

Description

The BIN alias command sets the default SEND and RECEIVE transfer mode to STREAM.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| BIN | | |

Where:

BIN (required) the verb for this command. The minimum spelling is BIN.

There are no qualifiers or parameters with this command.

Examples

To send the BINARY file *test.exe* to host NT3 using the FTP aliases, issue the following commands:

```
EFT> ftp sunrise
User: smith
Password: *****
EFT: Connected to Service Initiator on host 'sunrise'.
=====
Last login: Fri May 18 10:21:51 from localhost
Sun Microsystems Inc. SunOS 5.7 Generic October 1998

=====
EFT: Logged in as user 'smith'.
EFT: Connected to service '33243' on host 'sunrise'.
suse1> bin
suse1> put test.exe
EFT: SOURCE DESTINATION SIZE
EFT: -----
EFT: /home/guest/text.exe /home/smith/test.exe 193000
suse1> bye
```

Related Topics

- FTP Alias Command
- GET Alias Command
- PUT Alias Command
- APPEND Alias Command
- SEND Command
- RECEIVE Command

BYE Alias Command

Description

The BYE alias command causes EFT to terminate.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| BYE | | |

Where:

BYE (required) the verb for this command. The minimum spelling is BYE.

There are no qualifiers or parameters with this command.

Examples

To terminate EFT by using the FTP aliases, issue the *bye* command:

```
EFT> ftp sunrise
User: smith
Password: *****
EFT: Connected to Service Initiator on host 'sunrise'.
=====
Last login: Fri May 18 10:21:51 from localhost
Sun Microsystems Inc. SunOS 5.7 Generic October 1998
=====
EFT: Logged in as user 'smith'.
EFT: Connected to service '33243' on host 'sunrise'.
suse1> bye
$
```

Related Topics

FTP Alias Command
EXIT Command

CD Alias Command

Description

The CD alias command sets the default remote directory on the remote host computer to which there is an active connection, to the directory name specified by the command.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-------------|
| CD | | [directory] |

Where:

CD (required) the verb for this command. The minimum spelling is CD.

directory (optional) the name of the remote directory to set as the default.

Note: the exact format of the CD command is dependent on the type of the remote host computer (e.g., MVS, Unix, NT, etc).

Examples

During a EFT session to host NT3, to set the default remote directory to C:\JONES\TEST, issue the following FTP alias commands:

```
EFT> cd /home/jones/test
```

Related Topics

- FTP Alias Command
- LCD Alias Command

CLOSE Alias Command

Description

The CLOSE alias command terminates the connection from the remote host computer to which there is an active connection.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| CLOSE | | |

Where:

CLOSE (required) the verb for this command. The minimum spelling is CLOSE.

There are no qualifiers or parameters with this command.

Examples

To terminate a EFT session using the FTP aliases, issue the *close* command:

```
EFT> ftp sunrise
User: smith
Password: *****
EFT: Connected to Service Initiator on host 'sunrise'.
=====
Last login: Fri May 18 10:21:51 from localhost
Sun Microsystems Inc. SunOS 5.7 Generic October 1998
=====
EFT: Logged in as user 'smith'.
EFT: Connected to service '33243' on host 'sunrise'.
susel> ftp sunrise
User: smith
Password: *****
sunrise> close (note: terminates connection to sunrise)
EFT> set host sunrise
susel> close (note: terminates connection to sunrise)
EFT>
```

Related Topics

FTP Alias Command
DISCONNECT Command

DELETE Alias Command

Description

The DELETE alias command issues the system DELETE command on the remote host to which there is an active connection. The filename specified by the command is deleted.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| DELETE | | filename |

Where:

DELETE (required) the verb for this command. The minimum spelling is DELETE.

filename (required) the name of the file on the remote host computer to delete.

Note: the exact format of the DELETE command is dependent on the type of the remote host computer (e.g. MVS, Unix, NT, etc).

Examples

During a EFT session to host NT3, to delete the file *test.exe* in directory C:\JONES, issue the following FTP alias commands:

```
EFT> ftp sunrise
User: smith
Password: *****
EFT: Connected to Service Initiator on host 'sunrise'.
=====
Last login: Fri May 18 10:21:51 from localhost
Sun Microsystems Inc. SunOS 5.7 Generic October 1998
=====
EFT: Logged in as user 'smith'.
EFT: Connected to service '33243' on host 'sunrise'.
susel> delete test.txt
```

Related Topics

FTP Alias Command
CD Alias Command

DIR Alias Command

Description

The DIR alias command issues DIRECTORY as a system command on the remote host computer to which there is an active connection. The contents of the directory specified by the command are displayed.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-------------|
| DIR | | [directory] |

Where:

DIR (required) the verb for this command. The minimum spelling is DIR.

directory (optional) the name of the directory to display. If not specified, the current working directory is displayed.

Note: The exact format of the DIR command is dependent on the type of the remote host computer (e.g. MVS, Unix, NT, etc).

Examples

During an EFT session to host suse1, to display all the files that begin with 'd', issue the following FTP alias commands:

```
EFT> ftp suse1
User: test1
Password:
EFT: Connected to Service Initiator on host 'suse1'.
=====
Last login: Mon Oct 7 10:28:14 CDT 2013 from suse1 on pts/15
10:44:55: license expiration warning - expiring 20131026 - not functional
10:44:55: 20131026 (UA-4141).
=====
EFT: Logged in as user 'test1'.
EFT: Connected to service '3000' on host 'suse1'.
suse1> dir d*
Unix: -rw-r--r-- 1 test1 users 10192446 2013-10-01 15:12 dickens
suse1> bye
```

Related Topics

FTP Alias Command
CD Alias Command

FTP Alias Command

Description

The FTP alias command is used to establish a connection to a remote host on the network. The host name specified must exist in the local network “hosts” database (either in the local hosts file, accessible through DNS, or described in the NCT file). The FTP alias command issues the EFT CONNECT command to establish the connection. Refer to the “CONNECT Command” on **page** for a description of CONNECT processing.

Format

| Command | Qualifiers | Parameters |
|---------|------------|----------------------------|
| FTP | | host [username] [password] |

Where:

- FTP** (required) the verb for this command. The minimum spelling is FTP.
- host** (required) the name of the remote host computer to which the connection should be established
- username** (prompt) the name of the user to use for the login on the remote host computer. The user will be prompted for this if it is not specified on the command line.
- password** (prompt) the password to use for the username login sequence on the remote host computer. The user will be prompted for this if it is not specified on the command line.

Examples

To establish a EFT session using the FTP alias commands, issue the following commands:

```
EFT> ftp nt2
User: smith
Password: *****
nt2>
```

Related Topics

- LOGIN Alias Command
- OPEN Alias Command
- CONNECT Command

GET Alias Command

Description

The GET alias 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 EFT wildcard characters * and ? as well as host specific wildcard characters where the two do not conflict. See the discussion on EFT wildcarding in “Source Wildcard Support for File Transfers” on [page](#) for further details.

