# Bull DPX/20

## Unified File Transfer (UFT)
## Reference Manual

AIX

ORDER REFERENCE
**86 A2 10CB 03**

# Bull DPX/20

## Unified File Transfer (UFT) Reference Manual

AIX

---

**Software**

**March 1996**

**Bull Electronics Angers S.A.**
**CEDOC**
Atelier de Reprographie
331 Avenue Patton
49004 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
**86 A2 10CB 03**

The following copyright notice protects this book under the Copyright laws of the United States and other countries which prohibit such actions as, but not limited to, copying, distributing, modifying, and making derivative works.

Copyright © Bull S.A. 1992, 1996

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

## Trademarks and Acknowledgements

We acknowledge the right of proprietors of trademarks mentioned in this book.

AIX® is a registered trademark of International Business Machines Corporation, and is being used under licence.

UNIX is a registered trademark in the USA and other countries licensed exclusively through X/Open.

*The information in this document is subject to change without notice. Groupe Bull will not be liable for errors contained herein, or for incidental or consequential damages in connection with the use of this material.*

# About This Book

This manual describes how to install UFT (Unified File Transfer) on DPX/20 systems. It describes in detail the inter-operation of the DPX/20 with other similar systems and with non-DPX systems. The non-DPX systems covered in this manual are the Bull DPS 6, DPS 7, DPS 8, and IBM systems.

## Audience

This manual is intended for UFT users in a networking environment comprising DPX/20 systems.

# Operating System Level

This document is at Revision 2 level, which applies to AIX Version 4.1.4 or higher.

Migration of UFT for an earlier version of AIX to a new version is detailed in the OSI Communications Porting Guide.

# Supported Environment

Throughout this manual, except where non-DPX systems are specifically mentioned, all information pertains to the DPX/20 family and other DPX systems.

# Document Overview

This manual is structured in five chapters, seven appendices, and a general index:

**Chapter 1** **UFT Introduction**
is an introduction to the functions provided, and the files processed and transferred by UFT.

**Chapter 2** **UFT User Commands**
lists the UFT commands and explains how they are used when handling file transfers and managing the UFT operational interface.

**Chapter 3** **Access to Non DPX Systems**
deals with setting up UFT between the DPX/20 and non-DPX systems, and gives in detail the characteristics of each system type and the files that can be supported.

**Chapter 4** **Installation and Configuration**
describes the installation of UFT on the DPX/20 for the different network configurations using Ethernet and X25.

**Chapter 5** **UFT API Programmatic Interface**
lists the UFT functions and explains how they are used when handling file transfers and managing the UFT operational interface.

**Appendix A** **Error Messages**
gives the repertory of error messages during the connection and running phases of UFT.

**Appendix B** **File Identifiers on Non DPX Systems**
describes the syntax of file identifiers on non-DPX systems.

**Appendix C** **File Management on DPS 7**
is a summary of file management commands on the DPS 7.

**Appendix D** **Creating Files on DPS 8**
gives a brief summary of the recommended procedure for creating DPS 8 files.

**Appendix E** **System Limitations**
explains system limitations to be taken into account when using UFT.

Reading this Appendix is an absolute prerequisite before transferring NFS files.

**Appendix F UFT API Programming Example**

**Appendix G Remote Hosts Table**

**Glossary**

**Index**

## Conventions Used in This Document

The following typographic conventions are used in this document:

| | |
|---|---|
| **Bold** | Bold characters are used to highlight key words, commands and subroutines. |
| *Italic* | Italic characters represent file names and user supplied values. |
| `Courier` | Courier characters are used in examples and for user commands entered on the terminal keyboard. |
| Fixed pitch | Fixed pitch characters are used to present system display information. |
| Display fields | Individual fields on a display screen are outlined and presented sequentially. |
| # | Numeric field. |
| X | Hexadecimal field. |
| ***Mandatory** | The names of fields a user must complete are presented in bold type and with an asterisk (*) to the left. |

## References to Standards

UFT is in conformance with ISO/DSA standards.

## Bibliographical References


1. OSI Services Reference Manual – 86 A2 05AQ
2. Bull DPX/20 UFT Package Software Release Bulletin – 86 A2 48CM
3. UFT Diagnostic Guide – 86 A2 54AJ
4. Bull DPX/20 AIX Commands Reference Manual – 86 A2 73AP to 86 A2 78AP
5. Bull DPX/2 UFT Reference Manual – 86 A2 61SS
6. Bull OSI/DSA6/SNA6 Comprehensive UFT Facility – 69 A2 CZ92
7. Bull DPS 7 UFT User's Guide – 47 A2 13UC
8. Bull UFT8 & FTAM8 User's Guide – 67 A2 EB60
9. Bull DPX/20 NetShare User's Guide – 86 A2 95AP
10. OSI Communications Porting Guide – 86 A2 44AP


## Terminology

Whenever the term DPX/20 is used, it also includes the DPX/20 ESCALA and the DPX/20 ESTRELLA ranges.

The term "Operating System" is used to indicate the proprietary operating system software, in this case AIX.

# Table of Contents

# Chapter 1. UFT Introduction

## Overview

UFT (Unified File Transfer) is an application for transferring files between a DPX/20 and:

- either another DPX/20 or a DPX/2,
- or a non DPX system such as a DPS 6, DPS 7, DPS 8 or IBM system.

The two systems can be connected through:

- either an X25 Wide-Area Network (public or private WAN)
- or an Ethernet (normal or integrated LAN), an FDDI or a Token Ring Local Area Network.

The following sections describe:

- the Structure of UFT, on page 1-2,
- UFT Functions, on page 1-2,
- the Running Environment of UFT, on page 1-3,
- the Files processed by UFT, on page 1-4.

# Structure of UFT

The UFT product includes a configurator, several commands and subcommands, requestor and server programs, and APIs.

The UFT commands are initiated by:

- either a user,
- or an application.

and are executed:

- either interactively,
- or from an input command file.

The UFT service itself is composed of two processes:

- a requestor (client) process:
  The requestor process executes UFT commands locally initiated. For file transfer commands, this process sends and/or receives files.
- a server process:
  The server process is called by a remote requestor process to execute UFT commands remotely initiated. For file transfer commands, the server process sends and/or receives files as required by the remote requestor process.

The UFT APIs supply all the functions of the requestor process of UFT.

# UFT Functions

UFT offers the following functions:

- connecting to a remote host,
- creating of a remote file,
- deleting of a remote file,
- requesting information about a remote file (attributes),
- sending files from the local system to the remote host,
- receiving files from the remote host to the local system,
- aborting/interrupting current file transfer,
- restarting an interrupted file transfer,
- disconnecting from the remote host.

Certain capabilities are negotiated at the time of connection to the remote host:

- creating a remote file,
- deleting a remote file,
- requesting information about a remote file (attributes),
- restarting an interrupted file transfer.

For UFT connections between systems of the DPX/20 or DPX/2 range, these negotiable capabilities are always available.

# Running Environment of UFT

The environment in which UFT can run must provide:

- the OSI communications stack,
- and one of the supported communications adaptors. See *OSI Services Reference Manual*.

The UFT environment is shown in the following Figure.

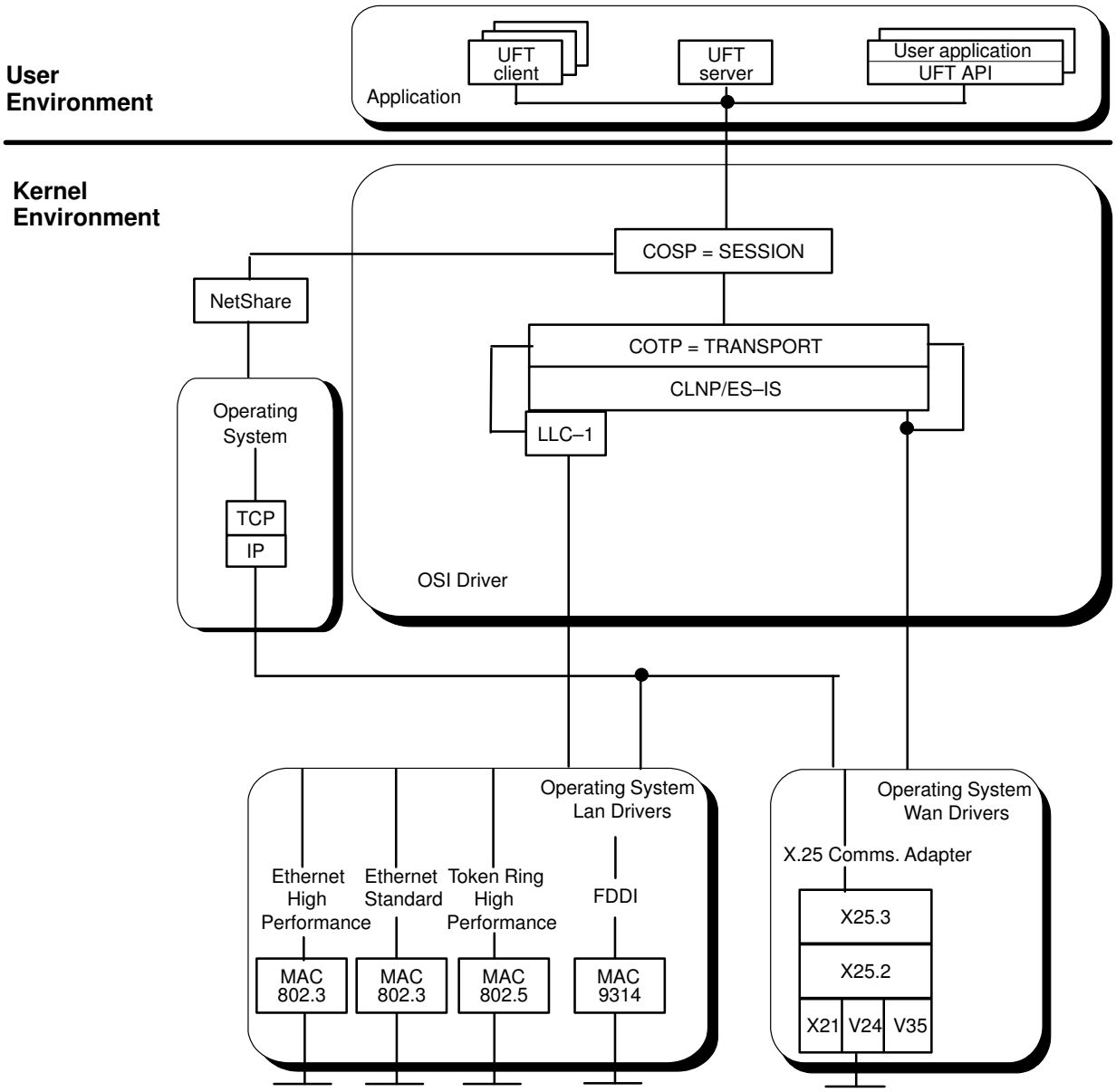



Figure 1. UFT Functional View.

# Files Processed by UFT

## Types of Files Processed by UFT

Files locally processed by UFT are of two types, namely:

- *byte stream* files such as *binary files* which consist of a stream of unstructured bytes,
- *sequential* files such as *text files* which are organized as variable length records where each line corresponds to a file record.

## File Transfer Modes

There are two file transfer modes:

- *block* mode (default) in which data is transferred as records of fixed size in binary code
- *line* mode in which data is transferred as records of variable size in ASCII code.

The size of the transfer records is restricted to a maximum value set by the user at the time of connection with the remote host (*max record size* parameter of the *connect* command).

The value set by the user cannot exceed 16384 bytes. The default size is 512 bytes.

The type of file to be transferred determines the transfer mode:

- *binary files* are always transferred in *block* mode even if *line* mode is specified. The reason for this is that UFT scans the first 512 bytes in the files to be sent and if more than 50% are non ASCII, transmission automatically takes place in block mode. A warning message is printed.
- for *text files,* the transfer mode depends on the systems involved in the transfer:

  **block mode** is recommended between DPX systems for better performance.

  **line mode** is used where the remote host is a non DPX system, for example:

  – DPS 7:
  DPS 7 data is encoded in EBCDIC. **Line mode** ensures that the DPS 7 is notified of the transcoding either when sending data (EBCDIC –> ASCII) or when receiving data (ASCII –> EBCDIC).

  – DPS 8:
  the DPS 8 file structure differs from that of the DPX/20. **Line mode** ensures that the file is broken up into lines according to the text file sent to resolve this incompatibility.

## File Name Structure

The structure of file names specified by UFT must be the same as the file name structure prevailing on each system. For non DPX systems such as the DPS 7 and DPS 8, see Appendix B File Identifiers on Non DPX Systems. For DPX systems, the UFT service imposes the following structure:

```
[label(dirname(]filename
```

where:

*label* the label of the disk or partition (virtual disk) on which the file resides but which has not been mounted.

*dirname* the name of the directory in which to mount the disk or partition.

**Note:** *dirname* is enclosed in two opening parentheses.

*filename* the name of the file on the DPX system.

If the file is on a disk or partition already mounted, only *filename* must be specified.

If the file is on a disk or partition to be mounted, both *label* and *dirname* must be specified. The disk or partition must therefore have a label. The UFT service displays a prompt on the system console requesting the operator to mount the disk or partition.

The user selects the physical unit on which to mount the disk or partition. The UFT service refuses the service requested after displaying the prompt ten times at intervals of around 40 seconds.

Examples:

1. The structure of the file name is `/usr/proc/file1` on a cartridge disk labelled `back` that must be mounted in the directory `/mnt304`:

   `back(/mnt304(/usr/proc/file1`

2. The structure of the file name is as follows for a file `/usr/proc/file1` on a fixed disk or a mounted cartridge disk or a mounted partition:

   `/usr/proc/file1`

A file located on the *requestor* system is identified either by its absolute *filename* or the *filename* associated with the current directory under UFT.

A file located on the *server* system is identified by its absolute *filename* for non DPX systems. If the *server* is a DPX system, the file is identified by its *filename* associated with the directory at remote login.

**Note:** On DPX systems, intermediate directories and files are created automatically when transferring or creating files.The current directory on the remote host cannot be changed.

# Chapter 2. UFT User Subcommands

## Overview

This section lists and describes the user subcommands that can be executed under UFT.

- Subcommands are first described briefly in alphabetical order, in “List of UFT Subcommands”, on page 2-2,
- Subcommands are then covered in detail in the **uft** command, on page 2-3,
- Commands are briefly described in “User Tasks”, on page 2-15, in particular how to connect to a remote host, transfer a file, request information on the connection, exit UFT, etc.
- Use of command files is described in “Automation of UFT Connection and Transfers”, on page 2-17.

# List of UFT User Subcommands

The following commands available under UFT are given in alphabetical order.

**cd** Changes the current directory of the local system under UFT.

**cfile** Executes a sequence of UFT commands.

**cmp** [**on** | **off**] Compresses data prior to transfer.

**cmp** Displays the current status of the compress mode.

**connect** Sets up a UFT service connection (alias **open**).

**create** Creates the *remote_file* on the remote host.

**CTRL C** Aborts the current file transfer.

**delete** Deletes the *remote_file* on the remote host.

**disc** Closes the UFT current connection to a remote host.

**exit** Exits from UFT (alias **quit**).

**help** Displays the list of available commands.

**help** *command*
Displays the syntax for the command specified.

**modify** Modifies the connection characteristics set at the time of the last **connect**.

**noabort** [**on** | **off**]
Enables or disables the full restart option.

**noabort** Displays the status of the full restart option.

**rcdir** Receives from the *remote_directory* only the files whose names already exist in the *local_directory* (alias **rd**).

**readatt** Gets information about the *remote_file* on the remote host.

**receive** Transfers a file from the remote host to the local system.

**restart** Restarts an interrupted transfer.

**send** Transfers a file from the local system to the remote host.

**sndir** Transfers all the files from the *local_directory* to the *remote_directory*.

**shell** Starts an interactive SHELL.

**trace** Sets or resets the trace facility.

**uft** Starts UFT.

**uft –n** Starts UFT running commands previously stored in *input_file*.

**uft –t** Starts UFT with the specified trace level.

**verbose** [**on** | **off**]
Displays dynamically the number of transferred kilobytes.

**verbose** Displays the current status of the verbose mode.

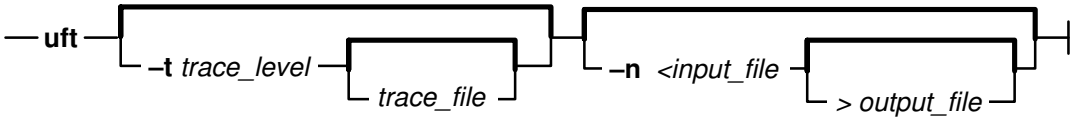
**who** Displays the characteristics of the current connection from the local point of view.

# uft Command

## Purpose

Transfers files between a local and a remote host.

## Syntax




## Description

The **uft** command is need to transfer files between a DPX/20 and:

- either another DPX/20 or a DPX/2,
- or a non–DPX system such as a DPS 6, DPS 7, DPS 8 or IBM system.

### Issuing Subcommands

At the `uft>` prompt, you can enter subcommands to perform tasks such as changing the current local directory, transferring multiple files in a single request, creating/deleting a remote file and escaping to the local shell to perform shell commands. See the "Subcommands" section on page 2-4 for a description of each subcommand.

## Flags

**–t** Sets the trace facility at the *trace_level* (0...3). If *trace_file* is omitted the trace is in the *stderr* file.

**–n** Starts UFT with a command file.

*input_file* contains the UFT commands and the replies to the various questions asked by UFT.

*output_file*:

- if specified, stores the results of executing the command file
- if omitted, displays the progress of execution on the console.

Example of an input file called *file1* which contains the following:

```
connect SYSTEM001
logloc
passloc
logdist
passdist
blank_line
blank_line
blank_line
blank_line
blank_line
blank_line
who
quit
```

The command **uft –n** <*file1* executes as follows:

- UFT reads the first line of *file1*:

  ```
  connect SYSTEM001
  ```

  This command asks a number of questions.

- For each question asked, UFT reads the next line of *file1* and takes it as the reply:

  ```
  question:       -> reply:

  local login              -> logloc
  local password           -> passloc
  remote login             -> logdist
  remote password          -> passdist
  remote billing           -> blank_line
  remote project           -> blank_line
  max record size          -> blank_line
  transfer mode            -> blank_line
  remote machine           -> blank_line
  ```

  *blank_line* means <enter> or <return> or <CR>. If a reply is mandatory, UFT then takes the default value specified in parentheses.

  The question `DPS8 file type` appears after `remote machine` if the answer to this question is DPS8 (value 2 instead of blank line as in the above example).

  `connect SYSTEM001` is executed when all questions are answered.

  UFT then reads the next lines, **who** and **quit**, and executes them one after the other.

**Note:** If the **uft** command is put in a `proc` file, run UFT as follows:

```
nohup uft -n <proc &
```

This avoids aborting a running transfer on keying in `CTRL C`.

For further information on using UFT with a command file, see “Automation of UFT Connection and Transfers” on page 2-17.

## Subcommands

The following **uft** subcommands can be entered at the `uft>` prompt. Use double quotes (” ”) to specify parameters that include blank characters.

**!***shell_command* [*Parameters*][;*shell_command* [*Parameters*]...]
: Invokes an interactive shell on the local host. An optional command or the list of shell commands, with one or more optional parameters, can be given with the shell command. See the **shell** subcommand.

**?** [*subcommand*]
: Displays a help message describing the command. If you do not specify a *subcommand* parameter, the **uft** command displays a list of available subcommands.

**cd** *local_directory*
: Changes the current directory of the local system.

On starting UFT, the user is in the current directory. When connecting to a remote host, the user is in the directory of the local login chosen during connection.

The *local_directory* can be changed using the command **cd** at any time.

If *local_directory* is omitted, the user is:

- either in the directory of the local login used to set up the current UFT connection,
- or in the directory of the user login if there is no current UFT connection.

**cfile** *local_file* [**at**]

Executes a sequence of UFT commands concerning transfers, creation, file deletion and attribute acquisition, once connection is established. All commands are allowed in the *local_file* except **cfile**, **connect**, **disc**, **exit** and **shell**.