Format

| Command | Qualifiers | Parameters |
|---------|------------|----------------------|
| GET | | source [destination] |

Where:

- GET** (required) the verb for this command. The minimum spelling is GET.
- 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

To receive the file *alpha.txt* from host NT2 using the FTP aliases, issue the following commands:

```
EFT> ftp nt2
User: smith
Password: *****
nt2> get alpha.txt
EFT: SOURCE                               DESTINATION           SIZE
EFT: -----
EFT: C:\GUEST\ALPHA.TXT                   C:\SMITH\ALPHA.TXT   54909
nt2> bye
```

Related Topics

- FTP Alias Command
- PUT Alias Command
- RECEIVE Command

LCD Alias Command

Description

The LCD alias command sets the default local directory on the local host computer, to the name specified by the command.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| LCD | | directory |

Where:

LCD (required) the verb for this command. The minimum spelling is LCD.

directory (required) the name of the local directory to set as the default.

Examples

During a EFT session to host SUNRISE, to set the default local directory to C:\SMITH\TEST, issue the following FTP alias commands:

```
EFT> ftp sunrise
User: smith
Password: *****
sunrise> lcd C:\SMITH\TEST
```

Related Topic

FTP Alias Command

CD Alias Command

LS Alias Command

Description

The LS alias command issues DIRECTORY as a system command on the remote host computer to which there is an active connection. The contents of the directory specified by the command are displayed.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-------------|
| LS | | [directory] |

Where:

LS (required) the verb for this command. The minimum spelling is LS.

directory (optional) the name of the directory to list.

Note: The exact format of the LS command is dependent on the type of the remote host computer (e.g., MVS, Unix, NT, etc).

Examples

During a EFT session to host NT3, to list the contents of directory C:\JONES, issue the following FTP alias commands:

```
EFT> ftp nt3
User: smith
Password: *****
nt3> cd c:\jones
nt3> ls
nt3> bye
```

Related Topics

- DIR Alias Command
- CD Alias Command
- LCD Alias Command

LSMem Alias Command

Description

The LSMem alias command issues a remote directory command on the remote host computer to which there is an active connection. The contents of the directory specified by the command are displayed.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-------------|
| LSMem | | [directory] |

Where:

LSMem (required) the verb for this command. The minimum spelling is LSM.

directory (optional) the name of the directory to display.

Note: The exact format of the LSMEM command is dependent on the type of the remote host computer (e.g., MVS, Unix, NT, etc).

Examples

During a EFT session to host NT3, to list the contents of directory C:\JONES\TEST, issue the following FTP alias commands:

```
EFT> ftp nt3
User: smith
Password: *****
nt3> lsm c:\jones\test
nt3> bye
```

Related Topics

- DIR Alias Command
- CD Alias Command
- LCD Alias Command

MKDIR Alias Command

Description

The MKDIR alias command issues the mkdir command on the remote host computer to which there is an active connection. A new directory is created in the current working directory, having a name specified by the command.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| MKDIR | | directory |

Where:

MKDIR (required) the verb for this command. The minimum spelling is MKDIR.

directory (required) the name of the directory to create.

Note: The exact format of the MKDIR command is dependent on the type of the remote host computer (e.g., MVS, Unix, NT, etc.).

Examples

During a EFT session to host NT3, create a new directory in C:\JONES, called TEST, by issuing the following FTP alias commands:

```
EFT> ftp nt3
User: smith
Password: *****
nt3T> cd C:\JONES
nt3> mkdir test
nt3> bye
```

Related Topics

- DIR Alias Command
- CD Alias Command
- LCD Alias Command

OPEN Alias Command

Description

The OPEN alias command is used to establish a connection to a remote host on the network. The host name specified must exist in the local network “hosts” database (either in the local hosts file, accessible through DNS, or defined in the NCT). The OPEN alias command issues the EFT CONNECT command to establish the connection. Refer to the “CONNECT Command” on [page](#) for a description of CONNECT processing.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------------------------|
| OPEN | | [host] [username] [password] |

Where:

- OPEN** (required) the verb for this command. The minimum spelling is OPEN.
- host** (required) the name of the remote host computer to which the connection should be established
- username** (required) the name of the user to use for the login on the remote host computer. If this parameter is not specified, the ftp alias will prompt the user to enter a username.
- password** (required) the password to use for the username login sequence on the remote host computer. If this parameter is not specified, the ftp alias will prompt the user to enter the password.

Examples

To establish a EFT session using the FTP alias commands, issue the following commands:

```
EFT> ftp
EFT> open
Hostname: nt3
User: smith
Password: *****
nt3>
```

To specify parameters on the “open” command line, issue the following commands:

```
EFT> ftp
EFT> open nt3 smith
Password: *****
nt3>
```

Related Topics

- LOGIN Alias Command
- FTP Alias Command
- CONNECT Command

PUT Alias Command

Description

The PUT alias 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 EFT wildcard characters * and ? as well as host specific wildcard characters where the two do not conflict. See “Source Wildcard Support for File Transfers” on [page](#) and “UNIX File Handling” on [page](#) for more details.

Format

| Command | Qualifiers | Parameters |
|---------|------------|----------------------|
| PUT | | source [destination] |

Where:

- PUT** (required) the verb for this command. The minimum spelling is PUT.
- source** (required) the file specification for the file on the local host that you intend to send.
- destination** (optional) the file specification for the file on the remote host.

Examples

To send the file *alpha.txt* from host NT2 to host NT3 using the FTP aliases, issue the following commands:

```
EFT> ftp nt3
User: smith
Password: *****
nt3> put alpha.txt
EFT: SOURCE                               DESTINATION           SIZE
EFT: -----
EFT: C:\GUEST\ALPHA.TXT                   C:\SMITH\ALPHA.TXT   54909
nt3> bye
```

Related Topics

- FTP Alias Command
- GET Alias Command
- SEND Command

PWD Alias Command

Description

The PWD alias command issues the SHOW REMOTE DIRECTORY command on the remote host computer to which there is an active connection. On remote hosts that support this command, the path to the current working directory is displayed.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| PWD | | |

Where:

PWD (required) the verb for this command. The minimum spelling is PWD.

There are no qualifiers or parameters with this command.

Examples

To establish a connection to remote host NT3, and display the current path information, issue the following FTP alias commands:

```
EFT> ftp nt3
User: smith
Password: *****
nt3> pwd
nt3>
```

Related Topics

FTP Alias Command

CD Alias Command

RENAME Alias Command

Description

The RENAME alias command issues the RENAME system command on the remote host computer to which there is an active connection. The file specified by “file1” is renamed to “file2”.

Format

| Command | Qualifiers | Parameters |
|---------|------------|-------------|
| RENAME | | file1 file2 |

Where:

RENAME (required) the verb for this command. The minimum spelling is RENAME.

file1 (required) the name of an existing file to rename.

file2 (required) the new name for the file.

Note: the exact format of the RENAME command is dependent on the type of the remote host computer (e.g., MVS, Unix, NT, etc).

Examples

To establish a connection to remote host NT3, and rename *alpha.txt* to *test.txt*, issue the following FTP alias commands:

```
EFT> ftp nt3
User: smith
Password: *****
nt3> rename alpha.txt test.txt
nt3>
```

Related Topics

FTP Alias Command

RM Alias Command

Description

The RM alias command issues the DELETE system command on the remote host computer to which there is an active connection. The filename specified by the command is deleted.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| RM | | filename |

Where:

RM (required) the verb for this command. The minimum spelling is RM.

filename (required) the name of the file to delete.

Note: the exact format of the RM command is dependent on the type of the remote host computer (e.g., MVS, Unix, NT, etc).

Examples

To establish a connection to remote host NT3, and delete *test.txt*, issue the following FTP alias commands:

```
EFT> ftp nt3
User: smith
Password: *****
nt3> rm test.txt
nt3>
```

Related Topics

FTP Alias Command

RMDIR Alias Command

Description

The RMDIR alias command issues the RMDIR system command on the remote host computer to which there is an active connection. The directory specified by the command is deleted.

Format

| Command | Qualifiers | Parameters |
|---------|------------|------------|
| RMDir | | directory |

Where:

RMDIR (required) the verb for this command. The minimum spelling is RMD.

directory (required) the name of the directory to delete.

Note: the exact format of the RM command is dependent on the type of the remote host computer (e.g., MVS, Unix, NT, etc).

Examples

To establish a connection to remote host NT3, and delete the *alpha* directory, issue the following FTP alias commands:

```
EFT> ftp nt3
User: smith
Password: *****
nt3> rmdir alpha
nt3>
```

Related Topics

- FTP Alias Command
- MKDIR Alias Command

Installation Procedures

Prerequisites

The following are prerequisites for installing eFTxx3 eFT:

- The privileges to create directories, load files from the distributed software media, create links, edit system startup, and system login file.
- Only if installing eFTxx3N product, an installed and operational Hxx0IP NetEx product.

Pre-Installation

The location of the executable programs from your previous installation of eFT will be required for the Installation section.

The previous installation of eFT will have to be de-installed prior to installation.

Accessing the UNIX eFT software distribution

The eFT software is available as an RPM which may be downloaded from NESi. Contact NESi Customer Support to request the download link.

Getting the NESi Public Key

The RPM software distribution package is signed to ensure integrity and authenticity. It is recommended to install the NESi public key and verify the signature of any software packages before installation.

You can download the key from the relevant documentation page for your eFT product at <http://www.netex.com/>.

Importing the NESi Public Key

Install the public key as super user with the command:

```
# rpm --import RPM-GPG-KEY-netex.txt
```

Verifying Signatures

You can verify the RPM signature to ensure that a package has not been modified since it has been signed. Verification will also check that a package is signed by the vendors or packagers key.

To verify the signature, use the `-K` or `--checksig` option to the rpm command:

```
# rpm -K eFTxx3...rpm
```

Installation

This eFT installation uses the RPM package management utility. eFT installs in the same fashion as all other RPM-based software distributions.

A link to the distribution can be obtained from NESi Support. The installation steps are:

Execute RPM as 'root' to install or upgrade the software:

If this is an initial installation, install the software as super user with the command:

```
# yum install eFTxx3...rpm
```

or

```
# rpm -i eFTxx3...rpm
```

If the NESi public key has not been installed use the command:

```
# yum --nogpgcheck install eFTxx3...rpm
```

or

```
# rpm -i --nosignature eFTxx3...rpm
```

Customers do NOT have the option of installing eFTxx3 into a user-defined directory location.

If eFT or its executables are not in the PATH, then hard coded paths to the executables will have to be used in all eFT jobs/scripts. This can become a significant maintenance problem for the users in the future. However if this is how all your existing eFT jobs are setup you could add symbolic links from the current location to the path specified in your jobs.

Upgrading eFTxx3

If you are upgrading an existing installation of eFT, it is strongly recommended that any running eFT (and NetEx/IP if running in a NetEx environment) process be stopped. (All versions older than 5.4 of eFT should be uninstalled using the uninstall procedure defined in that version.) Using the "rpm -U" command preserves any customized files in this package and the replacement files are installed with extensions of .rpmnew. Any files that are not in the package but in package directories will also be preserved. Upgrade the software as super user with the command:

```
# yum upgrade eFTxx3...rpm
```

or

```
# rpm -U eFTxx3...rpm
```

If the NESi public key has not been installed use the command:

```
# yum --nogpgcheck upgrade eFTxx3...rpm
```

or

```
# rpm -U --nosignature eFTxx3...rpm
```

Customers do NOT have the option of installing eFTxx3 into a user-defined directory location.

If eFT or its executables are not in the PATH, then hard coded paths to the executables will have to be used in all eFT jobs/scripts. This can become a significant maintenance problem for the users in the future. However if this how all your existing eFT scripts are setup you will have to add symbolic links from the current location to the path specified in your scripts.

Removing eFTxx3

During RPM removal, any customized files and log files will not be deleted. Remove the software as super user with the command:

```
# yum erase eFTxx3
```

or

```
# rpm -e eFTxx3
```

Removing the NESi Public Key

To remove the NESi public key, as super user issue the command:

```
# rpm -e gpg-pubkey-3d6b35d3-51bb5907
```

Starting, Stopping & Verifying the eFT Installation

For Unix/Linux systems:

The service command should be used to stop & start eFT:

```
# service eftip stop
# service eftip start
# service eftip restart
```

The chkconfig command should be used to verify installation:

```
# chkconfig --list eftip
```

For AIX:

The startsrc command or SMIT should be used to start eFT:

```
# startsrc -s eftip
```

The stopsrc command or SMIT should be used to stop eFT:

```
# stopsrc -s eftip
```

The lssrc & lsitab commands or SMIT can be used to verify installation:

```
# lssrc -S -s eftip
# lsitab eftip
```

Completion and Testing of Installation

The eFT installation can be verified by running a simple test script called `verify.ua`. `eftip-server` must already be running.

Now invoke the eFT test script:

```
# eftip-client /usr/share/nesi/eftip/user/verify.ua
```

The script will take over from this point. It will give you directions and prompts for any parameters it needs to complete a simple test. If for some reason the test fails, use the error information and the eft logs to diagnose the problem.

Appendix A. Updating the TCP/IP Network Control Files

The */etc/services* network control file can be updated to change the EFT default. If Network Information Service (NIS - formerly Yellow Pages) is active, update the equivalent files on the master NIS server and 'make' the changes.