In the *local_file*, a comment line has to start with a # character.

**at** enables deferred execution of the command file (only if the syntactic and semantic analysis is correct). However, the user needs to be authorized by the system administrator to use the **at** command.

If **at** is omitted, the commands in the *local_file* are executed immediately.

The syntax of the **at** command is the same as for the Unix **at** command.

When **at** is specified, UFT requests:

- the date and time at which execution should start (the **at** command to enter the date and time can only be used with the authorization of the System Administrator)
- the name of the file in which the result of the deferred execution should be recorded.

Two files are then created:

- one to run the connection and transfer automatically at the requested time when the user directly invokes the command /**usr**/**bin**/**transdiff**. See “Automation of UFT Connection and Transfers” on page 2-17.
- the other contains the connection parameters plus the UFT commands to be run at the time specified.

See example 1 (without **at**) and example 2 (with **at**), on page 2-17.

**close** Closes the UFT current connection to a remote host. See the **disc** subcommand.

**cmp** [**on** | **off**] Compresses data prior to transfer.

Typing **cmp** only gives the current status of the compress mode.

**on** activates the compress mode

When **cmp** is enabled (**on**), all data subsequently transferred is compressed before transfer, in accordance with DSA62. If the file to be transferred has a good compress rate, the throughput is improved.

**off** disables the compress mode.

**connect** *remote_host*

Opens a UFT service connection with the *remote_host*.

The *remote_host* must have been declared in the table */etc/isohosts*. See “Remote Host Access Table” on page 4-5.

To set up the connection between the local and remote hosts, UFT needs the following information:

- the *local login* known by the local system, which can be different from that set for the user before system login under UFT,
- the *local password* corresponding to the local login,
- the *remote login* known by the remote host,
- the *remote password* corresponding to the remote login,

  The **connect** command proposes a default login (the current login) so that the user does not have to enter his local and remote login nor his local password if they are the same. The remote password must be entered in all cases.

**Note:** Accept the default value by pressing the Enter key (or by leaving a blank line in a command file).

- the *billing* identifier of the remote login (up to 12 characters),
- the *project* identifier of the remote login (up to 12 characters),

**Note:** On DPX systems, *billing* and *project* have no significance.

- the *max record size* for the transfer records expressed in bytes for which:
  - either the user supplies a value up to a maximum of 16384 by entering a number code. Enter 0 if you want to give a specific size, 1 for 512, 2 for 950 and 3 for 2000.
  - or UFT uses the default value 512.

  Make sure, however, that the *max record size* is greater than or equal to the longest line when transferring text in line mode.
- the *transfer mode*. Enter 1 for block mode and 2 for line mode. The default is block mode.See “File Transfer Modes” on page 1-4.
- the *remote machine type* (DPX, DPS 6, DPS 7, DPS 8 or IBM):

  Enter 1 for DPX, 2 for DPS 8, 3 for DPS 7, 4 for DPS 6 and 5 for Other (foreign hosts such as IBM systems).

  DPX is the default setting.
- if appropriate, the *DPS 8 file type:*

  The remote file type must be specified if the remote machine is a DPS 8.

The connection request results in one of the following:

- connection refused (*connect failed* message);
- access granted to all services negotiated with the remote host and available for the connection.

The parameters defined for the connection are used as default values for a subsequent connection, unless these parameters have been set differently

by using the **modify** command, in which case the last parameter setting prevails.

UFT only manages one connection at a time. To converse with another system, first disconnect from the current system before entering another connection request. See “Disconnecting from Remote Host” on page 2-15.

Once the connection request is accepted, the user’s access rights are those of the selected local login.

**create** *remote_file* [–**s** *remote_filesize*]

Creation of a *remote_file* on the remote host is available if the facility is also offered by the remote system. If both systems are DPX systems, this facility is always available.

The file name syntax required by the remote host must be observed:

- On all machines except the DPS 8, the file is created with a sequential organization.
- On the DPS 8, the file organization is dependent on the remote file type:

  GFRC ASCII files: sequential (linked)

  UFF sequential files: sequential

  UFF relative files: relative

–**s** *remote_filesize*

Option **–s** is used to specify the *remote_filesize* in kilobytes (1 Kbyte = 1024 bytes) of the file to be created. The default size is 100 Kbytes.

This size is not significant in the case of files created on DPX systems. However, it affects the static allocation of disk space on remote hosts of a different type.

On the DPS 8, files are created with an unlimited size which proportionally increases without user intervention if the current size is not sufficient to receive a file.

**Return Values**

If an error occurs, a message is displayed. See “Error Messages” on page A-1.

**CTRL C** Interrupts the current file transfer and displays the number of kilobytes already transferred.

If the **restart** facility is offered by the remote system, the transfer is interrupted and can be restarted at a later time. If this facility is not available, the transfer aborts.

CTRL C can interrupt sending and receiving of individual files. For directory transfers, only the current file transfer is affected, not the transfer of the directory. The files that follow in the directory continue to be transferred in sequence.

**delete** *remote_file*

Deletion of the *remote_file* on the remote host is available if the facility is also offered by the remote system. If both systems are DPX systems, this facility is always available.

The file name structure on the remote host must be observed.

**Return Values**

If an error occurs, the reason for failure to delete is displayed. See "Error Messages" on page A-1.

**disc** Disconnects from the remote host.

**Return Values**

If an error occurs, a message is displayed. See "Error Messages" on page A-1.

**exit** Quits UFT and closes the connection to the remote host.

**ga** *remote_file* Gets information about the *remote_file* on the remote host. Same as the **readatt** subcommand.

**get** *remote_file* **ap** *local_file*
Transfers a file from the remote host to the local system. See the **receive** subcommand.

**help** [*command*]
Displays help information. See the **?** command.

**modify** Modifies certain connection characteristics set at the time of the last **connect**.

The characteristics that can be modified are:

- the maximum size of the transfer records,
- the file transfer mode,
- the remote machine type,
- the remote file type.

The format of the questions asked is practically the same as for the **connect** command.

For *remote machine type*, select the value `other` if the remote machine is a foreign host such as an IBM system running the UFTF application.

The question *remote file type* is asked only if the remote machine type selected is DPS 8. Possible answers are GFRC (GFRC ASCII files), UFF sequential and UFF relative files. See "Transferring NFS Files with UFT" on page E-1.

**noabort** [**on** | **off**]
Enables (**noabort on**) or disables (**noabort off**) the full restart option. When this function is activated, a transfer interrupted by the user or by a connection breakdown can be restarted using the **restart** subcommand.

**noabort** displays the status of the full restart option.

**open** *remote_host*
Sets up a UFT service connection to the *remote_host*. See the **connect** subcommand.

**put** *remote_file* [**ap** *local_file*|**as** *local_file*]
Transfers a file from the local system to the remote host. Same as the **send** subcommand.

**quit** Exits from UFT. Same as the **exit** subcommand.

**rcdir** *local_directory* [**from** *remote_directory*]

Receives the contents of all files from the *remote_directory* that already exist in the *local_directory*.

**Note:** This command refers to a file system specific to DPX systems and can be used only where both systems belong to this family.

A recursive search is performed for the file names in the local directory. The list of files to be received is therefore based on this search in the local directory.

The creation of intermediate directories and files is not an issue because the list of files to be received already exists in the local directory.

Keying in **CTRL C** interrupts only the transfer of the current file; the other files in the directory continue to be transferred in sequence.

If the **restart** facility has been negotiated, interrupted file transfers can be restarted at a later time.

**from** *remote_directory*
The option **from** specifies the name of the *remote_directory*. The file names in the directory remain unchanged. The default remote directory name is *local_directory*.

By default, the remote directory has the same name as the local directory.

**Return Values**

If an error occurs, a message is displayed. See “Error Messages” on page A-1.

**rd** *local_directory* [**from** *remote_directory*]
Receives from the *remote_directory* only the files whose names already exist in the *local_directory*. In the same way as the **rcdir** subcommand.

**readatt** *remote_file*
This command to request attributes of a remote file is available if the facility is also offered by the remote system. If both systems are DPX systems, this facility is always available.

If the request for attributes is accepted, the attributes of the *remote_file* are displayed:

- file name
- current file size in Kbytes
- size of the file records in bytes
- type of file organization
- maximum file size in Kbytes
- date of file creation
- date of last access to the file
- date of last modification to the file.

If both systems are DPX systems, the size of the records and the type of file organization have no significance. The date of creation is the actual date of creation if no modification has been made, or else it is the date of the last modification.

**Return Values**

If an error occurs, a message is displayed. See “Error Messages” on page A-1.

**receive** *remote_file* [**ap** *local_file*|**as** *local_file*]

Transfers a file from the remote host to the local system.

**receive** *remote_file* **ap** *local_file* transfers a file from the remote host to the local system. The file received is concatenated to the local file.

**receive** *remote_file* **as** *local_file* transfers a file from the remote host to the local system. By default, the name of the local file is *remote_file*.

The **as** option renames the transferred file from *remote_file* to *local_file*.

If **as** *local_file* and **ap** *local_file* are omitted:

- **as** is the default option,
- the *local_file* has the same name as the *remote_file*.

If there is already a local file with the same name as the file received, the file received overwrites the contents of the existing file.

If there is no file of the same name as the file received, the file is automatically created, as are all intermediate directories that exist in the file access path.

**ap** concatenates the *remote_file* received to the existing *local_file*.

The transfer can be interrupted by keying in **CTRL C**. If the **restart** facility has not been negotiated, the transfer aborts. Otherwise, it is only interrupted and can be restarted at a later time.

**Return Values**

If an error occurs, a message is displayed. See “Error Messages” on page A-1.

**restart** [**–d**] [*transfer_id*...| **last**]

Restarts an interrupted transfer with the specified *transfer_id*. The last interrupted transfer can be restarted by specifying **last**. If no arguments are provided, a list of the interrupted transfers is printed.

The full restart facility makes it possible to restart an interrupted transfer whether the interruption was caused by the user (**CTRL C**) or by a connection breakdown (UFT process killed, power failure etc.).

In the case of a connection breakdown, the noabort mode must be active (**noabort on**), at that time, to enable subsequent restart. In the case of a user interrupt, the transfer can be restarted even if the noabort mode is not enabled.

When a transfer is interrupted, context information about the transfer is stored by the UFT *Requestor* and *Server* in two system files. This information becomes invalid if the transfer is not restarted within 30 days; then restart is no longer possible.

For restart to be available, the characteristics of the remote site in the */etc/isohosts* file must include expedited data. (See Structure of Remote Hosts Table on page 4-5).

When this command is invoked without any arguments, a summary of all user interrupted transfers is displayed. This information is given in the following format:

```
line 1: transfer_id transfer_mode record_size
remote_site_and_user

line 2: original_transfer_command

line 3: number_of_blocks/records_transferred
```

A transfer can be restarted only if the user is connected as the remote user at the time of transfer interruption.

A restarted transfer can be further interrupted and resumed. To restart an interrupted transfer, there is no need to set the mode and record size to the values they had at the time of interruption.

Restart takes place from the beginning of the file when:

- the receiving file no longer exists,
- the sending or receiving file is modified,
- there are no read permissions on the receiving file and the transfer mode is line.

This facility is available if it is also offered by the remote system.

**Parameters**

**–d**
deactivates transfers marked for restart.

*transfer_id*
is the identifier given by UFT at the beginning of the transfer (between 9 and 11 decimal digits) the *local_directory* can be changed using the command **cd** at any time.

When specifying *transfer_id*, metacharacters * and ? can be used for specifying the number of occurrences:

- * can match up to 11 characters,
- ? can match exactly one character.

**last**
is a keyword to indicate the last interrupted transfer.

**Return Values**

Restart fails and return an error message if:

- the sending file no longer exists,
- the access rights on the transferred file are modified preventing reading (for the source file) or writing (for the target file),
- the remote host no longer knows about the transfer (log file or entry corresponding to the interrupted transfer has been deleted using SMIT).
- a write problem occurred in the log file where the identifiers are saved. This can happen when the UFT server or requestor process is killed when flushing information on disk.

The different error messages returned on a failed restart are described in “Error Messages” on page A-1.

**sd** *local_directory* [**in** *remote_directory*]
Transfers all the files from the *local_directory* to the *remote_directory*. Same as the **sndir** subcommand.

**send** *local_file* [**ap** *remote_file*|**as** *remote_file*]
Transfers a file from the local system to the remote host.

**send** *local_file* **ap** *remote_file* transfers a file from the local system to the remote host. The file sent is concatenated to the remote file.

**send** *local_file* **as** *remote_file* transfers a file from the local system to the remote host.

If **as** *remote_file* and **ap** *remote_file* are omitted:

- **as** is the default option,
- the *remote_file* has the same name as the *local_file*.

The option **as** renames the transferred file from *local_file* to *remote_file*.

If the name specified for the remote file already exists, the file sent overwrites the existing file.

If the name specified for the remote file does not already exist, then a file is either created or not, depending on the type of the remote host:

for DPX systems:
the file is created automatically, as are all the intermediate directories that do not exist in the file access path;

for non–DPX systems:
a file is not created systematically. See “Access to Non–DPX Systems” on page 3-1.

The **ap** option concatenates the *local_file* sent to the existing *remote_file*. However, appending to a file with relative organization is not allowed.

The transfer can be interrupted by keying in **CTRL C**. If the **restart** facility has not been negotiated, the transfer aborts. Otherwise, it is only interrupted and can be restarted at a later time.

**Return Values**

If an error occurs, a message is displayed. See “Error Messages” on page A-1.

**shell** (or **sh**)
Starts an interactive SHELL. Key in **CTRL D** to switch back to UFT in non–interactive mode.

**!***shell_command* [*Parameters*][;*shell_command* [*Parameters*]...]

(*shell_commands* specified after the **!** execute under shell, after which control always returns to UFT).

The value stored in the environment variable `SHELL` invokes the appropriate *shell*. If this variable is not defined, *Bourne shell* is the default.

Example:

If using *korn–shell*, put the following lines in the `.kshrc` file:

```
SHELL=/bin/ksh
export SHELL
```

UFT then uses *korn–shell* each time **shell** is invoked.

**sndir** *local_directory* [**in** *remote_directory*]

Transfers all the files from the *local_directory* to the *remote_directory*.

**Note:** This command refers to a file system specific to DPX systems and can be used only where both systems belong to this family.

It sends the contents of all files in the *local_directory* to the *remote_directory*.

A recursive search is performed for file names in the local directory.

Any files and intermediate directories that do not exist on the remote host are created in the same way as when sending an individual file. See “Sending a File to a Remote Host”, on page 2-15.

Keying in **CTRL C** interrupts only the transfer of the current file; the other files in the directory continue to be transferred in sequence.

If the **restart** facility has been negotiated, interrupted file transfers can be restarted at a later time.

**Return Values**

If an error occurs, a message is displayed. See “Error Messages” on page A-1.

**trace** [<trace level (0...3)>[**in** <trace file name>]]

Sets or resets the trace facility of the requestor process.

**Note:** For this facility to be available it needs to be present in the SMIT configuration. See “Starting UFT Using SMIT” on page 4-11.
In this SMIT menu, it is also possible to specify the trace file name of the server process.

If the trace is being set for the first time (trace level different from 0), then the trace file name must be specified. If the trace level is already set, it can be changed by specifying a new trace level.

If no argument is supplied, the current status of the trace facility is displayed.

**verbose** [**on** | **off**]

When enabled (**on**), this command displays dynamically the number of transmitted Kbytes during a send or receive operation.

If no argument is specified, the current status of the verbose mode is displayed.

Examples:

```
uft> verbose
verbose is off
uft> verbose on
verbose is on
uft> send /tmp/file1 as /tmp/file_on_dorade
```

```
Sending /tmp/file1 as /tmp/file_on_dorade
Transfer Identification : 171783280
Transferred Size : 280 Kbytes
```

The value of the Transferred Size field changes dynamically as the transfer progresses.

**who**

Displays the characteristics of the current connection from the local point of view.

This command provides information on the current connection:

- the remote host with which the connection is established,
- the local login,
- the remote login,
- the maximum size of the transfer records,
- the file transfer mode,
- the current directory of the local system,
- the remote machine type,
- the remote file type, if the remote machine is a DPS 8,
- noabort status (on/off),
- compress status (on/off),
- verbose status (on/off),
- trace status (on/off),
- trace level (0–3),
- trace file.

## Implementation Specifics

Software Product/Option: OSI Stack

Standards Compliance: DSA62, OSI

## File

/**etc**/**isohosts** Contains the remote host access descriptions.

## Related Information

*OSI Services Reference Manual*.

Network Overview *AIX System Management Guide: Communications and Networks*

# User Tasks

This section contains a brief description of each command that can be executed under UFT.

## Connecting to a Remote Host

**connect** *remote_host* (alias **open**)

This command opens a UFT service connection with the *remote_host*.

## Compressing Data for Transfer

**cmp** [**on** | **off**]

This command compresses data prior to transfer.

## Modifying Characteristics of Current Connection

**modify**

This command modifies certain connection characteristics set at the time of the last **connect** or **modify**.

## Consulting Characteristics of Current Connection

**who**

This command provides information on the current connection.

## Disconnecting from a Remote Host

**disc** (alias **close**)

This command disconnects from the remote host.

## Creating a File on a Remote Host

**create** *remote_file* [**–s** *remote_filesize*]

This command creates a *remote_file* on the remote host. It is available if the facility is also offered by the remote system. If both systems are DPX systems, this facility is always available.

## Deleting a File on a Remote Host

**delete** *remote_file*

This command deletes the *remote_file* on the remote host. It is available if the facility is also offered by the remote system. If both systems are DPX systems, this facility is always available.

## Requesting the Attributes of a File on a Remote Host

**readatt** *remote_file* (alias **ga**)

This command requests attributes of a remote file. It is available if the facility is also offered by the remote system. If both systems are DPX systems, this facility is always available.

## Sending a File to a Remote Host

**send** *local_file* [**ap** *remote_file*|**as** *remote_file*] (alias **put**)

This command transfers a file from the local system to the remote host.

## Receiving a File from a Remote Host

**receive** *remote_file* [**ap** *local_file*|**as** *local_file*] (alias **get**)

This command transfers a file from the remote host to the local system.

## Sending a Directory to a Remote Host

**sndir** *local_directory* [**in** *remote_directory*] (alias **sd**)

This command refers to a file system specific to DPX systems and can be used only where both systems belong to this family.

## Receiving a Directory from a Remote Host

**rcdir** *local_directory* [**from** *remote_directory*] (alias **rd**)

This command refers to a file system specific to DPX systems and can be used only where both systems belong to this family.

## Aborting/Interrupting Current File Transfer

**CTRL C**

This command interrupts the current file transfer and displays the number of kilobytes already transferred.

## Restarting Interrupted File Transfer

**restart** [**–d**] [*transfer_id*>...|**last**]

This command restarts an interrupted transfer.

**noabort** [**on** | **off**]

This command enables or disables the full restart option.

## Getting Help on UFT Commands

**help** [*command*] (alias **?**)

If *command* is omitted, the list of all available commands is displayed. Otherwise, the instructions for the *command* are displayed.

## Changing Local Directory under UFT

**cd** [*local_directory*]

This command changes the current local directory under UFT.

## Switching to Shell

Switching to the shell command interpreter can be done by keying in:

**sh** (or **shell**)

## Displaying Kilobytes transferred

**verbose** [**on** | **off**]

When enabled (on), this command displays dynamically the number of transmitted kbytes during a send or receive operation.

## Trace Facility

**trace** [<trace level (0...3)>[**in** <trace file name>]]

The trace command sets or resets the trace facility.

## Exiting from UFT

**exit** (alias **quit**)

Quits UFT and closes the connection to the remote host.

# Automation of UFT Connection and Transfers

This section describes the three methods for running automatic file transfers under UFT, either by using the **cfile** command, or by a direct call to the **transdiff** command, or by starting uft with the option **–n**.

Only the first two commands enable deferred execution.

## cfile Command

**cfile** *file* [**at**]

The **cfile** command runs, from *file*, a sequence of UFT commands concerning transfers, creation, file deletion and attribute acquisition, once connection is established.

**at** enables deferred execution of the command file. However, the user needs to be authorized by the system administrator to use the **at** command. An example of use is given below. The appropriate system documentation can also be consulted on the subject.