In the */etc/services* file, add a TCP/IP 'eft' service. You must first select a unique port number. Port numbers in the range 0-5000 are reserved for TCP network services. EFT uses port number 6900 as the default. For example, add the following line to the end of the file:

```
eft          6900/tcp      EFT
```


Appendix B. Stopping the Service Initiator

The Service Initiator can be stopped by using the Service Initiator Control program . Generally, the command to stop the Service Initiator is:

```
$ /usr/share/nesi/eftip/si/control STOP -H host -S service
```

where ‘host’ is the name of the local network host and ‘service’ is the name of a network service being offered by the Service Initiator (“EFT” is generally the service name being offered). The optional OPERATOR password may need to be specified with the “-O” option. Stopping the Service Initiator this way will clean up /etc/utmp entries without affecting the associated servers that are still running. Stopping the Service Initiator with ‘kill’ will cause those /etc/utmp entries to remain in the file until re-boot or until they are re-used.

The log file may be examined by issuing the following, while the Service Initiator is active:

```
$ tail /usr/share/nesi/eftip/si/svcinit.log
```


Appendix C. Service Initiator Keywords

The Service Initiator startup input file `/usr/share/nesi/efitp/si/svcinit.conf` contains a list of keywords that can be set to alter the way the Service Initiator operates for a given SERVICE being offered.

SERVER

Specifies the command that is used to invoke or run the EFT server.

-PORT_MIN

If ephemeral port usage needs to be restricted (i.e. open up firewall), then this parameter and `port_max` must be set to non-zero. Minimum ephemeral port number; when not set or set to 0, the system ephemeral port numbers are used. DEFAULT: 0

-PORT_MAX

If ephemeral port usage needs to be restricted (i.e. open up firewall), then this parameter and `port_min` must be set to non-zero. Maximum ephemeral port number; when not set or set to 0, the system ephemeral port numbers are used. DEFAULT: 0

OPERATOR

Specifies a password that is required when issuing commands through the CONTROL program (used for shutting down the Service Initiator).

LOGTIMEOUT

Specifies the login timeout in seconds. This is used to terminate a login request that for some unknown reason, is hanging around.

MINIMUM

Specifies the minimum session number that will be offered for this service. For example, a MINIMUM value of 5 would result in SERVICE "EFT" being offered as "EFT005" up to MAXIMUM (below). Specifying MINIMUM or MAXIMUM do nothing to limit the number of EFT sessions that are allowed. They are simply provided as a naming tool.

MAXIMUM

Specifies the maximum session number that will be offered for this service. A value of 30 for example would result in the last offer of "USER" being "USER030", before the offers started over at MINIMUM.

TRACE

Allows different levels of tracing of the Service Initiator. Refer to the `svcinit.conf` file `/usr/nesi/efitp/si/svcinit.conf` for a description of the different trace levels.

UNIX_LOGIN

This keyword and the two that follow (`UNIX_PASSWD` and `UNIX_BADLOG`) are to be modified only if the site has made modifications to the default UNIX login utility `/bin/login`. These three keywords define strings that the Service Initiator looks for when interacting with `/bin/login` during an EFT login request. `UNIX_LOGIN` defines the first prompt used by `/bin/login` when it is looking for a user name. The default value is the string "login: ".

UNIX_PASSWD

(see `UNIX_LOGIN` above). Defines the prompt used by `/bin/login` when it is looking for a password. The default value is the string "Password: ".

UNIX_BADLOG

(see UNIX_LOGIN above). Defines the resulting string returned by /bin/login when an invalid login attempt has been made. The default value is the string "Login incorrect".

GATEWAY

Enables/disables gateway processing of remote NODE qualifier. This should be 'on' for gateway hosts, 'off' for all others. DEFAULT: 'off'

SIHELPER

Service name of the remote 'sihelper' process or NULL to disable SIHELPER support. DEFAULT: 'sihelper' to enable SIHELPER support.

SIPLUS

Enables/disables SI+ proxy connect support. If 'on', the standard connect request is forwarded to the remote NODE. DEFAULT: 'on'

RLOGIN

Enables/disables 'rlogin' to remote NODE. If 'on', the Unix 'rlogin' mechanism is used to connect to the remote NODE. DEFAULT: 'off'

SIP_TIMEOUT

Connect timeout to use for SI+ proxy connects. This should be short to prevent long delays if the requested service is not offered on the remote NODE. DEFAULT: 1 sec.

SIP_INTERVAL

Connect interval to use for SI+ proxy connects. See also SIP_TIMEOUT. DEFAULT: 1 sec.

LOGINTYPE

The method used to log in to the system on which the Service Initiator is running. Valid values are 'rlogin' and 'login'. If 'rlogin' is specified, the rlogin protocol is used to log in to LOGINHOST. The 'login' method will allocate a pty, fork a process, invoke login, and invoke SERVER. DEFAULT: 'login'

LOGINHOST

Host name or IP address to log in to when using LOGINTYPE 'rlogin'. DEFAULT: 'localhost'

OFFER_TIMEOUT

In a NETEX environment, only: length of time (in seconds) to offer before timing out. If the offer times out, it is automatically reissued. DEFAULT: 0 (offer indefinitely).

Appendix D. Product Configuration File

Each Network Executive Software product that has integrated the License Verification Software facility now contains a product configuration file. The EFT product includes this file in its installation distribution file set. During installation, this file (*prodconf*) is installed in the USER subdirectory of the installation directory (*/usr/share/nesi/eftip/user*).

The default *prodconf* file shipped with EFT contains the following information:

```
# Network Executive Software, Inc.  
# EFT Product Configuration file  
LICPATH      /usr/share/nesi/NESikeys    # NESi License Key file
```

A '#' character denotes a comment. Non-comment records must contain a keyword/value string. The initial version of this file contains only a single keyword/value record:

- LICPATH – this keyword defines the full path to the NESi License Key File (see “Appendix E. License Key File”).