**Example 1 (without at):**
After connection under UFT, **cfile** is entered:

uft>**cfile** *cmdfile*

*cmdfile* contains, for example:

```
create /tmp/test
send /etc/isohosts as /tmp/test
receive /etc/hosts as /tmp/test1
```

`create`, `send` and `receive` execute one after another and the results are displayed on the console on which *cfile* was entered.

When the sequence is finished, the prompt `uft>` reappears.

**Example 2 (with at)**:
After connection under UFT, *cfile* is entered:

```
uft>cfile cmdfile at
date and time 11:00:00
result file /tmp/resul
```

The following information is required for the deferred execution of *cmdfile*:

- date and time for execution,
- results file for connection and transfers.

*cmdfile* will run at 11:00 hours the same day, and the results will be placed in */tmp/resul*.

Recognition of *cfile* is translated as:

`job root.722005200.a` will be run on Friday 8 Jan 1993 at `11:00:00.`

Here the local login is `root`. Two files are created as a result.

The first file `root.722005200.a` is created in */usr/spool/cron/atjobs/* to automatically run the connection and transfer at 11:00 hours invoking the command */usr/bin/transdiff*.

```
REAL_USER=root
LOGIN_USER=root
REAL_GROUP=system
GROUPS=system,bin,sys,security,cron,audit
AUDIT_CLASSES=RLIMIT_CPU=2147483647
RLIMIT_FSIZE=2097151
RLIMIT_DATA=262144
```

```
RLIMIT_STACK=65536
RLIMIT_CORE=2048
RLIMIT_RSS=65536
UMASK=22
USRENVIRON:_=/usr/bin/at
LANG=Fr_
FRNLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls
          /msg/prime/%N
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin
     /X11:/sbin
LOGNAME=root
MAIL=/usr/spool/mail/root
LOCPATH=/usr/lib/nls/loc
USER=root
DISPLAY=dpx2153:1
SHELL=/bin/ksh
ODMDIR=/etc/objrepos
HOME=/TERM=xterm
MAILMSG=[YOU HAVE NEW MAIL]
PWD=/TZ=NFT-1A_z=!
LOGNAMESYSENVIRON:NAME=root
TTY=/dev/pts/0
LOGNAME=root
LOGIN=rootumask 022
cd/
/usr/bin/transdiff /usr/spool/uft/CjOWQ1AAA/tmp/resul
```

The second file (`/tmp/resul` in the above example) in */usr/spool/uft* contains the connection parameters with the UFT commands to be run at the time specified. Each line contains one argument as follows (the second column contains comments):

```
0               -> deletion of command file
auber           -> remote host in /etc/isohosts
root            -> local login
passwd          -> local password
telecom         -> remote login
passwd          -> remote password
                -> remote billing
0               -> remote project
512             -> max record size
1               -> block transfer mode
1               -> remote machine is DPX
create /tmp/test
send /etc/isohosts as /tmp/test
send /etc/hosts as /tmp/test1
```

## Direct Call to Transdiff Command

**transdiff** ensures the connection and transfer sequence. It can be invoked directly by the user. A file must be created containing the information required for connection and transfer. This file must have the same structure as the second file created in the previous method (i. e. */tmp/resul*)

Run the command:

/**usr**/**bin**/**transdiff** *command_file result_file*

Contents of the Command File:

1. 0: this command file will be deleted after a command execution
   value other than 0: (file to be kept)
2. remote entry name in */etc/isohosts* for accessing the remote site
3. local user name (local login)

4. local user password
5. user name for remote machine accepting connection
6. remote user password
7. account number: DPS information
8. project name: DPS information
9. record size
10. transfer mode (1: block mode; 2: line mode; default: 1)
11. remote machine type:
    (1:DPX; 2:DPS 8; 3:DPS 7; 4:DPS 6; 5:OTHER)
12. uft commands:
    get–send–create–delete–readatt–modify–restart

**Example:**

```
/usr/bin/transdiff commo /tmp/resul
```

*command_file*= commo:

```
4              -> command file not to be deleted
auber          -> remote host in /etc/isohosts
root           -> local login
passwd         -> local password
telecom        -> remote login
passwd         -> remote password
               -> remote billing
               -> remote project
512            -> record size
2              -> ASCII file
1              -> DPX remote
who
readatt /tmp/test
send /etc/isohosts as /tmp/test
send /etc/hosts as /tmp/test1
```

Execution of the above command file (*commo*) produces the following result file:

*result_file*= /tmp/resul:

```
File create: available
File delete: available
Attributes request: available
File transfer restart: available
The current local directory is: /
who

You are connected to site:       auber
Local login:                     root
Remote login:                    telecom
Max record size:                 512
Transfer mode:                   line
Remote machine:                  DPX
Verbose mode:                    off
Compression mode:                off
No abort mode:                   off
Trace:                           off
Trace level:                     0
Trace file:                      -
```

```
cd /

readatt /tmp/test

Name of file:                    /tmp/test
Current size:                    35713K bytes
Record size:                     2560 bytes
File organization:               sequential
Record format:                   variable
Max file size:                   35713K bytes
Creation date time:              1992 10 06 at 10:59 GMT
Last retrieval date time:        1992 10 06 at 10:44 GMT
Last modification date time:     1992 10 06 at 10:59 GMT

send /etc/isohosts as /tmp/test

Transfer identification:         1419408883
Transfer duration:               2 seconds
Transfer traffic:                0.91K bytes/second

send /etc/hosts as /tmp/test1

Transfer identification:         1981257804
Transfer duration:               11 seconds
Transfer traffic:                0.81K bytes/second
```

## Call to uft –n

```
uft -n < command_file > result_file
```

Establishes the connection with the remote site and runs the UFT commands.

It is more flexible than `cfile` or a direct call to `transdiff`, but it does not enable differed execution.

Example

```
uft -n <commo >/tmp/resul
```

where:

*command_file*= commo:

```
co auber
root            local login
passwd          local password
telecom         remote login
passwd          remote password
blank line      remote billing
blank line      remote project
0               max record size
1024            your choice of max rec size
blank line      transfer mode
blank line      remote machine type

send /etc/isohosts as /tmp/test
readatt /tmp/test
cre /tmp/test1
send /etc/hosts as /tmp/test1
get /tmp/test1 as /tmp/comeback
who
clos
ex
```

Execution of the above command produces the following result file:

*result_file*= /tmp/resul:

```
uft>co auber
Local login:
Local password:
Remote login:
Remote Password:
Remote Billing:
Remote Project:
Max record size (0:your choice; 1:512; 2:950; 3:2000;
default: 512):
Enter max record size (between 1 and 16384):
Transfer mode (1:block mode; 2:line mode; default:1)
Remote machine type (1:DPX: 2:DPS8;3:DPS7; 4:DPS6: 5:OTHER;
default:1):
Create command: available
Delete command: available
Readatt command: available
Restart command: available
The current local directory is: /


uft>send /etc/isohosts as /tmp/test

Transfer identification:         1610757709
Transfer duration:               2 seconds
Transfer traffic:                0.87K bytes/second


uft>readatt /tmp/test

Name of file:                    /tmp/test
Current size:                    35642K bytes
Record size:                     2560 bytes
File organization:               sequential
Record format:                   variable
Max file size:                   35642K bytes
Creation date time:              1992 10 06 at 12:21 GMT
Last retrieval date time:        1992 10 06 at 10:44 GMT
Last modification date time      1992 10 06 at 12:21 GMT


uft>cre /tmp/test

Create failed
Refused by server
File already exists (reject). (601001f)


uft>send /etc/hosts as /tmp/test1

Transfer identification:         431208685
Transfer duration:               12 seconds
Transfer traffic:                0.74K bytes/second


uft>get /tmp/test1 as /tmp/comeback

Transfer identification:         1869211227
Transfer duration:               12 seconds
Transfer traffic:                0.74K bytes/second
```

```
uft>who

You are connected to site:       auber
Local login:                     root
Remote login:                    telecom
Max record size:                 1024
Transfer in mode:                block
Remote machine:                  DPX
Verbose mode:                    off
Compression mode:                off
No abort mode:                   off
Trace:                           off
Trace level:                     0
Trace file:                      -

The current directory is: /
uft>clos
uft>ex
```

# Chapter 3. Access to Non DPX Systems

## Overview

UFT offers the capability of connecting DPX systems to non–DPX systems such as the DPS 6, DPS 7 and the DPS 8 via an X25 network (public or private), an Ethernet local area network (RLE3), an FDDI network or a Token Ring.

The transmission protocols used by the different systems are:

- ISO for DPX systems,
- ISO/DSA for the DPS 6, DPS 7 and DPS 8,

DPX systems can connect to the DPS6, DPS 7 and DPS 8 through a Datanet, CPNET, or Mainway, which incorporates the PID (ISO/DSA Plug) for protocol conversion.

DPX systems can also connect directly to these systems if the PID is located on the mainframe itself.

Description of the different accesses are given in following sections:


- “Connection to Non DPX Systems”, on page 3-2,
- “Link With the DPS 7”, on page 3-3,
- “Link With the DPS 8”, on page 3-5,
- “Link With IBM Systems”, on page 3-7,
- “Link With the DPS 6”, on page 3-8.

# Connection to Non DPX Systems

To establish the connection with a non DPX system, use the command **connect** followed by the name of the remote host.

The name of the remote host must be declared in the table */etc/isohosts*. See “Installation and Configuration”, “Prerequisites” on page 4-3.

The login, password, billing and project codes of the remote host must be declared with the necessary access rights in the DPS 6, DPS 7 and DPS 8 system catalogs.

Since all identifiers are stored in capital letters on the DPS 6, DPS 7 and DPS 8, the identifiers for remote login and for known files on these systems must also be entered in capitals. See “File Identifiers on Non DPX Systems” on page B-1.

Once connected to the remote host, the list of facilities negotiated with the remote host is displayed.

All UFT commands can be executed except:

- those affecting the remote file which are not accepted by the remote host during protocol negotiation while connection is taking place:
  - **create**
  - **delete**
  - **readatt**
- those transferring a directory:
  - **sndir**
  - **rcdir**

These commands are specific to files on DPX systems.

Terminology varies from one system to another. Equivalent terms are shown in the following table:

| DPX | DPS 6 | DPS 7 | DPS 8 |
|---|---|---|---|
| **login** | user | user | userid |
| **password** | password<br>billing<br>project | password<br>billing<br>project | password |
| **working directory** | working directory | wd | UMC |
| **directory** | directory | library | catalog |
| **file** | file | member of library<br>or file | file |

# Link With the DPS 7

Once connected to the remote DPS 7, any sequence of UFT commands can be submitted.

## Types of DPS 7 Files Recommended for Transfers

On DPX systems, UFT only processes files in byte stream or sequential organization. See “Types of Files Processed by UFT” on page 1-4.

To ensure compatibility, files recommended on the DPS 7 are:

- SL source library or BIN binary library
- UFAS sequential files on disk with variable format records.

The DPS 7 does not automatically create libraries and UFAS files during the transfer, but it does create automatically a member of an existing library. Therefore, when sending files to the DPS 7, make sure that the DPS 7 has a receiving library or a receiving UFAS sequential file.

The DPS 7 can only create the member of the receiving library if the library exists.

To create receiving files on the DPS 7 through a remote console, see “File Management on DPS 7” on page C-1.

Before initiating the transfer, create the *library/file* using **create**.

## Text File Transfers with the DPS 7

Text file transfers on the DPS 7 use:

- either an SL source library:

  When sending to the DPS 7, the library member is automatically generated in SARF format. This member must be converted to SSF format in order to be printed.

  SSF format involves adding a member header and a line header for each line of the member (line number).

  The DPS 7 MAINTAIN_LIBRARY command is:

  ```
  MOVE member
  INFORM=SARF,TYPE=DAT;
  ```

  converts the library *member* from SARF format to editable SSF format.

  Library members are always transferred in SARF format, regardless of the original format. This means that line numbers may be lost on the receiving system if the member was in SSF format.

- or a UFAS sequential disk file with *variable* format records.

When connecting to the remote host:

- select *line* mode:
  - for the DPS 7 to transcode from ASCII (DPX systems) to EBCDIC (DPS 7) and vice versa,
  - to ensure that the file is divided into lines when received on DPX systems, and vice versa.
- check that the longest line to be sent to the DPS 7 is less than or equal to the maximum record size (RECSIZE) of the receiving SL source library or UFAS disk file:
  - in the case of a receiving SL source library, RECSIZE is chosen by the system when creating the SL source library (RECSIZE=256 bytes)
  - in the case of a receiving sequential UFAS disk file, RECSIZE is selected by the user when the file is created.

## Binary File Transfers with the DPS 7

Binary file transfers on the DPS 7 use:

- either a BIN library:
- or a UFAS sequential disk file with *variable* format records.

When connecting to the remote host:

- select *block* mode:
  - to transfer data in binary code,
  - and to enable the  data to be grouped into blocks when received on DPX systems, and vice versa.
- select the maximum size of transfer records, taking into account that the *max_record_size* must be less than or equal to the maximum record size of the receiving BIN library or UFAS sequential disk file:
  - in the case of a receiving BIN library, RECSIZE is selected by the system when the BIN binary library is created (RECSIZE=1024 bytes)
  - in the case of a receiving UFAS sequential disk file, RECSIZE is chosen by the user when the file is created.

# Link With the DPS 8

During UFT connection, the DPS 8 does not check the password given. When the DPS 8 is the requestor, the user name must not contain lower case letters and the user must not have a password because GCOS 8 cannot send lower case letters or passwords. Therefore, the DPS 8 cannot be the requestor when UFT is running on a C2 secure AIX which requires lower case user names and passwords.

This section only deals with **create**, **delete** and file transfer commands. The read attributes or get attributes command will abort the connection and therefore should not be used, even if it is available.

The DPS 8 does not process **readatt**.

**Note:** In the case of old GCOS versions, once connected to the remote DPS 8, only one UFT command will be accepted during a connect session. The DPS 8 does not allow the UFT user on DPX systems to perform a sequence of commands during the same connection. First disconnect using **disc**, reconnect using **connect** before submitting each command.

## Types of DPS 8 Files Recommended for Transfers

On DPX systems, UFT only processes files in byte stream or sequential organization. See “Types of Files Processed by UFT” on page 1-4.

To ensure compatibility, files recommended on the DPS 8 are:

- GFRC sequential disk files
- UFF sequential disk files
- UFF relative disk files.

**Note:** The maximum record size for GFRC files is 1272 bytes.

The receiving file is not automatically created by the DPS 8 when the transfer request is submitted. When sending a file to the DPS 8, make sure that the receiving file exists.

Create the file using **create**. See page 2-7.

## Creating Files on the DPS 8 with UFT

Before issuing **create**, set *machine type* and *file type* using **modify** and see Recommendations for Creating Files on DPS 8, on page D-1.

Example:

```
create 74502/TRANS/FIC$PASS -s 1000
```

where:

| | |
|---|---|
| 74502 | is the name of the UMC associated with the remote login on the DPS 8, |
| TRANS | is the name of the catalog which owns the access rights for UFT, |
| FIC | is the name of the file to be created, |
| PASS | is the password of the file to be created, |
| 1000 | is the initial size in Kbytes of the file to be created. |

The size of the file created is *unlimited* and will be proportionally expanded during the transfer when more space is required.

## Deleting Files on the DPS 8 with UFT

The command **delete** erases a file on a remote DPS 8.

Example:

```
delete 74502/TRANS/FIC$PASS
```

## Text File Transfers with the DPS 8

When connecting to the remote host:

- select line transfer mode to ensure that the file is transmitted in lines due to incompatibility of file structures,
- select the maximum size of transfer records taking into account that *max_record_size* must be less than or equal to the maximum size (1272 bytes) of a GFRC sequential disk file record.

## Binary File Transfers with the DPS 8

In the case of a binary file transfer, if the DPS 8 is the requestor, the option **–DT BIN** must be used to indicate binary transfer, because the default is line mode transfer.

Example:

```
TRAN –RN <remote node> –LF <local file> –RF <remote file> –DT BIN
```

When connecting to the remote host:

- select *block* mode to allow data transmission in blocks on reception on DPX systems,
- select the maximum size of records taking into account that the *max_record_size* must be less than 1019 bytes (DPS 8 restriction).

## Appending Local File to Remote File

Appending cannot be done if the remote receiving file is a UFF relative file. This limitation is due to the relative file organization.

# Link With IBM Systems

Once connected to the remote IBM system, a sequence of commands authorized within the connection can be submitted.

The remote IBM system must be running the UFTF (Unified File Transfer on a Foreign Host) application.

For the IBM system, type in the value OTHER at *remote machine type* prompt of the **connect** and **modify** commands.

The IBM system does not check the password given during the UFT connection.

## Types of IBM Files Recommended for Transfers

On DPX systems, UFT processes only files in byte stream or sequential organization. See “Types of Files Processed by UFT” on page 1-4.

To ensure compatibility, IBM files recommended for transfer are:

- QSAM sequential,
- VSAM sequential.

When sending files to an IBM system, make sure that the receiving file exists. This is because the files are not created automatically during the transfer.

## Text File Transfers with IBM Systems

When connecting to the remote host:

- set the *remote machine type* to OTHER
- select *line* mode:
  - so that the IBM system transcodes from ASCII (DPX systems) to EBCDIC (IBM systems), and vice versa;
  - to ensure that the file is transmitted in lines when received on DPX systems.
- check that the longest line to be sent to the IBM system is less than or equal to the maximum record size of the receiving file. This maximum record size is selected by the user when the file is created.

## Binary File Transfers with IBM Systems

When connecting to the remote host:

- set the *remote machine type* to OTHER
- select the *block* mode:
  - to transfer data in binary,
  - to allow the file to be transmitted in blocks when received on DPX systems.
- select the maximum size of transfer records, taking into account that *max_record_size* must be less than or equal to the maximum size of a record of the receiving file. This maximum value is selected by the user when the file is created.

# Link With the DPS 6

Once connected to the remote DPS 6 system, a sequence of UFT commands can be submitted.

If the DPS6 software release is not HVS2.20 or later, the DPS 6 string relative file is left in the DAMAGE state if the length of the last file record is null.

## Types of DPS 6 Files Recommended for Transfers

On DPX systems, UFT only processes files in byte stream or sequential organization. See “Types of Files Processed by UFT” on page 1-4.

To ensure compatibility, DPS 6 files recommended for transfer are:

- UFAS sequential
- String Relative.

When sending files to the DPS 6, make sure that the receiving file exists. Files are not created automatically by the DPS 6 during the transfer.

## Creating Files on the DPS 6 Using UFT

Before issuing **create**, set the *machine type* and *file type* using **modify** if necessary.

Types of files that can be created on the DPS 6 are:

- UFAS Sequential files with data code ASCII
- UFAS Sequential files with data code BINARY.

Example:

```
create ^B41U2D>UDD>UFTX>UFT_TEST -s 50
```

where:

| | |
|---|---|
| `^B41U2D` | is the name of the disk on which the file is to be created, (>> for default disk), |
| `UDD>UFTX` | is the directory path, |
| `UFT_TEST` | is the name of the file to be created, |
| `50` | is the initial size in Kbytes of the file to be created. |

## Deleting Files on the DPS 6 Using UFT

The command **delete** erases a file on the DPS 6.

Example:

```
delete ^BA1U2D>UDD>UFTX>UFT_DPS6
```

## Text File Transfers with the DPS 6

Text file transfers on the DPS 6 use:

- UFAS sequential files
- string Relative files

**Note:** String Relative files cannot be created using UFT on the DPS 6.

When connecting to the remote host:

- set the *remote machine type* to DPS 6,
- select *line* mode to ensure that the file is divided into lines during transmission or reception.

- select the maximum size of transfer record taking into account that:
  - *max_record_size* must be less than or equal to the record size of the file on the DPS 6
  - DPX adjusts the record size to the characteristics of the file on the DPS 6.

The receiving file must exist on the DPS 6 before the transfer.

## Binary File Transfers with the DPS 6

Binary file transfers on the DPS 6 use UFAS sequential files with datacode UNKNOWN.