Appendix E. License Key File

A single NESi License Key file must reside on each system on which one or more NESi products containing license support will be installed. The following guidelines apply:

- The default file name is *NESikeys*.
- The LICPATH keyword/value pair in the product configuration file (see “Appendix D. Product Configuration File”) specifies the full path name to this file. The default is: */usr/share/nesi/NESikeys* .
- The systems programmer installing the software must initially create this file prior to invocation of EFT components such as Service Initiator and Client.
- A leading ‘#’ character in a file record denotes a comment line.
- The systems programmer installing this software must edit this file to add a new encrypted Software Key each time such a key is obtained from NESi for a . This should be done prior to installing the product, and must be done prior to being able to run the product successfully. After the product has been started with a valid license key, new keys may be added to the top of the file and the product will utilize the first valid key when necessary.
- The file may contain multiple keys per product due to new product releases or a change to the platform’s fingerprint (on UNIX this corresponds to the hostname for the target host). To make the file easier to maintain over time, it is recommended that you add newer keys to the top and precede each key entry with a comment line that documents the product and release level of the product that the key is associated with. It will then be easier to delete older keys that are no longer valid for the product.
- The following shows an example of what a NESikeys file might look like after adding several keys to the file:

```
# Network Executive Software, Inc. Software License Key file
# eFT803IP 5.4 (exp. 2013-07-01)
CNBJ-YAA6-ABAA-DBPH-Q3QB-3KXJ

# eFT803IP 5.4 (exp. 2013-12-31)
CP4Z-YAAA-ABAA-CFK7-OPAV-OQZL
```


Appendix F. EFT Error Messages for UNIX

This appendix is intended to give users more information about EFT messages that may be seen during a session. All of the EFT 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 EFT messaging scheme is designed to generate various levels of messages which is why a single erroneous condition may result in two, three, or even four 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 alone, 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 (EFT host independent Message), eFTxx3 (EFT eFTxx3 UNIX host dependent message), SI (EFT Service Initiator), or SIxx3 (EFT eFTxx3 UNIX Service Initiator). If a message contains an eFTxx3, SIxx3, or some facility other than these (e.g. UA213 or MUX), this message must be looked up in the appropriate EFT manual for that host (e.g. eFTxx3 EFT).⁷

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 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:

- 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. In a NETEX environment, the user should refer to the accompanying NETEX message that is displayed for a further explanation of the prob-

⁷ EFT identifies a host by its facility number. Under UNIX, this number varies depending on the host that is running EFT. For example, if a Solaris system with UNIX is running EFT the facility would be ‘673’.

Although there are many different hosts that run UNIX EFT, because of similarity the manuals were combined into one. Now, the manual is identified by: eFTxx3 EFT for UNIX. Therefore, some of the following error messages will be identified by ‘eFTxx3’ as the facility from which the error came. As in the previous example, if a Solaris system with UNIX was running USER-Access’s and a host specific error occurred, the facility from EFT would read ‘UA673’, not ‘eFTxx3’ as indicated in the manual.

lem. These messages should be logged and reported to the system administrator unless it is obvious that the NETEX condition is temporary and for a known reason. These messages could possibly be a sign of a network interruption of some kind.

R Re-triable error condition. The command used to generate this error can be re-tried at some later time without fear of a fatal condition occurring.

TEXT The message text.

| Table 6. 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' |

| Table 6. Error Messages | | | | |
|--------------------------------|-------------|------------|----------------|--|
| FAC | Code | SEV | Comment | Text |
| UA | 809 | E | N,R | READ timed out after NNN |
| 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 |
| 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 sec 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 (PIFD) 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) (In a NETEX environment only) |
| UA | 4003 | W | | Alias 'SSS' is not defined |
| UA | 4102 | E | | BLOCKSIZE of NNN is out of range (NNN-NNN0) |
| 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 |

| Table 6. Error Messages | | | | |
|--------------------------------|-------------|------------|----------------|--|
| FAC | Code | SEV | Comment | Text |
| | | | | 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 | M | 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 |
| 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 (In a NETEX environment only) |
| UA | 4132 | E | A | Restricted command in server startup file |
| UA | 4133 | E | A,D | ProdConfRead Error |
| UA | 4134 | E | A,D | EFTUTL1 Error: CPU Serial Number could not be obtained. |
| UA | 4135 | E | A,D | License Verify Error |
| UA | 4136 | W | A,D | License Expiring Warning |
| UA | 4137 | E | A,D | License VerifyCapability Error |
| UA | 4138 | W | A,D | License VerifyCapability Warning |
| UA | 4139 | W | A | License Expired and Product will soon be non-operational |
| UA | 4140 | E | A | License Expired and Product is not operational |
| UA | 4201 | E | | Missing SSS parameter |

| Table 6. Error Messages | | | | |
|--------------------------------|-------------|------------|----------------|---|
| FAC | Code | SEV | Comment | Text |
| 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 |
| 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 | 4710 | E | | Cmd not supported for full screen emulation |
| UA | 4802 | E | D | Missing MAXRECORD specification |
| UA | 4803 | E | D | MAXRECORD (NNN) greater than maximum |

| Table 6. Error Messages | | | | |
|--------------------------------|-------------|------------|----------------|---|
| FAC | Code | SEV | Comment | Text |
| | | | | (NNN) |
| UA | 4804 | E | A | MAXRECORD (NNN+NNN) too large for BLOCKSIZE (NNN) |
| UA | 4807 | E | D | Invalid datamode (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 | 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 |
| 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 |

| Table 6. Error Messages | | | | |
|--------------------------------|-------------|------------|----------------|---|
| FAC | Code | SEV | Comment | Text |
| | | | | 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 | 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 |
| 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 (In a NETEX environment only) |
| UA | 5605 | E | D,N | Invalid translate code table length NNN |
| UA | 5306 | E | A | Incomplete ARCHIVE file – missing end-of-file |
| UA | 5606 | E | N | Failed on binary read of NETEX code table (NNN) (In a NETEX environment only) |
| UA | 5607 | E | N | Failed to exchange TRANSLATE information |
| UA | 5608 | Q | | Invalid TRANSLATE value 'SSS' |

| Table 6. Error Messages | | | | |
|--------------------------------|-------------|------------|----------------|---|
| FAC | Code | SEV | Comment | Text |
| 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' |
| EFTxx3 | 2002 | E | A,N,R | Data checksum (CRC) error at block NNN |
| EFTxx3 | 2004 | E | R | Sequence number error at block NNN |
| EFTxx3 | 2101 | E | A | Failed to allocate NNN bytes of dynamic memory |
| EFTxx3 | 8001 | E | | Keyboard interrupt |
| EFTxx3 | 8002 | E | A | File size limit exceeded |
| EFTxx3 | 8201 | E | D | Missing execute SHELL |
| EFTxx3 | 8202 | E | H | Failed to create execute pipe |
| EFTxx3 | 8203 | E | H | Failed to fork a child process |
| EFTxx3 | 8204 | E | | Command output line is longer |
| EFTxx3 | 8205 | E | H | Failed to get execute output line |
| EFTxx3 | 8301 | E | D | Missing SOURCE |
| EFTxx3 | 8302 | E | H | Failed to access file 'SSS' |
| EFTxx3 | 8303 | E | H | Failed to get status for file 'SSS' |
| EFTxx3 | 8304 | E | | File 'SSS' is a directory |
| EFTxx3 | 8307 | E | | Missing SOURCE file name |
| EFTxx3 | 8303 | E | | File 'SSS' already exists with -CREATE NEW option |
| EFTxx3 | 8309 | E | H | Failed to delete BACKUP file 'SSS' |
| EFTxx3 | 8310 | E | H | Failed to link 'SSS' to 'SSS' |
| EFTxx3 | 8311 | E | H | Failed to delete file 'SSS' |
| EFTxx3 | 8312 | E | H | Failed to create file 'SSS' |
| EFTxx3 | 8318 | E | H | Failed to CLOSE SSS |
| EFTxx3 | 8319 | E | A | Bad CHARACTER data |