However, the UFAS sequential file with datacode UNKNOWN must exist on the DPS 6 before the transfer. It cannot be created using UFT **create**.

When connecting to the remote system:

- select *block* mode to allow the file to be transmitted in blocks to the DPX systems,
- select the maximum size of records taking into account that:
  - *max_record_size* must be less than or equal to the record size of the file on the DPS 6
  - the DPX system adjusts the record size to the characteristics of the file on the DPS 6.

The receiving file must exist on the DPS 6 before the transfer.

# Chapter 4. Installation and Configuration

## Overview

This section describes how to prepare the files and tables specific to UFT and how to startup UFT.

For installing UFT and the OSI communications stack refer to How to Install the OSI Stack, in *OSI Services Reference Manual*.

UFT installation is detailed in the following sections:



## Package Contents

The media supplied (Licensed Program Product or LPP) contains the Object Program Product (OPP) `uft.cls`.

This OPP includes the UFT service daemons, the UFT command and configuration files, and the UFT APIs.

This OPP is also included into the GCOS-COM Bundle product. For this specific packaging description, refer to BASIC-COM and GCOS-COM Installation Guide.

## License Control – iFOR/LS

The Network Licensing System uses encrypted license keys for license management of software products to maintain compliance with terms of licensing agreements.

### Nodelocked License

The Licensing model used with UFT is “nodelocked”.

Nodelocking (also known as CPU locking) is a licensing mechanism requiring each node (workstation) on which the licensed software operates to obtain an authorized license for its unique System ID.

Either UFT license or GCOS–COM license is needed to run UFT.

### License Control Prerequisites

The prerequisites which apply to the licenses are derived from functional prerequisites.

To use UFT, it is mandatory to have the OSI Stack upper and lower layers (`osi_frame`, `osi_low`, `osi_high`) and therefore the associated licenses.

# UFT License Control Implementation

## UFT Requester

The local Nodelock key is tested when the **uft** process is started and at each connection command.

Once an outgoing connection has been accepted it cannot be ended by **uft** if the license becomes unavailable.

If the UFT Nodelocke license is not available, an error message similar to the following is displayed:

```
License Request Nodelocked: License not found in database
```

In the event of a NetLS error, the **uft** process stops running.

## UFT Server

The local Nodelock key is tested when the **uftser** process is started and at the time of each incoming connection.

Once an incoming connection has been accepted it cannot be ended by **uftser** if the license becomes unavailable.

If the UFT Nodelocke license is not available, an error message similar to the following is displayed:

```
License Request Nodelocked: License not found in database
```

Error messages are displayed during the “Start UFT” smit menu.

The UFT server cannot be started without the NetLS key. The server stops running if the license becomes unavailable. Although it does not stop previously accepted connections, it accepts no new ones.

## Connection to a Host with an Invalid License

When a requester issues a connect command, the incoming connection on the host side can be refused if the host’s NetLS key is unavailable. An error message similar to the following is displayed:

```
Connection failed
Remote has hung up (0x87010000)
```

or else:

```
Connection failed
Remote UFT server: Inactive or saturated (0x85000382)
```

# Installing UFT

## Prerequisites

### Software

The following OPPs (Optional Program Products) must be installed prior to installing UFT:

- `bos.rte`, version 4, revision greater than 0
- `osi_high.rte` and its prerequisites.

### Disk

To install the LPP (Licensed Program Product), the disk space required is 2 Mbytes.

## Procedure

### General

All LPPs are installed using SMIT (System Management Interface Tool).

### Installation for Standard Stations

LPPs are installed on standard stations using the following sequence of menus in SMIT:

- systems management
- software installation and maintenance
- install/update software
- install/update selectable software
- install software products at latest available level.

Complete the *input_device* field and then set the appropriate options.

The previous sequence of menus is subject to change. Refer to the appropriate AIX guide.

### Diskless Stations

UFT does not support diskless stations.

### Bundle Installation

Refer to BASIC-COM and GCOS-COM Installation Guide for the GCOS-COM Bundle installation facilities.

# Configuring UFT

UFT configuration is performed in three steps:

- Configuring the Communications Stack (mandatory),
- Configuring NetShare (for NetShare sites only),
- Managing remote sites.

## Configuring the Communications Stack

To use the SMIT configurator, refer to "OSI Stack Configuration", in *OSI Services Reference Manual*.

The number of simultaneous incoming and outgoing connections is not limited by UFT. It is, however limited by the number of available connections of the communications stack and the memory available. Refer to the *OSI Services Reference Manual*.

An easy configurator for both OSI Stack and UFT is available through the GCOS-COM Bundle. Refer to BASIC-COM and GCOS-COM Installation Guide for detailled description.

Only the detailled configuration of the UFT product itself is described in this document.

## Configuring NetShare

NetShare must be configured only if a NetShare site is to be connected.

To run UFT on top of TCP–IP, refer to the *NetShare User's Guide*.

# Managing Remote Sites

## Enter the SMIT Configuration Tool

**Note:** Some menus contain multiple choice parameters (a + character is displayed at the right side of the parameter). Use Esc+4 to display the list.

Enter the following command:

`smit` if you are using an X terminal,

`smit -C` if you are using an ASCII terminal.

The first menu which appears using the `smit -C` command is:

```
                        System Management
Move cursor to desired item and press Enter.



    Software Installation & Maintenance
    Devices
    Physical & Logical Storage
    Security & Users
    Diskless Workstation Management & Installation
    Communications Applications and Services
    Spooler (Print Jobs)
    Problem Determination
    Performance & Subsystems
    Applications
    Using SMIT (information only)



F1=Help       F2=Refresh        F3=Cancel       Esc+8=Image
Esc+9=Shell   Esc+0=Exit        Enter=Do
```

Select the `'Communications Applications and Services'` item and Press Enter.

```
            Communications Applications and Services
Move cursor to desired item and press Enter.


    Communication Bundle
    TCP/IP
    NFS
    TPAD-HPAD
    Unified File Transfer (UFT)
    NetShare
    XTI
    OSI Networking
    OTM Configurations

F1=Help       F2=Refresh        F3=Cancel       Esc+8=Image
Esc+9=Shell   Esc+0=Exit        Enter=Do
```

Select the 'Unified File Transfer (UFT)' item and Press Enter.

Fastpath = **smit [–C] uft**

```
                  Unified File Transfer (UFT)
Move cursor to desired item and press Enter.


    Installation
    Start UFT
    Stop UFT
    Restart Information Log
    Utilities


F1=Help       F2=Refresh        F3=Cancel       Esc+8=Image
Esc+9=Shell   Esc+0=Exit        Enter=Do
```

Select the 'Installation' item and Press Enter.

```
                          Installation
Move cursor to desired item and press Enter.


    List site entries in /etc/isohosts
    Remove site entries in /etc/isohosts
    Add site entries in /etc/isohosts


F1=Help       F2=Refresh        F3=Cancel       Esc+8=Image
Esc+9=Shell   Esc+0=Exit        Enter=Do
```

Select the 'Add site entries in /etc/isohosts' item and Press Enter.

```
               Add site entries in /etc/isohosts
Move cursor to desired item and press Enter.


    Add ETHERNET site entry in /etc/isohosts
    Add X25 site entry in /etc/isohosts
    Add FULL-IP or NULL-IP site entry in /etc/isohosts
    Add a NETSHARE site entry in /etc/isohosts


F1=Help       F2=Refresh        F3=Cancel       Esc+8=Image
Esc+9=Shell   Esc+0=Exit        Enter=Do
```

Adding an ETHERNET remote site, go to page 4-7

Adding an X25 remote site, go to page 4-8

Adding a FULL–IP or NULL–IP remote site, go to page 4-9

Adding a NETSHARE remote site, go to page 4-10

## Adding an Ethernet Remote Site

After selecting the 'Add ETHERNET site entry in /etc/isohosts' item in the 'Add site entries in /etc/isohosts' menu, the following menu appears.

```
           Add an ETHERNET site entry in /etc/isohosts

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                [Entry Fields]
*   Site name of host to add                     []
*   Local TSAP                                   [0x40,1,2,3,4,5]
*   Local ETHERNET address                       []             +
*   Remote TSAP                                  []
*   Remote ETHERNET address                      []


    F1=Help     F2=Refresh      F3=Cancel       F4=List
    Esc+5=Reset Esc+6=Command   Esc+7=Edit      Esc+8=Image
    Esc+9=Shell Esc+0=Exit      Enter=Do
```

**Site name of host to add**

Must not contain spaces or unprintable characters and is taken as the argument for `connect`.

**Local TSAP**

List of digits corresponding to local TSAP (Transport Service Access Point).

**Remote TSAP**

List of digits corresponding to remote TSAP (Transport Service Access Point). If the remote site is DPS 7/7000 or DPS 8, the remote TSAP must be 0x40, 0x01, *site name*.

**Local or Remote Ethernet Address**

An Ethernet address is a sequence of 6 bytes that are generated automatically with the dotted Hexadecimal form.

Each byte is coded using the characters "0" to "9" and the uppercase or lowercase form of the letters "abcdef". The bytes are separated by a "." (a dot) or ":" (a colon).

Example: `02:60:8c:2e:52:47 or 02.60.8c.2e.52.47`

## Adding an X25 Remote Site

After selecting the 'Add X25 site entry in /etc/isohosts' item in the 'Add site entries in /etc/isohosts' menu, the following menu appears.

```
              Add an X25 site entry in /etc/isohosts

  Type or select values in entry fields.
  Press Enter AFTER making all desired changes.

                                                    [Entry Fields]
  *   Site name of host to add                       []
  *   Local X25 address                              []              +
  *   Local TSAP                                     [0x40,1,2,3,4,5]
  *   Remote TSAP                                    [0x40,1,2,3,4,5]
  *   Switched or Permanent Virtual Circuit          [SVC]           +
                                  (SVC or PVC)
  *   Remote X25 address (or PVC name if PVC         []
                                  selected)
  *   Network facilities (list of digits hexa or     [none]
                                  decimal)


      F1=Help      F2=Refresh      F3=Cancel       F4=List
      Esc+5=Reset  Esc+6=Command   Esc+7=Edit      Esc+8=Image
      Esc+9=Shell  Esc+0=Exit      Enter=Do
```

**Site name of host to add**

Must not contain spaces or unprintable characters and is taken as the argument for `connect`.

**X25 Address (SVC or PCV)**

A SVC X25 address is a sequence of up to 15 bytes that can be entered in decimal BCD format (Binary Coded Decimal). Only the characters '0" to "9" are allowed.

A PVC X25 Virtual circuit number is a string composed of exactly 6 alphanumeric characters and exactly 2 numeric characters in the range 0 to the maximum number of PVCs defined on the X25 board.

Example: `175457882214 for a SVC`

Example: `dieppe02 for a PVC`

**Local TSAP**

List of digits corresponding to local TSAP (Transport Service Access Point).

**Remote TSAP**

List of digits corresponding to remote TSAP (Transport Service Access Point). If the remote site is DPS 7/7000 or DPS 8, the remote TSAP must be 0x40, 0x01, *site name*.

**Switched or Permanent Virtual Circuit**

SVC or PVC, depending on the Public Network.

**Network facilities**

depend on the Public Network.

## Adding a FULL–IP or NULL–IP Remote Site

After selecting the 'Add FULL-IP or NULL-IP site entry in /etc/isohosts' item in the 'Add site entries in /etc/isohosts' menu, the following menu appears.

```
        Add FULL-IP or NULL_IP site entry in /etc/isohosts

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                [Entry Fields]
*   Site name of host to add                     []
*   Local TSAP                                   [0x40,1,2,3,4,5]
*   Local NSAP address                           []             +
*   Remote TSAP                                  [0x40,1,2,3,4,5]
*   Remote NSAP address                          []
*   FULL-IP or NULL-IP                           [FULL-IP]      +


    F1=Help      F2=Refresh      F3=Cancel       F4=List
    Esc+5=Reset Esc+6=Command    Esc+7=Edit      Esc+8=Image
    Esc+9=Shell Esc+0=Exit       Enter=Do
```

**Site name of host to add**

Must not contain spaces or unprintable characters and is taken as the argument for `connect`.

**Local TSAP**

List of digits corresponding to local TSAP (Transport Service Access Point).

**Remote TSAP**

List of digits corresponding to remote TSAP (Transport Service Access Point). If the remote site is DPS 7/7000 or DPS 8, the remote TSAP must be 0x40, 0x01, *site name*.

**Local or Remote NSAP Address**

The two main parts of an NSAP address are IDP and DSP, as follows:

The IDP part is composed of two components:

- Authority Format Identifier (AFI) which is used to specify:
  - the IDI format
  - the authority in charge of allocating IDI values
  - the abstract syntax of the DSP.
- Initial Domain Identifier (IDI) which, depending on the AFI allocation value, specifies:
  - the addressing domain in which the DSP values have been allocated
  - the authority in charge of allocating DSP values in this domain.

The DSP part gives information on the system position in relation to the domain where it belongs.

For more information about NSAP address, refer to NSAP Syntax in *OSI Services Reference Manual*.

**FULL–IP or NULL–IP**

Choose FULL–IP if the OSI network uses the full OSI Network Layer. Otherwise choose NULL–IP.

## Adding a NETSHARE Remote Site

After selecting the 'Add a NetShare site entry in /etc/isohosts' item in the 'Add site entries in /etc/isohosts' menu, the following menu appears.

```
            Add a NETSHARE site entry in /etc/isohosts

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                  [Entry Fields]
*   Site name of host to add                       []
*   Local TSAP                                     [0x40,1,2,3,4,5]
*   Local NETSHARE address                         []              +
*   Remote TSAP                                    [0x40,1,2,3,4,5]
*   Remote NETSHARE address                        []


    F1=Help     F2=Refresh      F3=Cancel       F4=List
    Esc+5=Reset Esc+6=Command   Esc+7=Edit      Esc+8=Image
    Esc+9=Shell Esc+0=Exit      Enter=Do
```

**Site name of host to add**

Must not contain spaces or unprintable characters and is taken as the argument for `connect`.

**Local TSAP**

List of digits corresponding to local TSAP (Transport Service Access Point).

**Remote TSAP**

List of digits corresponding to remote TSAP (Transport Service Access Point). If the remote site is DPS 7/7000 or DPS 8, the remote TSAP must be 0x40, 0x01, *site name*.

**Local or Remote NETSHARE Address**

For information about the NETSHARE address, refer to the *NetShare User's Guide*.

# Activating UFT

UFT can be started in one of the following ways:

- using SMIT
- manually by entering /**etc**/**rc.uft.**
- automatically

## Starting UFT Using SMIT

Enter the SMIT Configuration Tool

Fastpath = **smit [–C] uft**, select '`Start UFT`'.

1. Select the `Communications Applications and Services` item and Press Enter,
2. Select the `Unified File Transfer (UFT)` item and Press Enter.

Select '`Start UFT`' item and Press Enter.

After selecting the '`Start UFT`' item in the '`Unified File Transfer (UFT)`' menu, the following menu appears.

```
                              Start UFT

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                    [Entry Fields]
*   Number of UFT servers to start                   [1]           +
*   Trace level for UFT server(s)                    [-10]         +
    Trace file for UFT server(s)                     []
*   Allow trace in UFT command                       [no]          +


    F1=Help     F2=Refresh      F3=Cancel       F4=List
    Esc+5=Reset Esc+6=Command   Esc+7=Edit      Esc+8=Image
    Esc+9=Shell Esc+0=Exit      Enter=Do
```

This menu is used to select the number of UFT servers to run simultaneously, the trace level for the server(s) and the file(s) where the traces are to be redirected. There are as many trace files, suffixed by a number, as servers running simultaneously.

One or two UFT servers can be started at the same time. If there are two UFT servers, there are fewer lost connections on simultaneous connections and reduced saturation.

## Starting UFT Manually

The standard shell command to type to start UFT is /**etc**/**rc.uft .** The full command syntax with options is as follows:

```
/etc/rc.uft [-v] <[start]|[stop]> [number of servers]
[trace level] [trace filename] <[yes]|[no]>
```

The option **–v** activates verbose mode.

The option **start** starts UFT. It is a default option, therefore UFT is started even if it is not specified.

Number of servers: 1 or 2, the default is 1.

The trace level specified can be –l0 or –l1 or –l2 or –l3; level 0 is the default. The meaning of these trace levels is as follows:

| | |
|---|---|
| l0 | no trace. |
| l1 | only error messages and internal exchanges are traced. |
| l2 | error messages and calls to the communication stack are traced. |
| l3 | full trace level, includes dump of data exchanges, functional calls, etc. |

The **trace filename**, if specified by the user stores trace log information.

The option **yes**|**no** is used to specify whether the trace is allowed in the **uft** command. The default setting is no.

Only the root user has permission to start the /**etc**/**rc.uft** shell script.

## Starting UFT Automatically

When UFT has been installed and started once, at each system reboot it is automatically restarted because /**etc**/**rc.uft** is executed automatically.

# Deactivating UFT

UFT can be stopped in one of the following ways:

- using SMIT
- manually on entering /**etc/rc.uft stop**.

## Stopping UFT Using SMIT

Enter the SMIT Configuration Tool

Fastpath = **smit [–C] uft**

1. Select the `Communications Applications and Services` item and Press Enter,
2. Select the `Unified File Transfer (UFT)` item and Press Enter.

Select '`Stop UFT`' item and Press Enter.

After selecting the '`Stop UFT`' item in the '`Unified File Transfer (UFT)`' menu, the following menu appears.

```
                              ARE YOU SURE?
  Continuing may delete information you may want
  to keep. This is your last chance to stop
  before continuing.
      Press Enter to continue.
      Press Cancel to return to the application.


  F1=Help       F2=Refresh        F3=Cancel
  Esc+9=Shell   Esc+0=Exit        Enter=Do
```

## Stopping UFT Manually

The command syntax for stopping UFT is the same as for starting UFT (see section “Starting UFT Manually” on page 4-11), but it is necessary to specify the option **stop** because it is not the default option.

# Managing Interrupted Transfer Using SMIT

The SMIT menu “Restart Information Log” enables the user to handle interrupted transfer data for both the requester and the server side. It comes under the main “Unified File Transfer” menu.

The cascading submenus are the same for the requester and the server. Therefore, only the Server Restart Information Log is shown in detail below.

Enter the SMIT Configuration Tool

Fastpath = **smit [–C] uft**

1. Select the `Communications Applications and Services` item and Press Enter,
2. Select the `Unified File Transfer (UFT)` item and Press Enter.

Select '`Restart Information Log`' item and Press Enter.

Select '`Server Restart Information Log`' item and Press Enter.

The cascading submenus are the same for the requester and the server. Therefore, only the Server Restart Information Log is shown in detail below.

```
                Server Restart Information Log

    Remove information log
    Show information log
```

```
                    Remove information log

    Remove identifier
    Remove all identifiers
```

```
                       Remove identifier

    Restart identifier                              []
```

At this stage the user must specify the identifier associated with the information log file to be removed. This identifier can be selected in the list of all identifiers. Below is an example of the information displayed on pressing the Enter key (see figure):

```
#transfer id:
20143896420000000000
#local login: root
#remote node: @JUC4
#local file name: /tmp/_U09210.Z
################################################
#transfer id:
19167822920000000000
#local login: marcel
#remote node: @JUC1
#local file name: /users/marcel/guionnea/new_LIB.a
################################################
#transfer id:
14360444250000000000
#local login: uftex3
#remote node: @BS3D
#local file name: /tmp/_U09910.Z
################################################
```

Cancel

```
Remove all identifiers
```

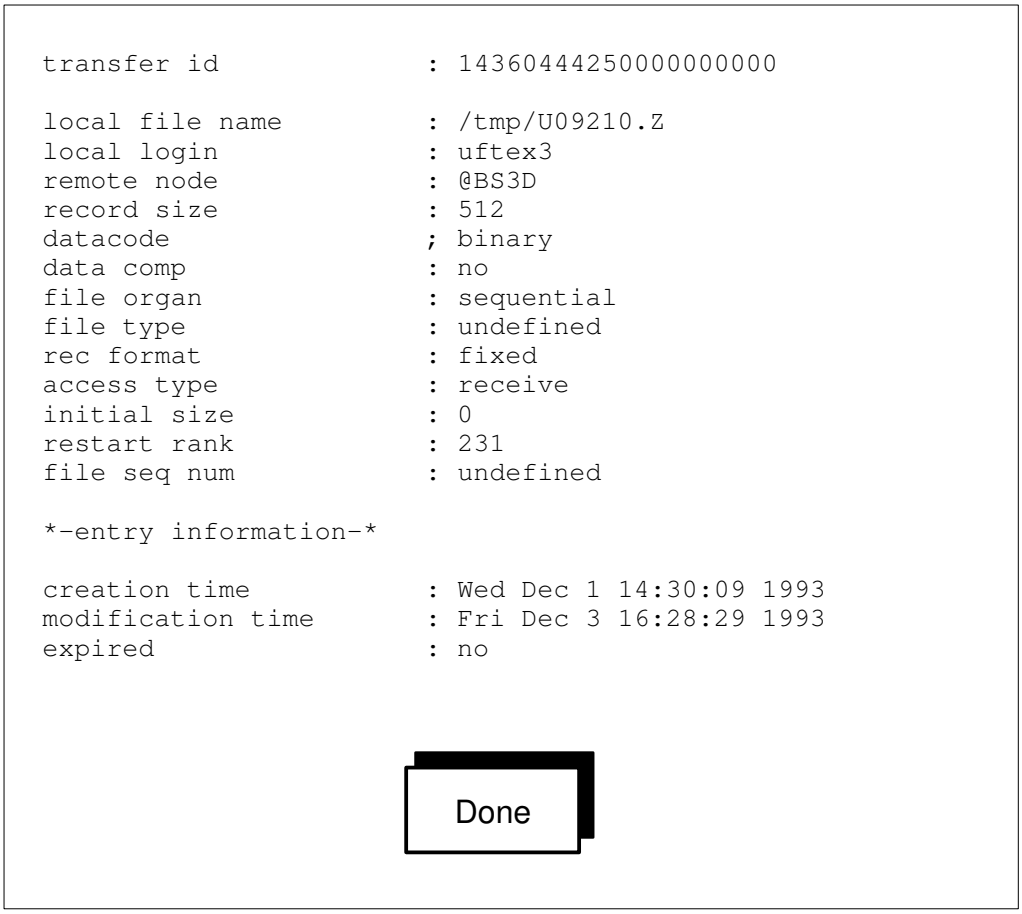
This command removes all identifiers.

```
                    Show information log
Show identifier
Show all identifiers
```

```
                        Show identifier
Restart identifier                              []
```

At this stage the user must specify the identifier associated with the information log file the user requires. This identifier can be selected in the list of all identifiers using the “list” button. See the example given above of the information displayed on pressing the Enter key.

On entering the restart identifier, the complete description of the associated entry is displayed in the following form:





This command displays the description of every entry in the restart information log.

# Chapter 5. UFT Application Program Interface

## Overview

The purpose of this chapter is to describe the Application Program Interface (API) of UFT V3.1 (Unified File Transfer). The UFT API V3.1 is delivered with the standard UFT V3 MI.

### General Features

The UFT API supplies all the functions of the requester (uft client) process of UFT. These functions enable a user to create an application that executes file transfer, file creation, file deletion... with a remote host supporting the UFT protocol.

### Operation

An application linked with the UFT API can perform the functions of UFT. The UFT API is a library and an include (*.h*) file containing the needed declarations.

### Compatibility

This API is incompatible with the unsupported UCB–API of UFT V2.x. A user application written over the UCB–API of UFT V2 must be rewritten to use the UFT V3 API.

An user application built over the UFT V3.1 can cooperate with every UFT server.

### External Dependencies

UFT must be installed on the machine executing a user application build using this API in order to:

- configure the file containing the description of the remote hosts (*/etc/isohosts*).
- execute the user application. UFT API checks that a local UFT server is running on the local host.

### File and Data Formats

The UFT API uses the same configuration file (*/etc/isohosts*) as the standard UFT product. This file and the data format used is described in ”Files processed by UFT”, on page 1-4.

## List of UFT API Functions

The UFT API functions must be called in a fixed order:

- **UFTinit()** is the first function to call. This function initializes the API, checks that a local UFT server is running on the local machine and sets up internal UFT API variables.
- **UFTtrace()** to set/reset the trace facility.
- **UFTcmp()** to compress transfer files.
- **UFTconnect()** is the function for connecting to a remote host with which the file transfers will be done.
- **UFTnoabort()**, **UFTmodify()** to set up API flags.
- **UFTinterrupt()** to set up a interrupt function used during file transfer.
- **UFTrec()**, **UFTsend()**, **UFTcreate()**, **UFTgetattribut()**, **UFTdelete()**, **UFTrestart()**, are the functions to manage file handling.
- **UFTwho()** to list the connection characteristics.

- **UFTdisconnect()** to disconnect from the remote host.

The states of UFT API are:

| States<br>Functions | 0<br>Reset | 1<br>Initialized | 2<br>connected |
|---|---|---|---|
| **UFTinit()** | 1 | 1 | 2 |
| **UFTconnect()** | NA | 2 | NA |
| **UFTdisconnect()** | NA | NA | 1 |
| **UFTsend()** | NA | NA | 2 |
| **UFTrec()** | NA | NA | 2 |
| **UFTdelete()** | NA | NA | 2 |
| **UFTcreate()** | NA | NA | 2 |
| **UFTgetattribut()** | NA | NA | 2 |
| **UFTnoabort()** | NA | NA | 2 |
| **UFTtrace()** | NA | 1 | 2 |
| **UFTcmp()** | 0 | 1 | 2 |
| **UFTmodify()** | NA | NA | 2 |
| **UFTrestart()** | NA | NA | 2 |
| **UFTwho()** | NA | NA | 2 |

Table 1. UFT API states

## User Interface Descriptions

The user interface is a program interface. This program interface (*/usr/lib/libUFTapi.a*) needs the description of UFT structures defined in */usr/include/UFTapi.h*.

## Procedure Call Interface General Mechanisms

Each function of the UFT API has the following profile:

int **UFTxxx** (*parm1, parm2...,return_code*)

The return of each function is a bit mask. If this return value is UFT_OK there is no error, otherwise this return value contains a bit error field. The possible value of this bit field differs for each UFT API function.

*return_code* is significant in some cases (defined for each function) and contains a more detailed error description.

Each function of the API is synchronous and their return is made only when UFT has done every protocol exchange needed.

# UFTinit()

## Purpose

Initialization of the UFT API.

## Syntax

**int UFTinit**(*return_code*)

**unsigned int *** *return_code*

## Description

This function initializes the UFT API and checks that a local UFT server is currently working.

## Parameters

*return_code* Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

## Return Code Values

UFT_OK: no error.

UFT_INIT_PROBLEM:
the initialization can not be performed, the UFT local server is not running or can not be accessible through the session loop back, or UFT is not installed on the local host.

## State Changes

The UFT API is initialized if no error occurs.

## Application Usage

This is the first function to call before any other.

# UFTconnect()

## Purpose

Connection to a remote host.

## Syntax

int **UFTconnect** (*table_site,lg_table_site,remote_host, remote_login, remote_passwd, remote_billing, remote_project, max_record_size, transfer_mode, remote_machine_type, GCOS8_file_type, return_code*)

struct TABSIT * *table_site*

unsigned int *lg_table_site*

char * *remote_host*

char * *remote_login*

char * *remote_passwd*

char * *remote_billing*

char * *remote_project*

unsigned int * *max_record_size*

unsigned int *transfer_mode*

unsigned int *remote_machine_type*

char *GCOS8_file_type*

unsigned int * *return_code*

## Description

This function opens a connection with the remote host using the remote login information.

## Parameters

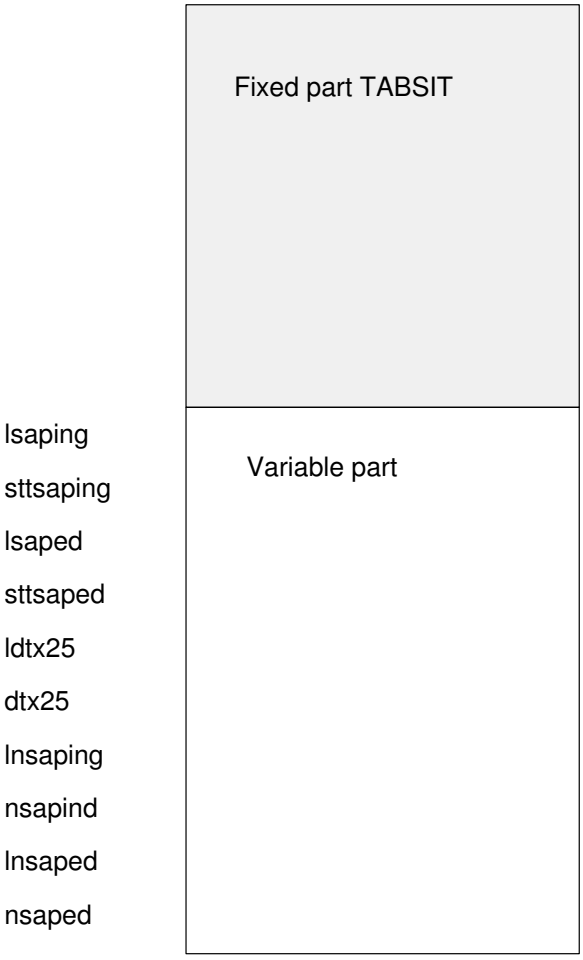
*table_site*

The *table_site* parameter addresses the TABSIT fixed part and variable part.

This structure contains the description of the remote hosts and it is used if the remote_host parameter is not initialized (equal to NULL).

```
struct TABSIT {
        unsigned char stcl ;
        unsigned char staltcl ;
        unsigned short stexp ;
        unsigned short stctrl ;
        unsigned short stcrdt ;
        unsigned short stltpdu ;
        unsigned short stcheck ;
        unsigned short stlnsdu ;
        unsigned short stlnpdu ;
        unsigned char stwi ;
        unsigned char stwo ;
        unsigned char stypres ;
        unsigned char stcnx ;
        struct tadrres stfrna ;
        struct tadrres sttona ;
        struct qostype stqos ;
} ;
```

```
struct tadrres {
        unsigned short ladrres;
        unsigned char typadr[8];
} ;
struct qostype {
        unsigned short lfac;
        unsigned char typfac[109];
} ;
```




**Fixed part**

| | |
|---|---|
| stcl | class of transport: 0, 2, 3,4 |
| staltcl | alternate class: 0, 2, 3, 4, 0xFF |
| stexp | use of express stream 1, 0 (need to be set to 1 for UFT) |
| stctrl | stream check: 0, 1 |
| stcrdt | transport credit: from 1 to 15 |
| stltpdu | size of tpdu in bytes, from 128 to 8192 |
| stcheck | total check for class 4: 0,1 |
| stlnsdu | size of nsdu in bytes |
| stlnpdu | size of npdu in bytes |
| stwi | network receive window from 1 to 127 |

| | |
|---|---|
| stwo | network transmit window from 1 to 127 |
| stypres | network type:<br>1: X25, WAN with SNPA address<br>2: Ethernet with SNPA address<br>3: FULL–IP with NSAP address<br>4: NULL–IP with SNPA address<br>5: SPEE/Netshare<br>6: X25,with PVC |
| stcnx | connection type (0 with connection, 1 without connection) |
| stfrna | local address (SNPA) |
| sttona | remote address (SNPA) |
| stqos | facility linked to network (e.g., TRANSPAC facilities) |

**variable part**

Lengths are coded with one byte. The information in the variable part is the same as that contained in the */etc/isohosts* file (see ”Structure of the Remote Hosts Table, on page G-1).

| | |
|---|---|
| lsaping | length of the sttaping field (value: 0 to 16) |
| sttsaping | local TSAP |
| lsaped | length of the tsaped field (value: 0 to 16) |
| tsaped | remote TSAP |
| ldtx25 | length of the dtx25 field |
| dtx25 | data X25.3 for calling packet |
| lnsaping | length of the nsaping field (value: 0 to 16) |
| nsaping | local address (NSAP) |
| lnsaped | length of the nsaped field (value: 0 to 16) |
| nsaped | remote address (NSAP) |

*lg_table_site*

Contains the size of the *table_site* (fixed part + variable part).

*remote_host*

It is a string containing a valid host name configured in the /etc/isohosts configuration file. If the *remote_host* is equal to NULL the *table_site* parameter is used.

*remote_login*

Remote login used during the UFT connection.

*remote_passwd*

Remote password used during the UFT connection.

If NULL no password.

*remote_billing*

Remote billing used during the UFT connection (up to 12 characters).

*remote_project*

Remote project used during the UFT connection (up to 12 characters).

*max_record_size*
Maximum record size negotiated during the connection phase. (see ”Connecting to a remote Host”, on page 2-15 for the limits and the used of this parameter).

*transfer_mode* The transfer mode can be UFT_MODE_LINE for line mode, or UFT_MODE_BLOC for block mode (see ”File Transfer Modes”, on page 1-4).

*remote_machine_type*
This parameter defines the remote machine type:

DPX: for the DPX and Pegasus range

DPS8: for the DPS8 range

DPS7: for the DPS7 range

DPS6: for DPS6 range

OTHER: for IBM range and other existing UFT

*GCOS8_file_type*
This parameter is used only if the *machine_type* is DPS8. It defines the type of GCOS8 file which is handled during transfers, file creation,... The possible values for this parameter are:

G8_GFRC_FILE

G8_UFAS_REL_FILE

G8_UFAS_SEQ_FILE

*return_code* Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

## Return Code Values

UFT_OK: The UFT command was successful.

UFT_PARM_ERR:
A parameter of the UFT function is incorrect.

UFT_ALREADY_CON:
A connection is already established, the state is already connected.

UFT_ERR: An error occurs in the UFT protocol. The UFT function cannot be performed. The *return_code* contains the UFT error number.

UFT_COM_ERR:
An error occurs in the communication stack. The *return_code* contains a communication error, and the connection is in disconnected state.

UFT_INIT_PROBLEM:
If the function **UFTinit()** was not previously issued.

## State Changes

If no error occurs, the connection is initialized to the remote host, the connection state is effective.

## Application Usage

The UFT API must be in initialized state before issuing the **UFTconnect()** (see **UFTinit()** function). **UFTconnect()** establishes the connection link to the remote host using the remote information. The connection is established with the remote host using the name given in the *remote_host* parameter, or if this parameter is equal to NULL, the connection is established with the site described in the *table_site* parameter.

## Related Information

See also the **connect** subcommand, and the ”Error Messages”, on page A-1.

# UFTdisconnect()

## Purpose

Disconnection from a remote host.

## Syntax

**int UFTdisconnect**(*return_code*)

**unsigned int *** *return_code*

## Description

This function closes a connection with the remote host.

## Parameters

*return_code*

Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

## Return Code Values

UFT_OK: The function UFTdisconnect is successful.

UFT_ERR: An error occurs from UFT protocol, *return_code* contains the description of the error.

UFT_COM_ERR:
An error occurs in the communication stack, the *return_code* contains a communication error, and the connection is in disconnected state.

UFT_NO_CONNECT:
There is no connection, a **UFTconnect** must be issued first.

## State Changes

If no error occurs, the connection is closed. The connection state is disconnected. The UFT API is in initialized state and ready for another connection.

## Application Usage

The connection must be in connected state to issue the UFTdisconnect function.

## Related Information

See also the **disc** subcommand, and the ”Error Messages”, on page A-1.

# UFTsend()

## Purpose

Send a file to the remote host already connected.

## Syntax

int **UFTsend** (*local_file_name, send _mode, remote_filename, return_code, transfer_id*)

char * *local_file_name*

int *send_mode*

char **remote _file_name*

unsigned * *return_code*

char ** *transfer_id*

## Description

The file (*local_file_name*) is sent to the remote host.

## Parameters

*local_file_name*

It is the full name of the file to send (e.g.: */users/cop/bin/file.uft*).

*send_mode*

The send mode can have the following values:

UFT_AS: The local file is sent to the remote host to the remote file, the remote file is cleared before the local site is sent.

UFT_AP: The local file is sent to the remote host and appended to the remote file.

*remote_file_name*

It is the full name of the remote file. See ”File Identifiers on Non-DPX Systems”, on page B-1, for the syntax of other files of non-unix machines.

*return_code*

Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

*transfer_id*

The *transfer_id* is the identification of the transfer. It can be used to restart the transfer if it is interrupted (see **UFTrestart ()**).

## Return Code Values

UFT_OK: The function **UFTsend()** is successful.

UFT_ERR: An error occurs from UFT the protocol. *return_code* contains the description of the error.

UFT_COM_ERR:
An error occurs in the communication stack. The *return_code* contains a communication error and the connection is in disconnected state.

UFT_TRANS_REST:
The transfer can be restarted.

UFT_NO_CONNECT:

There is no connection. A **UFTconnect()** must be issued first.

## State Changes

If no error occurs, the connection state remains connected. If UFT_COM_ERR occurred the connection state may change to disconnected.

## Application Usage

The connection state must be connected.

## Related Information

See also the **send** subcommand, and the ”Error Messages”, on page A-1.

# UFTrec()

## Purpose

Receive a file from a remote host already connected.

## Syntax

int **UFTrec** (*remote_file_name, rec_mode, local_file_name, return_code, transfer_id*)

char * *remote_file_name*

int *rec_mode*

char * *remote_file_name*

unsigned int * *return_code*

char ** *transfer_id*

## Description

The file (*remote_file_name*) is received from the remote host.

## Parameters

*remote_file_name*

It is the full name of the file to be received. See ”File Identifiers on Non DPX Systems”, on page B-1, for the syntax of other files of non Unix machines.

*send_mode*

The send mode can have the following values:

UFT_AS: The remote file is received from the remote host in the local file. The local file is cleared before the remote site is received.

UFT_AP: The remote file is received from the remote host and appended to the local file.

*local_file_name*

It is the full name of the local file.

*return_code*

Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

*transfer_id*

The *transfer_id* is the identification of the transfer. It can be used to restart the transfer if it is interrupted (see **UFTrestart()**).

## Return Code Values

UFT_OK: The function **UFTrec()** is successful.

UFT_ERR: An error occurs from the UFT protocol. *return_code* contains the description of the error.

UFT_COM_ERR:
An error occurs in the communication stack. The *return_code* contains a communication error and the connection is in disconnected state.

UFT_TRANS_REST:
The transfer can be restarted.

UFT_NO_CONNECT:

There is no connection. A **UFTconnect()** must be issued first.

## State Changes

If no error occurs, the connection is still initialized to the remote hosts. The connection state remains connected. If UFT_COM_ERR occured the connection state can be disconnected.

## Application Usage

The connection state must be connected.

## Related Information

See also the **receive** subcommand, and the ”Error Messages”, on page A-1.

# UFTdelete()

## Purpose

Delete a file of a remote host already connected.

## Syntax

int **UFTdelete**(*remote_file_name, return_code*)

char **remote_file_name*

unsigned int *return_code*

## Description

The file (*remote_file_name*) is removed from the remote host.

## Parameters

*remote_file_name*

The name of the file to be removed. See ”File Identifiers on Non DPX Systems”, on page B-1 for the syntax of other files for non UNIX machines.

*return_code*

Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

## Return Code Values

UFT_OK: The function **UFTdelete()** is successful.

UFT_ERR: An error occurs from UFT protocol, *return_code* contains the description of the error.

UFT_COM_ERR:
an error occurs in the communication stack. The *return_code* contains a communication error, and the connection is in disconnected state.

UFT_NO_CONNECT:
There is no connection. An **UFTconnect()** must be issued first.

## State Changes

If no error occurs, the connection is still initialized to the remote hosts. The connection state remains connected. If UFT_COM_ERR occured, the connection state can be disconnected.

## Application Usage

The connection state must be connected.

## Related Information

See also the **delete** subcommand, and the ”Error Messages”, on page A-1.

# UFTcreate()

## Purpose

Create a remote file.

## Syntax

int **UFTcreate** (*remote_file_name, file_size, return_code*)

char **remote_file_name*

int *file_size*

unsigned int * *return_code*

## Description

The file (*remote_file_name*) is created to the remote host.