| Table 6. Error Messages | | | | |
|--------------------------------|-------------|------------|----------------|---|
| FAC | Code | SEV | Comment | Text |
| EFTxx3 | 8320 | E | | Character record longer than NNN bytes |
| EFTxx3 | 8321 | E | H | Failed to get input from terminal |
| EFTxx3 | 8322 | E | H | File read of NNN bytes failed |
| EFTxx3 | 8324 | E | H | File write of NNN bytes failed |
| EFTxx3 | 8329 | E | A | Invalid RECORD file format |
| EFTxx3 | 8330 | E | | RECORD length NNN exceeds MAXRECORD of NNN |
| EFTxx3 | 8331 | E | | Bad RECORD read – requested NNN bytes – got NNN |
| EFTxx3 | 8334 | E | D | Transfer mode ‘NNN’ is not supported |
| EFTxx3 | 8336 | E | A | Unknown archive mode tag field fflSSS” |
| EFTxx3 | 8337 | E | A | Invalid wildcard specified ‘SSS’ |
| EFTxx3 | 8338 | E | | Bad ARCHIVE record length NNN |
| EFTxx3 | 8339 | E | | Invalid ARCHIVE tag field [SSS] |
| EFTxx3 | 8603 | E | H | Invalid directory ‘SSS’ |
| EFTxx3 | 8604 | E | | Failed to get working directory – SSS |
| EFTxx3 | 8701 | E | | Missing DESTINATION specifier |
| EFTxx3 | 8702 | I | | Piped command failed with an exit status of SSS |
| EFTxx3 | 8703 | E | H | Failed to create named pipe ‘SSS’ |
| EFTxx3 | 8704 | E | H | Failed to open named pipe ‘SSS’ |
| EFTxx3 | 8903 | E | | Remote BATCH is not supported |
| 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’ |
| SIxx3 | 8001 | E | | Missing USERNAME |
| SIxx3 | 8002 | E | | Login failed – invalid user name |
| SIxx3 | 8003 | E | | Login failed – invalid password |
| SIxx3 | 8005 | E | H | Failed to create pipes |
| SIxx3 | 8006 | E | H | Login failed – can’t fork child process |
| SIxx3 | 8007 | E | | Server ‘SSS’ failed to respond |

| Table 6. Error Messages | | | | |
|--------------------------------|-------------|------------|----------------|-----------------------------------|
| FAC | Code | SEV | Comment | Text |
| SIxx3 | 8008 | E | A | Login exceeded NNN second timeout |
| SIxx3 | 8009 | E | H | Read from child process failed |

Additional Descriptions

This list provides additional descriptions for some EFT 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 EFT to increase the GLOBAL environment.

UA-501 Protocol error - expected [SS] - got [SS]

Severity: Error

Explanation: The protocol type in EFT 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 EFT 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 functions 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, EFT server startup files may not contain any of the following commands: CONNECT, DISCONNECT, LOCAL, RECEIVE, REMOTE, or SEND. These commands also may not be embedded within EFT aliases.

UA-4133 ProdConfRead Error

Severity: Error

Explanation: This message is displayed when there are errors in reading the PRODCONF file. The specific message text will vary depending on the cause of the problem. The messages that could be reported are:

- “Failed to open product config file ‘SSSSSS’” where SSSSSS is the name of the prodconf file the license verification program attempted to access.
- “No data in file ‘SSSSSS’” indicates that the prodconf file, indicated by SSSSSS, is empty.
- “Missing LICPATH value in file ‘SSSSSS’” indicates that the prodconf file does not contain a value for the LICPATH parameter.

UA-4134 EFTUTL1: CPU Serial Number could not be obtained.

Severity: Error

Explanation: The license verification routine was unable to read the CPU serial number.

UA-4135 License Verify Error

Severity: Error

Explanation: This message is displayed when there are errors in reading the license keys file or the contents of that file. The specific message text will vary depending on the cause of the problem. The messages that could be reported are:

- “Failed to open NESikeys file ‘SSSSSS’” indicating that keys file SSSSSS does not exist or cannot be accessed for other reasons.
- “Product EFT213 not licensed to run on this platform” indicates that a valid key for the referenced product does not exist.
- License keys may be invalid due to:
 - License expiration
 - Incorrect LPAR specified when requesting the License Key
 - Incorrect S/N specified when requesting the License Key
 - No Key in the keys file
 - Old or corrupt key

UA-4136 License Expiring Warning

Severity: Warning

Explanation: This message is displayed when the current license key is close to expiring. The expiration date can also be found by performing a ‘Show local/remote’. A new license key should be obtained and installed prior to the license expiration to prevent a disruption in service.

UA-4137 LicenseVerifyCapability Error

Severity: Error

Explanation: This message is displayed when there is an incompatibility between the EFT features configured and the features enabled by the software key. The actual text of the message will vary depending on the reason for the incompatibility. Possible messages are:

- “Product License protocol CAPability is incorrect.” This message indicates that EFT is attempting to run over a protocol (NetEx versus TCP/IP) that it is not licensed to use.

UA-4138 LicenseVerifyCapability Warning

Severity: Warning

Explanation: *License Capabilities Verify warning (reserved for future use).*

UA-4139 License Expired and Product will soon be non-operational

Severity: Warning

Explanation: EFT License keys contain expiration dates. This license key contains a grace period, which is allowing you to run past your term date. When the grace period is reached, the product will become not operational.

UA-4140 License Expired and Product is not operational

Severity: Error

Explanation: EFT License keys contain expiration dates. When the expiration date is reached, the product will become not operational.

UA-4501 Nested (or recursive) input/alias limit of NNN exceeded

Severity: Error

Explanation: EFT 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 EFT 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: EFT 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 EFT 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 EFT 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: This warning results from the TRANSLATE command to define character translations. Characters that cannot be redefined are uppercase alpha (“A”...”Z”), digits (“0”...”9”), space (“ ”), equal (“=”), and null (“”).

EFTxx3-2002 Data checksum (CRC) error at block NNN

Severity: Error

Explanation: When the CRC qualifier is enabled for a SEND or RECEIVE operation, a 32-bit CRC is calculated by the sender and verified by the receiver. The verification has failed due to some network interruption. Retry the transfer.

EFTxx3-2004 Sequence number error at block NNN

Severity: Error

Explanation: When the CRC qualifier is enabled for SEND or RECEIVE, a block sequence number is assigned by the sender and verified by the receiver. The verification has failed usually indicating lost data. Retry the transfer.

EFTxx3-2101 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.