## Parameters

*remote_file_name*

This is the name of the file to be created. See ”File Identifiers on Non-DPX Systems”, on page B-1 for the syntax of other files of non-unix machines.

*file_size*

The initial size of the file to be created.

*return_code*

Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

## Return Code Values

UFT_OK: The function **UFTcreate()** is successful.

UFT_ERR: An error occurs from UFT protocol, *return_code* contains the description of the error.

UFT_COM_ERR:
An error occurs in the communication stack, the *return_code* contains a communication error and the connection is in disconnected state.

UFT_NO_CONNECT:
There is no connection. An **UFTconnect()** must be issued first.

## State Changes

If no error occurs, the connection is still initialized to the remote hosts. The connection state remains connected. If UFT_COM_ERR occured the connection state can be disconnected.

## Application Usage

The connection state must be connected.

## Related Information

See also the **create** subcommand, and the ”Error Messages”, on page A-1.

# UFTgetattribut()

## Purpose

Get attributes of a remote file.

## Syntax

int **UFTgetattribut** (*remote_file_name, attributs, return_code*)

char * *remote_file*

UFTattrib * *attributs*

unsigned int * *return_code*

## Parameters

*remote_file_name*

The name of the remote file from which to get the attributs.

*attributs*

[A structure UFTattrib containing all the attributs of the file.

```
struct {
        unsigned int recs;
        unsigned char file_organ ;
        int file_seq_num ;
        char file_type;
        char record_format;
        unsigned int inc_alloc_size;
        unsigned int max_alloc_size;
        int cur_alloc_size;
        unsigned char alloc_size_unit;
        date creat_date ;
        date last_ret_date ;
        date last_mod_date ;
} UFTattrib ;
```

recs — the record size of the file

file_organ — the file organization:

UFT_SEQUENTIAL
UFT_RELATIVE
UFT_INDEXED
UFT_RANDOM
UFT_UNSTRUCTURED
UFT_QUEUED
UFT_BYTE_STREAM
On DPX only UFT_SEQUENTIAL file (binary file) and UFT_BYTE_STREAM file (ASCII file) are supported.

file_seq_num — file number for tape storage.
On DPX this value is always –1.

file_type

UFT_UNDEF_FILE (for all files exept GCOS 8 files)

UFT_GFRC_FILE (for GCOS 8 file)

UFT_UFAS_FILE (for GCOS 8 file)

record_format

UFT_FIXED

UFT_VARIABLE

inc_alloc_size incremental size

max_alloc_size maximum file size

cur_alloc_size current file size

alloc_size_unit define the unit for the size fields:

0: KBytes

1: number of records

creat_date creation date

last_ret_date last access date

last_mod_date last modification date

*return_code*

Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

## Return Code Values

UFT_OK: The function **UFTgetattribut()** is successful.

UFT_ERR: An error occurs from UFT protocol, *return_code* contains the description of the error.

UFT_COM_ERR:

An error occurs in the communication stack The *return_code* contains a communication error and the connection is in disconnected state.

UFT_NO_CONNECT:

There is no connection. A **UFTconnect()** must be issued first.

## State Changes

If no error occurs, the connection remains in connected state. If UFT_COM_ERR occured the connection state can be disconnected.

## Application Usage

The connection state must be connected.

## Related Information

See also the **readatt** subcommand, and the ”Error Messages”, on page A-1.

# UFTnoabort()

## Purpose

Set/unset the noabort mode.

## Syntax

**UFTnoabort** (*flag, return_code*)

int * *flag*

unsigned int * *return_code*

## Parameters

*flag*

The possible values for this flag are:

UFT_SET: To set the noabort mode.

UFT_UNSET: to disable the no abort mode.

The flag is returned to the user with the value UFT_SET or UFT_UNSET:

UFT_SET: The noabort mode is accepted.

UFT_UNSET: The noabort mode is refused because the remote host does not accept the **UFTrestart()** function.

*return_code*

Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

## Return Code Values

UFT_OK: The function **UFTnoabort()** is successful.

UFT_PARM_ERR:
A parameter is incorrect.

UFT_NO_CONNECT:
There is no connection, a **UFTconnect()** must be issued first.

## State Changes

Every transfer will be done in noabort mode (see ”Restarting Interrupted File Transfer”, on page 2-16), if the return flag is set to UFT_SET.

## Application Usage

The connection must be initialized.

## Related Information

See also the **noabort** subcommand, and the ”Error Messages”, on page A-1.

# UFTtrace()

## Purpose

Set the trace level and the trace file.

## Syntax

int **UFTtrace** (*trace_level, trace_file, return_code*)

unsigned int *trace_level*

char * *trace_file*

unsigned int * *return_code*

## Parameters

*trace_level*

This parameter gives the level of trace and can take the following values:

0: no trace

1: trace level 1

2: trace level 2

3: trace level 3

*trace_file*

The file in which the traces are recorded.

*return_code*

Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

## Return Code Values

UFT_OK: The function **UFTtrace()** is successful.

UFT_PARM_ERR:
A parameter is incorrect, bad file name or trace level.

## State Changes

The API is in traced state if the level is greater than 0.

## Application Usage

The UFT API must be in the initialized state.

## Related Information

See also the **trace** subcommand, and the ”Error Messages”, on page A-1.

# UFTcmp()

## Purpose

Set/unset the compress mode.

## Syntax

int **UFTcmp** (*flag, return_code*)

int * *flag*

unsigned int * *return_code*

## Parameters

*flag*

The possible values for this flag are:

UFT_SET: To set the compress mode.

UFT_UNSET: To disable the compress mode.

*return_code*

Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

## Return Code Values

UFT_OK: The function **UFTcmp()** is successful.

UFT_PARM_ERR: A parameter is incorrect, bad file name or trace level.

## State Changes

If the compress mode is set, every file transfer is done in compress mode, otherwise no compression is done.

## Application Usage

The UFT API must be in the initialized state.

## Related Information

See also the **cmp** subcommand, and the ”Error Messages”, on page A-1.

# UFTmodify()

## Purpose

Modify the parameter of the connection.

## Syntax

int **UFTmodify** (*max_record_size, transfer_mode, remote_machine_type, GCOS8_file_type, return_code*)

unsigned int * *max_record_size*
unsigned int *transfer_mode*
unsigned int *remote_machine_type*
char *GCOS8_file_type*
unsigned int * *return_code*

## Parameters

*max_record_size*

Maximum record size negotiated during the connection phase. (see ”Connecting to a remote Host”, on page 2-15, for the limits and the used of this parameter).

*transfer_mode*

The transfer mode can be UFT_MODE_LINE for line mode, or UFT_MODE_BLOC for block mode (see ”File Transfer Modes”, on page 1-4).

*machine_type*

This parameter defines the remote machine type:

DPX: for the DPX and DPX/20 range

DPS8: for the DPS8 range

DPS7: for the DPS7 range

DPS6: for DPS6 range

OTHER: for IBM range and other existing UFTs

*GCOS8_file_type*

This parameter is used only if the *machine_type* is DPS 8. It defines the type of GCOS8 file which is handled during transfers, file creation,... The possible values for this parameter are:

G8_GFRC_FILE

G8_UFAS_REL_FILE

G8_UFAS_SEQ_FILE

*return_code*

Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

## Return Code Values

UFT_OK: The function **UFTmodify()** is successful.

UFT_PARM_ERR:
A parameter is incorrect.

UFT_NO_CONNECT:
There is no connection. A **UFTconnect()** must be issued first.

## State Changes

No state change, only connection parameters are changed.

## Application Usage

The state must be connected before issuing this function.

## Related Information

See also the **modify** subcommand, and the ”Error Messages”, on page A-1.

# UFTrestart()

## Purpose

Restart an interrupted transfer.

## Syntax

int **UFTrestart** (*transfer_id, return_code*)

char * *transfer_id*

unsigned int * *return_code*

## Parameters

*transfer_id*

It is the identifier of the interrupted transfer.

*return_code*

Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

## Return Code Values

UFT_OK: The function **UFTrestart()** is successful.

UFT_ERR: An error occurs from the UFT protocol. *return_code* contains the description of the error.

UFT_COM_ERR:
An error occurs in the communication stack. The *return_code* contains a communication error and the connection is in disconnected state.

UFT_TRANS_REST:
The transfer can be restarted.

UFT_NO_CONNECT:
There is no connection. A **UFTconnect()** must be issued first.

## State Changes

If no error occurs, the connection is initialized to the remote host. The connection state is connected. If UFT_COM_ERR occurred the connection state may be disconnected.

## Application Usage

The connection state must be connected.

## Related Information

See also the **restart** subcommand, and the ”Error Messages”, on page A-1.

# UFTinterrupt()

## Purpose

This function can be called by a user signal handler to interrupt a transfer (**UFTsend()**, **UFTrec()** or **UFTrestart()**).

## Syntax

void **UFTinterrupt**()

## Description

During a file transfer, a signal handler can call this function to interrupt the file transfer. The information about the file transfer is logged and the file transfer can be restarted using **UFTrestart**().

If the interruption failed, the *return_code* of **UFTsend()** or **UFTrec()** contains the reason for the failure. If the transfer interruption is successful, **UFTsend()** or **UFTrec()** returns UFT_TRANS_REST and the file transfer can be restarted using the returned *transfer_id* (see **UFTsend()**, **UFTrec()**, **UFTrestart()**).

## State Changes

If no error occurs, the connection is initialized to the remote host. The connection state is connected. If UFT_COM_ERR occurred, the connection state may be disconnected.

## Related Information

See also the **CTRL C** subcommand, and the ”Error Messages”, on page A-1.

# UFTwho()

## Purpose

get the characteristics of the current connection from the local point of view.

## Syntax

int **UFTwho** (*uftwho*, *return_code*)

UFT_who **uftwho*

unsigned int *return_code*

## Parameters

*uftwho*

A structure UFT_who containing all the characteristics of the connection:

| | |
|---|---|
| connexion_en_cours | 1:connected |
| curdir | current directory of the local system |
| site | remote host to which the user is connected |
| locallogin | the local login |
| remotelogin | the remote login |
| remote | remote type machine (DPX,....) |
| maxlgrecs | maximun size of the transfer records |
| transfermode | the file transfer mode (UFT_MODE_LINE or UFT_MODE_BLOC) |
| uftcompac | compress mode or no |
| uftnoabort | noabort mode |
| trace_level | level trace |
| trace_file | name of the trace file |

*return_code*

Specifies the result of the call execution, which is returned to the local program. *return_code* can have one of the values explained below.

## Return Code Values

UFT_OK: The function **UFTwho()** is successful.

UFT_PARM_ERR:
A parameter is incorrect.

UFT_NO_CONNECT:
There is no connection. A **UFTconnect()** must be issued first.

## Application Usage

The connection must be active.

## Related Information

See also the **who** subcommand, and the ”Error Messages”, on page A-1.

# Appendix A. Error Messages

## Overview

This appendix deals with the error messages displayed during the Connection Phase and the Running Phase of UFT.

For more information about diagnostics, see ”UFT Diagnostic Guide”.

## Error Messages during Connection Phase

Error messages concerning faulty installation or operation may be displayed during the connection phase:

- **`Remote UFT server: inactive or saturated`**

The remote server is inactive or saturated.

- **`No such file or directory`**

The file `/etc/isohosts` does not exist or a special file is not available.

- **`remote host name not available`**

The site name does not exist in */etc/isohosts* or the parameters of this remote host description are incorrect.

- **`Session layer lock in progress (the Netls key for the Communication Stack is probably incorrect)`**

An iFor/LS key for the stack is incorrect, check your nodelock file.

- **`Problem with Communication Stack (Stack not loaded)`**

The stack must be loaded.

## Error Messages during Running Phase

If the command entered does not exist, UFT displays the message:

```
Unknown command....
```

If the syntax of the command is incorrect, UFT displays the message:

```
Syntax error
```

and gives the syntax of the command.

If there is an error during execution of the command, UFT displays an error message. The error report generally consists of three lines:

| | |
|---|---|
| first line: | command level error, |
| second line: | reason for the error, |
| third line: | if present, further information. |

Example:

```
Connect failed
Connection failed
Disconnection by session (84010080)
```

This error occurred during the execution of the **connect** subcommand, see page 2-6.

Return codes can appear in the second and third lines. A return code consists of 4 bytes structured as follows:

- **`origin`** 1 byte, where:
  - the leftmost bit signifies:

`0:` error on local system

`1:` error on remote host

  - the 7 rightmost bits identify the OSI layer that set the return code.
- **`severity`** 1 byte that can assume the following values:

`0:` normal

`1:` operating error

`2:` system error

- **`cause, diagnostic`** 2 bytes containing a detailed report.

Example:

```
Connect failed
Connection failed
Disconnection by session (84010080)
```

`84010080` is an FCB return code (3rd line), analyzed as follows:

84 error on the remote host set by level 4 (transport layer)

01 operating error

00 cause not specified by remote host

80 diagnostic matching *disconnection*

**Note:** Only return codes set by UFT are documented (OSI layer 6).

The most frequent session and transport return codes (OSI layers 5 and 4) are listed in the following pages.

For an explanation of unlisted return codes, use the tool **pmaderror** described in *OSI Diagnostic Interactive Toolkit (ODIT) User's Guide*.

# Return Codes

## Severity Code 0

| Return Code | Mnemonic | Description |
|---|---|---|
| 06000000: | normal: | no error |
| 06000001: | errtronq: | truncated data |
| 06000002: | errcrefich: | remote host refused to create file |
| 06000003: | errattfich: | remote host refused file attributes request |
| 06000004: | errassfich: | remote host refused to assign file |
| 06000005: | errouvfich: | remote host refused to open file |
| 06000006: | errdeassfich: | remote host refused to deassign file |
| 06000007: | errfermfich: | remote host refused to close file |
| 06000008: | errtransf: | remote host detected transfer error |
| 06000009: | errdetfich: | remote host refused to delete file |
| 0600000a: | errrefferm: | refused to close UFT connection |

## Severity Code 1

| Return Code | Mnemonic | Description |
|---|---|---|
| 06010010: | errdescout: | incorrect number of output descriptors |
| 06010011: | errdescin: | incorrect number of input descriptors |
| 06010012: | errnocan: | incorrect connection identification |
| 06010013: | errblk: | incorrect UCB header |
| 06010014: | errinfo: | the info field should be set to zero |
| 06010015: | errfcb: | incorrect FCB header |
| 06010016: | errlgfcb: | incorrect FCB size |
| 06010017: | errlgadtran: | incorrect transport addressing length |
| 06010018: | errrecs: | incorrect record size (exceeds 16384 byte) |
| 06010019: | errattrib: | incorrect attributes |
| 0601001a: | errdat: | incorrect date structure |

### Services not Supported on Connection

| Return Code | Mnemonic | Description |
|---|---|---|
| 06010020: | errservcr: | illegal creation service |
| 06010021: | errservsup: | illegal deletion service |
| 06010022: | errservatt: | illegal attribute request service |
| 06010023: | errservred: | illegal restart service |

### Check Pointing

| Return Code | Mnemonic | Description |
|---|---|---|
| 0601002e: | errnockm: | checkpoint option not negotiated |
| 0601002f: | errckmnum: | incorrect ckm number |

### Parameter Errors

| Return Code | Mnemonic | Description |
|---|---|---|
| 06010030: | errparam: | incorrect parameter in FCB |
| 06010031: | errdonnees: | no data during send (length=0) |
| 06010032: | errlgdata: | data length on `connect` or `disc` > 512 |
| 06010033: | errlgemiss: | length sent > record length |
| 06010034: | errlgrecept: | length of data received incorrect (length=0) |
| 06010035: | errfileid: | incorrect file identification |
| 06010036: | errlgfileid: | length of file_id > max size of file name (> 127) |
| 06010037: | errlgtr: | incorrect transfer identification length |
| 06010038: | errtrid: | incorrect transfer identification |
| 06010039: | errlgrest: | incorrect transfer restart identification length (>20 or =0) |
| 0601003a: | errrest: | incorrect restart identification |
| 0601003b: | erretfcb: | unknown FCB return code |

### Status Errors

| Return Code | Mnemonic | Description |
|---|---|---|
| 06010040: | errreqetat: | incompatible request with status |
| 06010041: | errreq: | illegal request or request in progress |
| 06010042: | errconnect: | connection refused by remote host |
| 06010043: | errferm: | disconnection in progress |
| 06010046: | errintr: | interrupted transfer |

### Abort Codes

| Return Code | Mnemonic | Description |
|---|---|---|
| 06010050: | errsession: | session abort |
| 06010051: | errprot: | abort on protocol error by remote host |
| 06010052: | erraband: | abort in progress |

## Severity Code 2

| Return Code | Mnemonic | Description |
|---|---|---|
| 06020060: | errctx: | no further UFT context |

## FCB Return Codes

| Return Code | Mnemonic | Description |
|---|---|---|
| 00000000: | erfcbnormal: | no error |
| 06010001: | erfcbmispar1: | missing parameter or incorrect value |
| 06010002: | erfcbinvpar1: | invalid parameter |
| 06010003: | erfcbincpar1: | unknown parameter |
| 06010004: | erfcbduppar1: | illegal duplication of parameter (only last value retained) |
| 06010005: | erfcbunkpar1: | unknown parameter (ignored) |
| 06010006: | erfcbseqpar1: | parameter out of sequence |
| 06010007: | erfcbsuppar1: | unsupported parameter value |
| 06010008: | erfcbaccperm: | insufficient access rights |
| 06010009: | erfcbnofile: | file does not exist |
| 0601000a: | erfcbfilebusy: | file is busy (locked) |
| 0601000b: | erfcbnodevice: | unknown unit |
| 0601000c: | erfcbsyntax: | incorrect file syntax |
| 0601000d: | erfcbioerror: | input/output error |
| 0601000e: | erfcbsystem: | system error |
| 0601000f: | erfcbunschar: | insufficient file characteristics |
| 06010010: | erfcbmemory: | no space available |
| 06010011: | erfcbnego: | negotiation error |
| 06010012: | erfcbcantopen: | cannot open file |
| 06010013: | erfcbcomp: | decompacting error |
| 06010014: | erfcbinvchar: | character not ASCII or EBCDIC |
| 06010015: | erfcbabort: | forced abort |
| 06010016: | erfcbnoack: | too many *synch* points not acknowledged |
| 06010017: | erfcbtranserr: | transmission error |
| 06010018: | erfcbspace: | file is full |
| 06010019: | erfcbrecordsize: | record size > max size |
| 0601001a: | erfcbnumrec: | invalid record number |
| 0601001b: | erfcbkey: | key error |
| 0601001c: | erfcbdupkey: | key duplication |
| 0601001d: | erfcbclose: | problem on closing file |
| 0601001e: | erfcbwexist: | file exists (warning) |
| 0601001f: | erfcbexist: | file exists (reject) |
| 06010020: | erfcbunsspace: | no space available on disk |
| 06010021: | erfcbunswchar: | insufficient characteristics |
| 06010022: | erfcbwfileinex: | file does not exist (warning) |
| 06010050: | erfcbshutdown: | abort by remote user |
| 06010051: | erfcbunsupmes: | unsupported protocol message |
| 06010052: | erfcbmessseq: | unexpected protocol message |
| 06010053: | erfcbmescont: | incorrect protocol message content |
| 06010054: | erfcbillegjet: | incorrect use of data token |
| 06010055: | erfcbmesssess: | incorrect use of session |
| 06010056: | erfcbtimeout: | inactivity timeout expired |
| 06010077: | erfcbintegrestart: | cannot assume data integrity in restart |

**Common Errors set by the Session Layer:**

| Return Code | Mnemonic | Description |
|---|---|---|
| 05600001: | indtronc: | truncated data |
| 0500003a: | errclo: | closure in progress |
| 05010013: | errrefclo: | remote host refused disconnection |
| 05010018: | errproftrp: | incorrect transport profile |
| 05010033: | erretat: | status incompatibility |
| 0501003b: | errab: | abort in progress |
| 0501003c: | errsync: | resynchronization in progress |
| 0501003d: | errdem: | identical request in progress |
| 0501003e: | errto: | timeout error |
| 05010043: | errrefus: | connection refused |
| 05010044: | errtrans: | transport closure |
| 05010066: | indclo: | session closure in process |
| 0502004b: | errgenses: | no more session contexts |
| 0502004e: | errtsapinc: | unknown TSAP |

| Return Code | Description |
|---|---|
| 85000381: | invalid remote session address |
| 05000382: | local user of service not connected to SSAP |
| 85000382: | remote user of service not connected to SSAP |
| 05000383: | spm local saturation |
| 85000383: | spm remote saturation |
| 05000384: | unknown protocol version (local) |
| 85000384: | unknown protocol version (remote) |
| 07000300: | no reason (local) |
| 87000300: | no reason (remote) |
| 07000301: | local rejection by user (temporary saturation) |
| 87000301: | remote rejection by user (temporary saturation) |
| 07000301: | local user rejection (reason in user data) |
| 87000302: | remote user rejection (reason in user data) |
| 07010000: | local user abort |
| 87010000: | remote user abort |
| 05020006: | local protocol error |
| 85020006: | remote protocol error |
| 05020003: | connection timeout |

**Common Errors set by the Transport Layer**

| Return Code | Description |
|---|---|
| 04010002: | session local entity not connected |
| 84010002: | session remote entity not connected |
| 04010003: | unknown local network address |
| 84010003: | unknown remote network address |
| 04010080: | disconnection requested by local session |
| 84010080: | disconnection requested by remote session |
| 04010085: | local protocol error |
| 84010085: | remote protocol error |

# UFT API Protocol Errors

| Return Code | Mnemonic | Description |
|---|---|---|
| 10010001: | erfcbmispar1: | missing parameter or incorrect value |
| 10010002: | erfcbinvpar1: | invalid parameter |
| 10010003: | erfcbincpar1: | unknown parameter |
| 10010004: | erfcbduppar1: | illegal duplication of parameter (only last value retained) |
| 10010005: | erfcbunkpar1: | unknown parameter (ignored) |
| 10010006: | erfcbseqpar1: | parameter out of sequence |
| 10010007: | erfcbsuppar1: | unsupported parameter value |
| 10010008: | erfcbaccperm: | unsufficient access rights |
| 10010009: | erfcbnofile: | file does not exist |
| 1001000a: | erfcbfilebusy: | file is busy (locked) |
| 1001000b: | erfcbnodevice: | unknown unit |
| 1001000c: | erfcbsyntax: | incorrect file syntax |
| 1001000d: | erfcbioerror: | input/output error |
| 1001000e: | erfcbsystem: | system error |
| 1001000f: | erfcbunschar: | insufficient file characteristics |
| 10010010: | erfcbmemory: | no space available |
| 10010011: | erfcbnego: | negotiation error |
| 10010012: | erfcbcantopen: | cannot open file |
| 10010013: | erfcbcomp: | decompacting error |
| 10010014: | erfcbinvchar: | character not ASCII or EBCDIC |
| 10010015: | erfcbabort: | forced abort |
| 10010016: | erfcbnoack: | too many *synch* points not acknowledged |
| 10010017: | erfcbtranserr: | transmission error |
| 10010018: | erfcbspace: | file is full |
| 10010019: | erfcbrecordsize: | record size > max size |
| 1001001a: | erfcbnumrec: | invalid record number |
| 1001001b: | erfcbkey: | key error |
| 1001001c: | erfcbdupkey: | key duplication |
| 1001001d: | erfcbclose: | problem on closing file |
| 1001001e: | erfcbwexist: | file exists (warning) |
| 1001001f: | erfcbexist: | file exists (reject) |
| 10010020: | erfcbunsspace: | no space available on disk |
| 10010021: | erfcbunswchar: | insufficient characteristics |
| 10010022: | erfcbwfileinex: | file does not exist (warning) |
| 10010050: | erfcbshutdown: | abort by remote user |
| 10010051: | erfcbunsupmes: | unsupported protocol message |
| 10010052: | erfcbmessseq: | unexpected protocol message |
| 10010053: | erfcbmescont: | incorrect protocol message content |
| 10010054: | erfcbillegjet: | incorrect use of data token |
| 10010055: | erfcbmesssess: | incorrect use of session |

| Return Code | Mnemonic | Description |
|---|---|---|
| 10010056: | erfcbtimeout: | inactivity timeout expired |
| 10010077: | erfcbintegrestart: | cannot assume data integrity in restart |
| 10010078: | erfcbuserrestart: | bad remote site or bad connection on restart |
| 10010080: | err_notfound: | host name not found in */etc/isohosts* |
| 10010081: | err_format: | format error in */etc/isohosts* |
| 10010082: | err_param: | parameter entry error in */etc/isohosts* |
| 10010083: | err_tronq: | incomplete entry in */etc/isohosts* |

# Appendix B. File Identifiers on Non DPX Systems

## Overview

The following sections give the syntax of file identifiers:



## Syntax of File Identifiers on the DPS 7

### Members of Source or Binary Libraries

#### Uncataloged Library

**Syntax:**

```
library_name..member_name:media:device
```

**where:**

| | |
|---|---|
| `library_name` | designates the library name |
| `member_name` | designates the member of the library |
| `media` | designates the name of the medium on which the library is stored |
| `device` | designates the type of medium on which the library is stored. |

**Example:**

```
D7LIB..FICH:L736:MS/D500
```

with:

| | |
|---|---|
| D7LIB: | library |
| FICH: | member of D7LIB |
| L736: | name of medium |
| MS/D500: | type of medium |

#### Cataloged Libraries:

**Syntax:**

```
library_name..member_name
```

**where:**

| | |
|---|---|
| `library_name:` | designates the name of the library |
| `member_name:` | designates the member of the library |

**Example:**

D7LIB**..**FICH

with:

| | |
|---|---|
| D7LIB: | library |
| FICH: | member of D7LIB |

## UFAS Sequential Files

### Uncataloged Files:

**Syntax:**

```
file_name:media:device
```

**where:**

| | |
|---|---|
| `file_name` | designates the name of the file |
| `media` | designates the medium on which the file is stored |
| `device` | designates the type of medium on which the file is stored |

**Example:**

D7FICH**:**LAB1**:**MS/D500

with:

| | |
|---|---|
| D7FICH: | file |
| LAB1: | medium name |
| MS/D500: | type of medium (D500 disk) |

### Cataloged Files:

**Syntax:**

```
file_name
```

**where:**

| | |
|---|---|
| `file_name` | designates the file name |

**Example:**

D7FICH

with:

| | |
|---|---|
| D7FICH: | file |

# Syntax of File Identifiers on the DPS 8

## For a disk file:

**Syntax:**

```
UMC_name/UFT_catalog_name/file_name
```

**where:**

| | |
|---|---|
| `UMC_name` | designates the name of the UMC associated with the login on the DPS 8 |
| `UFT_cat_name` | designates the name of the catalog including the UFT access rights to files |
| `file_name` | designates the name of the file in the UFT catalog. This file inherits the rights of the UFT catalog, and can therefore be handled by UFT. |

**Example:**

74502/UFT/FICH

with:

| | |
|---|---|
| 74502: | UMC on DPS 8 |
| UFT: | catalog with required access rights |
| FICH: | file that can be used by UFT |

# Syntax of File Identifiers on the DPS 6

## For a disk file:

```
^disk_name>dir_comp1>dir_comp2>...>file_name
```

**where**

| | |
|---|---|
| `disk_name` | designates the name of the disk on which the file resides (**>>** for default disk) |
| `dir_comp1` | designates the component of the directory path |
| `file_name` | the target file name |

**Example:**

^B41U21D>UDD>UFTX>UFT_TEST

with:

| | |
|---|---|
| B41U21D: | disk name |
| UDD UFTX: | components of the path |
| UFT_TEST: | name of the file |

# Syntax of File Identifiers on IBM Systems

For IBM systems, the dataset name must be specified.

The dataset name is up to 44 alphanumeric characters, which completely qualify the cataloged dataset or cluster.

**Example:**

```
GROUP1.FILE2
PROJECT(JOBA)
```

# Appendix C. File Management on the DPS 7

## Overview

The following sections describe how to:

- Create Libraries and Files, on page C-1.
- Display Libraries and Files, on page C-3.

## Creating Libraries and Files

The file creation facility of UFT is not supported by release GCOS7 V5 or earlier releases on the DPS 7. One solution is to use a terminal connected to IOF to create the files on DPS 7 before sending them by UFT. Some products such as PAD, TWS2107, OTM provide a remote terminal function. In release V6 of GCOS 7 this problem has been solved.

The creation of members of an existing library is implicitly performed during the put/send command in all GCOS 7 releases.

### BUILD_LIBRARY Command

### Syntax

```
BLIB LIB=library_name:media:device
     FILESTAT=UNCAT
     SIZE=number_of_cylinders
     MEMBERS=number_of_members
     TYPE=library_type;
```

### Description

Creates a Source or Binary Library.

### Flags

| | |
|---|---|
| *library_name* | identifies the library |
| *media* | identifies the medium on which the library is stored |
| *device* | identifies the device running the medium |
| UNCAT | constant value |
| *number_of_cylinders* | defines the size of the library expressed in disk cylinders |
| *library_type* | identifies the type library source (SL) or binary (BIN) |

### Example

```
BLIB LIB=D7LIBSL:L738:MS/D500
     FILESTAT=UNCAT
     SIZE=2
     MEMBERS=10
     TYPE=SL
```

Characteristics of the library created:

| | |
|---|---|
| name | D7LIBSL |
| status | uncataloged (UNCAT) |
| initial_size | 2 x D500 disk cylinders |
| max_number_of_members | 10 |
| type | source library (SL) |
| medium | cartridge disk (MS/D500) |
| medium_name | L738 |

# BUILD_FILE Command

## Syntax

```
BF FILE=file_name:media:device
   FILESTAT=UNCAT
   UFAS=SEQ
   UNIT=CI
   SIZE=number_of_CI
   CISIZE=size_of_CI
   RECSIZE=record_size
   RECFORM=V
```

## Description

Creates a UFAS Sequential Variable File

## Flags

| | |
|---|---|
| *file_name* | identifies the file |
| *media* | identifies the medium on which the file is stored |
| *device* | identifies the device running the medium |
| *number_of_CI* | defines the number of Control Intervals allocated to the file |
| *size_of_CI* | defines the size in bytes of a CI (unit of allocation) |
| *record_size* | defines the size in bytes of a file record |

## Example

```
BF FILE=D7FICSEQ:L686:MS/D500
   FILESTAT=UNCAT
   UFAS=SEQ
   UNIT=CI
   SIZE=20
   CISIZE=2048
   RECSIZE=1024
   RECFORM=V
```

Characteristics of the File created:

| | |
|---|---|
| name | D7FICSEQ |
| status | uncataloged (UNCAT) |
| initial_size | 20 CIs (CI is 2048 bytes in this case) |
| record_size | 1024 bytes (maximum) |
| record_format | V (variable) |
| medium | fixed disk (MS/D500) |
| medium_name | L686 |

# Displaying Libraries and Files

## LIST_FILE Command

### Syntax

```
LSF {library|file_name} ALL;
```

### Description

Displays library and file Characteristics:

### Flags

*library* or *file_name* identifies the library or file

### Example

```
LSF STOCK:L738:MS/D500 ALL;
```

For printing, displays the characteristics of the uncataloged library or file:

```
STOCK:L738:MS/D500
```

## PRINT_FILE Command

### Syntax

```
PRF file_name FORMAT=ALPHA;
```

### Description

Displays the contents of a file

### Flags

*file_name* identifies the file

### Example

```
PRF D7FIC:LAB3:MS/D500 FORMAT=ALPHA;
```

Displays the contents of the uncataloged file `D7FIC:LAB3:MS/D500` for printing in alphanumeric mode.

## MAINTAIN_LIBRARY Command

### Syntax

```
MNLIB library_type library_name;
```

### Description

Displays the contents of a library.

## Flags

| | |
|---|---|
| *library_type* | identifies the type of the library, source (SL) or binary (BIN) |
| *library_name* | identifies the library |

## Example

```
MNLIB SL D7LIBSL:L738:MS/D500
```

This starts the library management processor with the uncataloged `D7LIBSL:L738:MS/D500` as the name of the current library.

MNLIB command changes the library and allows its members to be displayed.

## Subcommands

The following printing commands can be used with the MAINTAIN_LIBRARY command.

- to print the list of library members:

  ```
  LIST *
  ```

- to print the contents of a library member:

  ```
  PRINT member_name [HEXA]
  ```

  where

| | |
|---|---|
| *member_name* | Identifies the member of the current library. |
| HEXA | Option for printing the content of the file in hexadecimal. By default, the content of the file is printed in alphanumeric mode. |

## Example

```
PRINT FICH;
```

Prints the content of the file FICH in alphanumeric mode.

# Appendix D. Creating Files on the DPS 8

## Recommendations for Creating Files with UFT

The size of files created on the DPS 8 is expressed in *little links* (1 LL = 1280 characters).

The UFT application for the DPS 8 converts the size chosen by a remote UFT user from Kbytes to LL.

The GFRC sequential organization stores control records and data records:

- when the file is created
- when the file is filled.

Since the unit of the DPS 8 file is the 4–byte word, all records are multiples of 4 bytes. The last word of a record is padded to make up 4 bytes if it is less than 4 bytes long.

On **create**, DPS 8 files are allocated with *unlimited* size. During the transfer, the size proportionally expands as more space is required.

However, for an already existing remote file, its current size must be sufficient to receive the total contents of the local file.

# Appendix E. System Limitations

## Transferring NFS Files with UFT

To send or receive an NFS file, the lock manager daemons `/usr/bin/rpc/lockd` and `/usr/bin/rpc/statd` must both be running.

The daemons lock the file during its transfer, thereby enabling UFT to preserve the integrity of the file contents.

See *DPX/20 Commands Reference Manual* (6 Volumes) for details on the lock manager.

**The uft or uftser process may block if the user attempts to transfer an NFS file when the above daemons are not running**.

## Reverse Charging Facility

It is possible to accept or refuse the reverse charging facility at the X25 level. It is not possible to do this at the UFT level. Therefore, it is not possible to refuse UFT connection requests with reverse charging when X25 accepts the facility.

## Simultaneous Connections

The number of simultaneous incoming and outgoing connections is not limited by UFT. It is, however limited by the maximum number of available connections of the communications stack and the memory available.

## File Size

The size of a file to transfer must agree with the value of ULIMIT set in the system. For file sizes greater than ULIMIT, the value of ULIMIT must be increased.