EFTxx3-8002 File size limit exceeded

Severity: Error

Explanation: This error indicates that there is not enough room for the file to fit on the current UNIX partition. The file size limit is the amount of free space on the destination partition.

EFTxx3-8319 Bad CHARACTER data detected

Severity: Error

Explanation: This error indicates the transfer is using CHARACTER MODE and the data contains imbedded null bytes. Null bytes typically indicate non-CHARACTER data (binary). This file cannot be sent in CHARACTER MODE. Use a different mode of transfer (e.g., STREAM).

EFTxx3-8329 Invalid RECORD file format

Severity: Error

Explanation: The format of the record file is not native to UNIX EFT. A file to be read by EFT in RECORD MODE must have been previously written by EFT in RECORD MODE also.

EFTxx3-8336 Unknown archive mode tag field ‘SSS’

Severity: Error

Explanation: When restoring an archive file using mode RESTORE or COPY, the indicated prefix field is not supported. This error indicates an incompatible host type (i.e., not a peer) or an incompatible EFT version.

EFTxx3-8337 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.

EFTxx3-8351 Record length NNN exceeds buffer length NNN

Severity: Error

Explanation: This message indicates the transfer is being performed on a RECORD oriented file and the length of the RECORD exceeds the internal buffer length allocated. To remedy this problem increase the MAXRECORD value to allocate a larger internal buffer.

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.

SIxx3-8008 Login exceeded NNN second timeout.

Severity: Error

Explanation: The cause of this error is that either LOGIN (CONNECT) failed to successfully login and activate the EFT Responder, or the system is extremely busy (cannot get logged in during allotted time interval). To remedy the error either try again, or if it is due to a busy system, have the remote site administrator increase the LOGTIMEOUT value in the Service Initiator startup file and stop and restart the Service Initiator.

Index

| | |
|-----------------------------------|---------|
| <i>t</i> | |
| - | 49 |
| ! | |
| ! | 49 |
| # | |
| # | 49 |
| * | |
| * | 46, 49 |
| ? | |
| ? | 46 |
| { | |
| { | 50 |
| } | |
| } | 50 |
| A | |
| abbreviating alias names | 87 |
| accepting input | 79 |
| ADD function | 57 |
| alias | 18 |
| alias definition | 84 |
| alias input | 87 |
| alias name abbreviation | 87 |
| alias names | 87 |
| alias parameters | 86 |
| alias types | 85 |
| arithmetic operations | 57 |
| ASK command | 102 |
| asterisk | 49 |
| B | |
| BACKUP mode | 47 |
| -BLOCK | 11 |
| Boolean qualifiers | 15, 16 |
| braces | 50 |
| breaking a connection | 24 |
| C | |
| changing prompts | 17 |
| CHARACTER mode | 47 |
| checking command status | 83 |
| CHR function | 58 |
| CMP function | 59 |
| command format | |
| ASK | 102 |
| CONNECT | 104 |
| CONTINUE | 108 |
| DISCONNECT | 109 |
| EXIT | 110 |
| GOTO | 112 |
| HELP | 113 |
| INPUT | 115 |
| LOCAL | 118 |
| ON | 120 |
| OUTPUT | 123 |
| QUIT | 125 |
| RECEIVE | 126 |
| REMOTE | 127 |
| SEND | 129 |
| SET | 130 |
| SET ALIAS | 132 |
| SET GLOBAL | 134 |
| SET HOST | 136 |
| SET VARIABLE | 138 |
| SHOW | 140 |
| SHOW ALIAS | 141 |
| SHOW GLOBAL | 142 |
| SHOW HOST | 143 |
| SHOW QUALIFIER | 144 |
| SHOW VARIABLE | 145 |
| TEXT | 146 |
| TRANSLATE | 147 |
| command interrupt | 32 |
| command qualifiers | 13 |
| command status | 83 |
| commands | 13, 101 |
| commands on local host | 29 |
| commands on remote hosts | 27 |
| COMPRESS | 41 |
| CONNECT command | 104 |
| connecting to a remote host | 19 |
| connection limitation | 22 |
| CONTINUE command | 108 |
| continuing alias definition | 85 |
| COPY mode | 47 |
| CRC | 41 |
| CREATE | 41 |
| creating aliases | 84 |
| D | |
| dash | 49 |
| DATE function | 60 |
| debugging | 88 |

| | |
|---------------------------------------|--------|
| alias | 88 |
| input script..... | 88 |
| DEC function..... | 61 |
| default values for file transfer..... | 41 |
| defining aliases | 18, 84 |
| description of commands..... | 101 |
| description of NetEx/eFT | 1 |
| destination wildcard support..... | 46 |
| developing scripts | 77 |
| DFN function..... | 62 |
| DIRECTORY definition..... | 43 |
| disabling string substitution..... | 76 |
| DISCONNECT command..... | 109 |
| disconnecting from a host..... | 24 |
| displaying aliases..... | 18 |
| displaying output | 79 |
| displaying qualifier values..... | 14 |
| displaying qualifiers | 13 |
| DIV function..... | 57 |
| double quotes..... | 50 |
| E | |
| echoing input | 78 |
| editing files | 32 |
| ENCRYPT function..... | 63 |
| ENV function..... | 64 |
| EQ function | 69 |
| EQS function | 65 |
| error message formatting | 89 |
| error messages | 18 |
| exclamation point | 49 |
| executing local commands..... | 29 |
| executing remote commands | 27 |
| EXIT command | 110 |
| exiting NetEx/eFT | 19 |
| EXPAND..... | 42 |
| EXT function | 66 |
| extended alias definition..... | 85 |
| F | |
| file handling..... | 41 |
| file specifications..... | 44 |
| file transfer..... | 24 |
| file transfer defaults | 41 |
| file transfer examples..... | 45 |
| file transfer qualifiers..... | 26, 41 |
| FLOW | 42 |
| functions | |
| ADD..... | 57 |
| CHR | 58 |
| CMP | 59 |
| DATE..... | 60 |
| DEC..... | 61 |
| DFN..... | 62 |
| DIV..... | 57 |
| ENCRYPT | 63 |

| | |
|--------------|----|
| ENV | 64 |
| EQ..... | 69 |
| EQS..... | 65 |
| EXT | 66 |
| GE..... | 69 |
| GT..... | 69 |
| INC | 61 |
| INDEX..... | 67 |
| LE | 69 |
| LEN | 68 |
| LOWER..... | 70 |
| LT | 69 |
| MOD..... | 57 |
| MSG | 71 |
| MUL | 57 |
| NDF..... | 62 |
| NE..... | 69 |
| NES..... | 65 |
| PARAMS..... | 72 |
| SLEEP | 73 |
| STATUS | 74 |
| SUB | 57 |
| TIME | 75 |
| UPPER..... | 70 |

G

| | |
|------------------------|-----|
| GE function..... | 69 |
| -Global..... | 11 |
| GLOBAL keyword | 11 |
| GOTO command..... | 112 |
| GOTO instruction | 81 |
| GT function..... | 69 |

H

| | |
|---------------------------------|-----|
| HELP command..... | 113 |
| help files..... | 95 |
| user defined | 95 |
| -HOMEdir | 11 |
| HOMEDIR keyword..... | 11 |
| host information exchange | 21 |
| host-independent commands..... | 31 |
| DIRECTORY | 43 |

I

| | |
|--|-----|
| INC function | 61 |
| INDEX function..... | 67 |
| information exchange..... | 21 |
| initiator | 1 |
| input and output | 16 |
| INPUT command | 115 |
| input files | 77 |
| input scripts | 77 |
| interrupting a command | 32 |
| interrupting string substitution | 76 |
| invoking NetEx/eFT..... | 11 |
| ISO model | 3 |

K

| | |
|-------------------------|----|
| keyboard interrupt..... | 32 |
| keywords | |
| -BLOCK..... | 11 |
| -GLOBAL..... | 11 |
| -HOMEDIR..... | 11 |
| -OUTPUT..... | 11 |
| -ROOTDIR..... | 12 |
| -SEARCH..... | 12 |
| -SERVICE..... | 12 |

L

| | |
|-------------------------------|-----|
| labels..... | 81 |
| LE function..... | 69 |
| left brace..... | 50 |
| LEN function..... | 68 |
| limit to number of hosts..... | 22 |
| LOCAL command..... | 118 |
| local commands..... | 29 |
| local startup files..... | 12 |
| local user guide..... | 11 |
| logical operations..... | 69 |
| LOGIN command..... | 21 |
| LOWER function..... | 70 |
| LT function..... | 69 |

M

| | |
|----------------------------|----|
| MAXRECORD..... | 42 |
| message formatting..... | 89 |
| messages..... | 18 |
| METHOD..... | 42 |
| MOD function..... | 57 |
| MODE..... | 42 |
| modes..... | 47 |
| BACKUP..... | 47 |
| CHARACTER..... | 47 |
| COPY..... | 47 |
| RECORD..... | 47 |
| RESTORE..... | 47 |
| STREAM..... | 47 |
| MSG function..... | 71 |
| MUL function..... | 57 |
| multicommand alias..... | 85 |
| multiple connections..... | 22 |
| multiword alias names..... | 88 |

N

| | |
|---------------------------------|---------|
| NDF function..... | 62 |
| NE function..... | 69 |
| NES function..... | 65 |
| nested string substitution..... | 76 |
| NetEx/eFT advantages..... | 1 |
| NetEx/eFT description..... | 1 |
| NetEx/eFT HELP command..... | 16, 113 |
| NetEx/eFT input and output..... | 16 |

| | |
|---------------------------|----|
| NetEx/eFT invocation..... | 11 |
| NetEx/eFT overview..... | 1 |
| NetEx/eFT prompt..... | 4 |
| NONE..... | 93 |
| notice to the reader..... | iv |

O

| | |
|----------------------------------|-----|
| ON command..... | 120 |
| ON ERROR instruction..... | 82 |
| ON INTERRUPT instruction..... | 82 |
| online help..... | 16 |
| -OUTput..... | 11 |
| output and input..... | 16 |
| OUTPUT command..... | 123 |
| OUTPUT keyword..... | 11 |
| overriding qualifier values..... | 15 |

P

| | |
|----------------------------|--------|
| PARAMS function..... | 72 |
| PARTIALRECORD..... | 42 |
| passing parameters..... | 79, 86 |
| positional parameters..... | 80 |
| pound..... | 49 |
| prompts..... | 4, 17 |

Q

| | |
|-------------------------|--------|
| qualifier | |
| Boolean..... | 15 |
| command line..... | 15 |
| qualifier value..... | 14, 15 |
| QUIET..... | 43 |
| QUIT command..... | 125 |
| quitting NetEx/eFT..... | 19 |

R

| | |
|--------------------------------|---------|
| RECEIVE command..... | 25, 126 |
| RECEIVE qualifiers..... | 26 |
| RECORD mode..... | 47 |
| REMOTE command..... | 127 |
| remote editing operations..... | 32 |
| remote host..... | 19 |
| remote host commands..... | 27 |
| remote user's guide..... | 35 |
| responder..... | 1 |
| RESTORE mode..... | 47 |
| right brace..... | 50 |
| RNT..... | 43 |
| RNT_BUFALLOC..... | 43 |
| RNT_INTERVAL..... | 43 |
| RNT_TIMEOUT..... | 43 |
| -ROOTdir..... | 12 |
| ROOTDIR keyword..... | 12 |

S

| | |
|---------------------|----|
| sample session..... | 4 |
| script..... | 88 |

| | | | |
|-------------------------------------|-----|---------------------------------------|-----|
| script input | 79 | STATUS function | 74 |
| script output | 79 | STREAM mode | 47 |
| scripts | 77 | string functions | 54 |
| scripts with string functions | 80 | string functions within scripts | 80 |
| -SEARCH | 12 | string substitution | 76 |
| SEARCH keyword | 12 | disabling | 76 |
| SEARCH keywords | 93 | nested | 76 |
| (NONE) | 93 | SUB function | 57 |
| (SITE) | 93 | T | |
| (USER) | 93 | temporary qualifier changes | 15 |
| SEND command | 129 | TEXT command | 146 |
| SEND qualifiers | 26 | TIME function | 75 |
| -SERVICE | 12 | transfer modes | 47 |
| SERVICE keyword | 12 | BACKUP | 47 |
| SET ALIAS command | 132 | CHARACTER | 47 |
| SET command | 130 | COPY | 47 |
| SET GLOBAL command | 134 | RECORD | 47 |
| SET HOST command | 136 | RESTORE | 47 |
| SET VARIABLE command | 138 | STREAM | 47 |
| setting qualifiers | 15 | transferring files | 24 |
| SHOW ALIAS command | 141 | TRANSLATE command | 147 |
| SHOW command | 140 | types of aliases | 85 |
| SHOW GLOBAL command | 142 | U | |
| SHOW HOST command | 143 | UNIX file handling | 41 |
| SHOW INPUT command | 16 | UPPER function | 70 |
| SHOW LOCAL command | 21 | USER | 93 |
| SHOW OUTPUT command | 17 | user defined help files | 95 |
| SHOW QUALIFIER command | 144 | W | |
| SHOW REMOTE command | 22 | wildcard characters | 46 |
| SHOW VARIABLE command | 145 | wildcard support | |
| SITE | 93 | destination | 46 |
| SLEEP function | 73 | source | 46 |
| SPACE | 43 | | |
| starting NetEx/eFT | 11 | | |
| startup files | | | |
| local | 12 | | |
| remote | 12 | | |