# Appendix F. API Programming Example

The sources of this program are found in */usr/lpp/uft/samples.*

Compilation command: `cc -lUFTapi -lPW program.c -o program`

```
/*
 * @BULL_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: caller.c,v $
 * Revision 1.2  1996/01/24  10:40:28  uft
 *
 * $EndLog$
 */
#include <stdio.h>
#include <signal.h>
#include <fcntl.h>
#include "UFTapi.h"

void    DOUFTinit();
void    DOUFTconnect();
void    DOUFTdisconnect();
void    DOUFTsend();
void    DOUFTrec();
void    DOUFTdelete();
void    DOUFTcreate();
void    DOUFTgetattribut();
void    DOUFTnoabort();
void    DOUFTtrace();
void    DOUFTcmp();
void    DOUFTmodify();
void    DOUFTinterrupt();
void    DOUFTrestart();
void    DOUFTinitcon();
void    DOUFTh();
void    DOUFTh1();
void    DOUFTprint();
void    DOUFTsprint();
void    DOUFTwho();

struct  scall
{
        char    *name;
        void     (*funct)();
}       syscalls[] =
{
        "init"          ,DOUFTinit,
        "connect"     ,DOUFTconnect,
        "disconnect"     ,DOUFTdisconnect,
        "send"          ,DOUFTsend,
        "rec"           ,DOUFTrec,
        "delete"     ,DOUFTdelete,
        "create"     ,DOUFTcreate,
        "getattribut"     ,DOUFTgetattribut,
        "noabort"     ,DOUFTnoabort,
        "trace"         ,DOUFTtrace,
        "cmp"           ,DOUFTcmp,
        "modify"     ,DOUFTmodify,
        "restart"     ,DOUFTrestart,
        "initcon"     ,DOUFTinitcon,
```

```
        "interrupt"     ,DOUFTinterrupt,
        "tracoff"     ,DOUFTh,
        "traceon"     ,DOUFTh1,
        "print"         ,DOUFTprint,
        "psrint"     ,DOUFTsprint,
        "who"           ,DOUFTwho,
        ""      ,(void (*)())0
}

static  char    tbfin[] = "fin";

int        level_trace = 0 ;
#define PRINTF(A)\
{\
     if ( level_trace == 0 )\
        {\
                    printf(A);\
        }\
}

void perreur( char * s , unsigned int crd)
{
        printf("\n");
     if ( crd > 256 )
     {
        printf("erreur dans %s  %x \n", s , crd);
        return;
     }
     if ( (crd & UFT_ALREADY_CON) == UFT_ALREADY_CON)
     {
        printf("erreur dans %s   UFT_ALREADY_CON \n",s);
     }
     if ( (crd & UFT_ERR) == UFT_ERR )
     {
         printf("erreur dans %s  UFT_ERR \n",s);
     }
     if ( (crd & UFT_PARM_ERR) == UFT_PARM_ERR )
     {
        printf("erreur dans %s  UFT_PARM_ERR \n",s);
     }
     if ( (crd & UFT_NO_CONNECT) == UFT_NO_CONNECT )
     {
         printf("erreur dans %s  UFT_NO_CONNECT \n",s);
     }
     if ( (crd & UFT_COM_ERR) == UFT_COM_ERR )
     {
        printf("erreur dans %s   UFT_COM_ERR \n",s);
     }
     if ( (crd & UFT_INIT_PROBLEM) == UFT_INIT_PROBLEM )
     {
        printf("erreur dans %s   UFT_INIT_PROBLEM \n",s);
     }
     if ( (crd & UFT_TRANS_REST) == UFT_TRANS_REST )
     {
        printf("erreur dans %s   UFT_TRANS_REST \n",s);
     }

     printf("erreur dans %s  %x \n", s , crd);
}


void main()
{
char    syscall[10];
```

```
struct  scall   *sc;
int             index ;
        printf("For inputs,     (s) means string,\n");
        printf("                (d) means decimal,\n");
        printf("                (x) means hexadecimal,\n");
        printf("                (o) means octal\n");

        for ( printf("\nsyscall ? "); scanf("%s", syscall) != EOF
; printf("\nsyscall ? ") )
     {
                for (sc = syscalls; sc->funct && strcmp(sc->name,
syscall); sc++);
                if (sc->funct)
                        (*sc->funct)();
                else {
                        if ( strcmp(syscall,tbfin) == 0 )
                                break ;
                        printf("syscall:\n");
                        for (sc = syscalls,index=0; sc->funct;
sc++,index++)
                        {
                                                printf("%s ",
sc->name);
                                                if (index == 10)
                                                {
                                                        index = 0
;

printf("\n");
                                                }
                        }
                }
        }
}

void      DOUFTprint()
{
     int        num_scenario ;
     int        recall;

     recall = scanf("%d",&num_scenario);
     printf("\n Scenario = %d \n",num_scenario);
}

void     DOUFTsprint()
{
        char    num_scenario[128] ;
        int     recall;

        recall = scanf("%s",&num_scenario);
        printf("\n Scenario = %s \n",num_scenario);
}

void      DOUFTh()
{
     level_trace = 1 ;
}
void      DOUFTh1()
{
     level_trace = 0 ;
}

void     DOUFTinit()
{
```

```
     unsigned int       return_code = 9999 ;
     int        crd = 9999 ;

     crd = UFTinit (&return_code) ;

     if ( return_code != 0 )
     {
        perreur (”UFTinit\treturn_code =”,return_code);
     }
     if ( crd != 0 )
     {
        perreur (”UFTinit\tcrd =”, crd );
     }
}

void     DOUFTcmp()
{
     unsigned     return_code = 9999 ;
     int     crd = 9999 ;
     int        recall ;
     int        flag_comp ;

     PRINTF(”Compress mode 1(set) 2(unset)?”);

     recall = scanf(”%d”,&flag_comp);

     crd = UFTcmp(&flag_comp, &return_code) ;
     if ( return_code != 0 )
     {
        perreur (”UFTcmp\treturn_code =”,return_code);
     }
     if ( crd != 0 )
     {
        perreur (”UFTcmp\tcrd =”,crd);
     }
}
void    DOUFTsend()
{
     unsigned   return_code = 9999 ;
     int        crd = 9999 ;
     int        recall ;
     char       nom_local[256];
     char       nom_distant[256];
     int        type_trans ;
     char       *transfer_id ;

     PRINTF(”nom local file :? ”);
     recall = scanf(”%s”,nom_local);
     PRINTF(”nom distant file :? ”);
     recall = scanf(”%s”,nom_distant);
     PRINTF(”Type transfert 0 (UFT_AS) 1(UFT_AP)?:”);
     recall = scanf(”%d”,&type_trans);
     if ( type_trans == 0 )
     {
        type_trans = UFT_AS ;
     }else if ( type_trans == 1 )
     {
        type_trans = UFT_AP ;
     }
     crd = UFTsend( nom_local , type_trans , nom_distant
,&return_code , &transfer_id);
     if ( crd != 0 )
     {
```

```
         perreur (”UFTsend\tcrd =”,crd);
      }
      if ( return_code != 0 )
      {
         perreur (”UFTsend\treturn_code =”,return_code);
      }
      if ( (crd&UFT_TRANS_REST) == UFT_TRANS_REST )
      {
         printf(”Transfert Id = %s \n”,transfer_id);
      }
}
void    DOUFTrec()
{
      unsigned   return_code = 9999 ;
      int        crd = 9999 ;
      int        recall ;
      char    nom_local[256];
      char    nom_distant[256];
      char       *transfer_id ;
      int     type_trans ;

       PRINTF(”nom distant file :? ”);
      recall = scanf(”%s”,nom_distant);
      PRINTF(”nom local file :? ”);
      recall = scanf(”%s”,nom_local);
      PRINTF(”Type transfert 0 (UFT_AS) 1 (UFT_AP)?:”);
      recall = scanf(”%d”,&type_trans);
      if ( type_trans == 0 )
      {
         type_trans = UFT_AS ;
      }else if ( type_trans == 1 )
      {
         type_trans = UFT_AP ;
      }
      crd = UFTrec( nom_distant , type_trans , nom_local
,&return_code , &transfer_id);
      if ( crd != 0 )
      {
         perreur (”UFTrec\tcrd =”,crd);
      }
      if ( return_code != 0 )
      {
         perreur (”UFTrec\treturn_code =”,return_code);
      }
      if ( (crd&UFT_TRANS_REST) == UFT_TRANS_REST )
      {
         printf(”Transfert Id = %s ”,transfer_id);
      }
}
void     DOUFTgetattribut()
{
      unsigned   return_code = 9999 ;
      int     crd = 9999 ;
      int     recall ;
      char  nom_file[256] = ”” ;
      UFTattrib attributs ;


      PRINTF(”nom file :?”);
      recall = scanf(”%s”,nom_file);
      crd =  UFTgetattribut ( nom_file , &attributs,
&return_code);
      if ( crd != 0 )
      {
```

```
        perreur (”UFTgetattribut\tcrd =”,crd);
     }
     if ( return_code != 0 )
     {
        perreur (”UFTgetattribut\treturn_code =”,return_code);
     }
}

void     DOUFTdelete()
{
     unsigned return_code = 9999 ;
     int        crd = 9999 ;
     int         recall ;
     char       nom_file[256] = ”” ;

     PRINTF(”nom fichier a deleter:?”);
     recall = scanf(”%s”,nom_file);
     crd = UFTdelete( nom_file , &return_code);
     if ( crd != 0 )
     {
        perreur (”UFTdelete\tcrd=”,crd);
     }
     if ( return_code != 0 )
     {
        perreur (”UFTdelete\treturn_code =”,return_code);
     }
}

void     DOUFTcreate()
{
     unsigned return_code = 9999 ;
     int        crd = 9999 ;
     int        recall ;
     int        file_size ;
     char  nom_file[256] = ”” ;

     PRINTF(”nom file a creer :?”);
     recall = scanf(”%s”,&nom_file);
     PRINTF(”file size :?”);
     recall = scanf(”%d”,&file_size);
     crd = UFTcreate(nom_file,file_size,&return_code);
     if ( crd != 0 )
     {
        perreur (”UFTcreate\tcrd=”,crd);
     }
     if ( return_code != 0 )
     {
        perreur (”UFTcreate\treturn_code =”,return_code);
     }
}

void     DOUFTconnect()
{
     unsigned return_code = 9999 ;
     int        crd = 9999 ;
     int        recall ;
     char  nom_site[256] = ”” ;
     char  remote_log[256] = ””;
     char  remote_pass[256]  = ””;
     char  remote_bil[256] = ””;
     char  remote_proj[256] = ””;
     unsigned max_record_size = 1 ;
     unsigned remote_type_machine =  1 ;
     char GCOS8_file_type ;
```

```
     unsigned int transfer_mode ;


     PRINTF(”nom Site :?”);
     recall = scanf(”%s”,&nom_site );
     PRINTF(”login distant :?”);
     recall = scanf(”%s”,&remote_log);
     PRINTF(”passwd distant :?”);
     recall = scanf(”%s”, &remote_pass);
     PRINTF(”Max record size ( 1->512  2->950  3->2000  ):”);
     recall = scanf(”%d”,&max_record_size);
     if ( max_record_size == 1 )
     {
        max_record_size = 512 ;
     }
     if ( max_record_size == 2 )
     {
        max_record_size = 950 ;
     }
     if ( max_record_size == 3 )
     {
         max_record_size = 2000 ;
     }
     PRINTF(”Mode transfert:? 1(Mode block)2( Mode line)  ?”);
     recall = scanf(”%d”,&transfer_mode);
        if ( transfer_mode == 1 )
        {
                transfer_mode = UFT_MODE_BLOC ;
        }else if(transfer_mode == 2)
                {
                        transfer_mode = UFT_MODE_LINE ;
                }

      PRINTF(”1->DPX  2->DPS8  3->DPS7 4->DPS6 5->OTHER  ”);
     recall = scanf(”%d”,&remote_type_machine);
     PRINTF(” type de fichiers GCOS8 (1 GFRC) ( 2 UFF REL)   (3
UFF SEQ) :?”);
     recall = scanf(”%d”,&GCOS8_file_type);

     crd =
UFTconnect(NULL,0,nom_site,remote_log,remote_pass,remote_bil,remo
te_proj,&max_record_size,transfer_mode,remote_type_machine,GCOS8_
file_type,&return_code);

     if ( crd != 0 )
     {
        perreur (”UFTconnect\tcrd =”,crd);
     }
     if ( return_code != 0 )
     {
        perreur (”UFTconnect\treturn_code =”,return_code);
     }
}

void     DOUFTdisconnect()
{
     unsigned int return_code ;
     unsigned int crd ;

     crd = UFTdisconnect(&return_code) ;

     if ( return_code != 0 )
     {
        perreur (”UFTdisconnect\treturn_code =”,return_code);
```

```
      }
      if ( crd != 0 )
      {
         perreur (”UFTdisconnect\tcrd =”,crd);
      }
}
void     DOUFTnoabort()
{
      unsigned int return_code ;
      unsigned int crd ;
      int        recall ;
      int flag = 1;

      PRINTF(”noabort SET(1) UNSET(2):?”);
      recall = scanf(”%d”,&flag);

      crd = UFTnoabort(&flag, &return_code) ;

      if ( return_code != 0 )
      {
         perreur (”UFTnobaort\treturn_code =”,return_code);
      }
      if ( crd != 0 )
      {
         perreur (”UFTnobaort\tcrd =”,crd);
      }

}
void DOUFTmodify()
{
      unsigned int return_code ;
      unsigned int crd ;
      int recall ;
      unsigned int max_record_size = 1 ;
      char GCOS8_file_type ;
      unsigned int transfer_mode ;
      unsigned int remote_type_machine ;

      PRINTF(”Max record size ( 1->512  2->9503->2000  ):”);
      recall = scanf(”%d”,&max_record_size);
      if ( max_record_size == 1 )
      {
         max_record_size = 512 ;
      }
      if ( max_record_size == 2 )
      {
         max_record_size = 950 ;
      }
      if ( max_record_size == 3 )
      {
         max_record_size = 2000 ;
      }
      PRINTF(”Mode transfert:? 1(Mode block) 2(Mode line)”);
      recall = scanf(”%d”,&transfer_mode);

      if ( transfer_mode == 1 )
      {
         transfer_mode = UFT_MODE_BLOC ;
      }else if(transfer_mode == 2)
         {
                transfer_mode = UFT_MODE_LINE ;
         }

      PRINTF(”1->DPX  2->DPS8  3->DPS7 4->DPS6 ”);
```

```
     recall = scanf("%d",&remote_type_machine);

     PRINTF(" type de fichiers GCOS8 (1 GFRC) ( 2 UFF REL) (3 UFF
SEQ) :?");
     recall = scanf("%d",&GCOS8_file_type);
     crd = UFTmodify(
&max_record_size,transfer_mode,remote_type_machine,GCOS8_file_typ
e,&return_code);
     if ( return_code != 0 )
     {
        perreur ("UFTmodify\treturn_code =",return_code);
     }
     if ( crd != 0 )
     {
        perreur ("UFTmodify\tcrd =",crd);
     }
}
void     DOUFTrestart()
{
     unsigned return_code = 9999 ;
     int     crd = 9999 ;
     int        recall ;
     char transfer_id[256] = "" ;

     PRINTF("Transfer_id :?");
     recall = scanf("%s",transfer_id);

     crd = UFTrestart(transfer_id,&return_code);

     if ( return_code != 0 )
     {
        perreur ("UFTrestart\treturn_code =",return_code);
     }
     if ( crd != 0 )
     {
        perreur ("UFTrestart\tcrd =",crd);
     }
}

void    DOUFTinterrupt()
{
     sigset (SIGINT, UFTinterrupt);
     test_who();
}
void     DOUFTtrace()
{
     unsigned return_code = 9999 ;
     unsigned crd = 9999 ;
     unsigned trace_level ;
     char trace_file[256];


     PRINTF("trace level  0 a 3) :?");
     scanf("%d",&trace_level);
     PRINTF("trace file :?");
     scanf("%s",trace_file);
     crd = UFTtrace(trace_level,trace_file,&return_code);
     if ( return_code != 0 )
     {
        perreur ("UFTtrace\treturn_code =",return_code);
     }
     if ( crd != 0 )
     {
        perreur ("UFTtrace\tcrd =",crd);
```

```
     }
}

/*
extern  int     UFTconnect(struct TABSIT *,unsigned int,char
*,char *,
                char *,char *,char *,unsigned int *,unsigned int,
                unsigned int,char ,unsigned int *);
*/

void    DOUFTinitcon()
{
     unsigned return_code = 9999 ;
     int     crd = 9999 ;
     int     recall ;
     char  nom_site[256] = "" ;
        char  remote_log[256] = "root";
        char  remote_pass[256]  = "twitwi";
        char  remote_bil[256] = "";
        char  remote_proj[256] = "";
        unsigned max_record_size =512 ;
        unsigned remote_type_machine =  1 ;
        char GCOS8_file_type ;
        unsigned int transfer_mode ;
     int uftactamd ;
     char table[400];
     struct TABSIT table_site ;
     int i,j;

     PRINTF("nom Site :?");
        recall = scanf("%s",&nom_site );

     crd = find(nom_site,300,table,"ufts:",(char *)&uftactamd);

     dump_mem(table,crd);

     crd = UFTconnect((struct TABSIT
*)table,crd,NULL,remote_log,remote_pass,remote_bil,remote_proj,&m
ax_record_size,transfer_mode,remote_type_machine,GCOS8_file_type,
&return_code);
     if ( return_code != 0 )
     {
        perreur ("UFTconnect\treturn_code =",return_code);
     }
     if ( crd != 0 )
     {
        perreur ("UFTconnect\tcrd =",crd);
     }
}
int dump_mem(buffer,lg)
char * buffer;
unsigned int lg;
{
     int i ,j ;
     char s[30];

     printf("\n");
     for (j=0,i=0;i<lg;i++)
     {
        if(i==0)
        {
                printf("%02x",buffer[i]);
                if(isprint(buffer[i])==0)
                {
```

```
                        s[0]='.';
                }else{
                        s[0]=buffer[i];
                }
                continue;
        }
        if(i%4==0)
        {
                printf(" ");
        }
        if (i%24==0)
        {
                printf(" %s\n",s);
        }
        printf("%02x",buffer[i]);
        if(isprint(buffer[i])==0)
        {
                s[i%24]='.';
        }else{
                s[i%24]=buffer[i];
        }
     }
     printf(" %s\n",s);
     return(0);
}
void DOUFTwho()
{
        UFT_who uftwho;
        unsigned int return_code ;
        int crd ;

        crd = UFTwho(&uftwho,&return_code);

        if ( return_code != 0 )
        {
                perreur ("UFTconnect\treturn_code
=",return_code);
        }
        if ( crd != 0 )
        {
                perreur ("UFTconnect\tcrd =",crd);
        }
        printf("You are connected to site \t: %s\n",uftwho.site
);
        printf("Local login  \t\t\t: %-12s\n",uftwho.locallogin);
        printf("Remote login  \t\t\t:
%-12s\n",uftwho.remotelogin);
        printf("Remote machine type\t\t: ");
        switch(uftwho.remote)
        {
                case DPX    :   puts ("DPX")   ;        break ;
                case DPS8   :   puts ("DPS8")  ;        break ;
                case DPS7       :       puts ("DPS7")  ;
break ;
                case DPS6       :       puts ("DPS6")  ;
break ;
                case OTHER      :       puts ("Foreign Host") ;
break ;

                default         :       puts ("Unknown") ;
break ;

        }
        printf ("Max record size   \t\t: %d \n",
```

```
uftwho.maxlgrecs);
        if(uftwho.transfermode == UFT_MODE_BLOC)
        {
                printf("Transfer mode \t\t\t: block \n");
     }
     if(uftwho.transfermode == UFT_MODE_LINE)
     {
                printf("Transfer mode \t\t\t: line\n") ;
        }
        printf ("Compression mode\t\t: %s\n",(uftwho.uftcompac ==
UFT_SET)?"on":"off")
;
        printf ("No abort mode\t\t\t: %s\n",(uftwho.uftnoabort ==
UFT_SET)?"on":"off");
        printf ("Trace level\t\t\t: %d\n",uftwho.trace_level);
        printf ("Trace file\t\t\t: %s\n",uftwho.trace_file);
        printf("\nThe current local directory is\t:
%s\n",uftwho.curdir);
```

# Appendix G. Remote Hosts Table

To connect with a remote host, specify the *remote host*. For example, `SYSTEM001` in **connect**.

UFT checks if this host name exists in the file */etc/isohosts* and recovers the information necessary to connect to it.

The command /**usr**/**lpp**/**uft**/**samples**/**uftact** is used to create and to subsequently update the remote hosts access file */etc/isohosts*. A conventional text editor can also be used to modify the */etc/isohosts* file.

## Structure of the Remote Hosts Table

The */etc/isohosts* system table incorporates the remote host access descriptions.

The syntax of the description is as follows:

```
SITE-NAME service names
class of transport (0, 2, 4):
alternate class (0, 2, 4):
expedited data (0, 1):
flow control (0, 1):
credit (1 to 15):
checksum control for class 4 (0, 1):
tpdu size:
nsdu size:
npdu size:
input network window (1 to 127):
output network window (1 to 127):
type of network (1-Transpac, 2-Ethernet):
type of connection (0 SVC, 1 PVC):
local address:
remote address:
length of network facilities (1 to 10):
network facilities (list of digits):
length of local TSAP (0 to 16):
local TSAP (list of digits):
length of remote TSAP (0 to 16):
remote TSAP (list of digits):
network calling data (list of digits):
```

The */etc/isohosts* table has been defined for a range of applications such as UFT and TPAD–HPAD. Consequently, some fields of the description may not be relevant to UFT.

### Syntax Rules

All lines of the sites table, except the first, have the following syntax:

```
comment:values
```

- `comment` is not analyzed by **connect**,
- the colon (:) is mandatory,
- `values` can be a number (decimal or hexadecimal) or a list of numbers separated by commas.

Examples of possible values:

```
12                  (decimal value 12)
0xfc                (hexadecimal value fc)
12,13,0xa2,0x2,3    (decimal and hexadecimal values)
```

- the lines between two site description tables are ignored by **connect**
- the description of the site table must not contain blank lines and must be in the order specified above
- no line must be left out except that concerning TSAPs if the length of these is null.

For example, the description of the table can end with:

```
length of local TSAP:      0
length of remote TSAP:     5
remote TSAP:               1,4,0x12,0xf2,112
```

A validity check is performed on the values in the table by **connect**. For example: *input network window* must be within the range of 1 through 127.

## Meaning of Fields

**SITE–NAME** — Must not contain spaces or unprintable characters and is taken as the argument for `connect`.

**service names** — This field is no longer in use but is still present for backward compatibility.

**class of transport** — Class of transport.
For UFT, values are:
**-** 2 for a public or private X25 link
**-** 4 for an Ethernet, Token Ring or FDDI link.
Class 0 is not permitted for UFT as it does not include the express flow at transport level.

**alternate class** — Backup class for transport where the transport class proposed by the local system is refused by the remote host during negotiation. *alternate class* must be less than or equal to *class_of_transport*. If equal, one single class is imposed.

**expedited data** — Sets *express flow* on transport where:
**-** 0 no express flow on transport
**-** 1 express flow permitted on transport.
Setting the *express flow* on the transport enables management of transfer interrupts. See Section “Restarting Interrupted File Transfer” on page 2-16.

| | |
|---|---|
| **flow control** | Sets the flow control on the transport where:<br>**-** 0 no flow control on the transport<br>**-** 1 flow control on the transport.<br>flow control on transport is recommended. |
| **credit** | Credit on the transport protocol must be set as a function of *tpdu size*. A credit of 3 for a tpdu of 1024 bytes is recommended. |
| **class 4 checksum control** | Sets an additional check for class 4 transport to allow detecting errors during the message transportation:<br>**-** 0 no additional check<br>**-** 1 additional check. |
| **tpdu size** | Size of tpdu (transport protocol data unit) in bytes. Each tpdu is a data frame transmitted to the network layer. A size of 1024 bytes is recommended regardless of the type of network. |
| **nsdu size** | Size of nsdu (network service data unit) in bytes. Each nsdu is a service frame transmitted to the link layer. A size of 1024 bytes is recommended regardless of the type of network. |
| **npdu size** | Size of npdu (network protocol data unit) in bytes. Each npdu is a data frame transmitted to the link layer.<br>**For UFT, this field is not used.** |
| **input network window** | Size of the reception window at network interface layer.<br>**For UFT, this field is not used.** |
| **output network window** | Size of the sending window at network interface layer.<br>**For UFT, this field is not used.** |
| **type of network** | Type of network for access to the remote host:<br>**-** 1 for an X25 public or private network<br>**-** 2 for an Ethernet, Token Ring or FDDI local area network<br>**-** 3 for FULL–IP (X25 or LAN network with NSAP address)<br>**-** 4 for NULL–IP (Ethernet, FDDI, Token Ring with SNPA address)<br>**-** 5 for SPEE (NSAP address with IP or LAN and without IP on X25) or for NetShare. |

**type of connection:** Type of connection in the case of an X25 public or private network:
- 0 switched virtual circuit (SVC)
- 1 permanent virtual circuit (PVC).

**local address** Network address of local system. The format of the address must be compatible with the type of network:
- X25 address format
- Ethernet, Token Ring, FDDI address format
- local NSAP for FULL–IP, SPEE or NetShare.

**remote address** Network address of remote host. The format of the address must be compatible with the type of network:
- X25 address format
- Ethernet, Token Ring, FDDI address format
- remote NSAP for FULL–IP, SPEE or NetShare.

**length of network facilities** Length in bytes of the *network facilities* field, a value between 1 and 10, or else blank if there are no *network facilities*.

**network facilities** List of digits corresponding to the network facilities that are negotiated when setting up the connection. This field is only significant for an X25 network. Examples of facilities are *caller charging* and *closed subscriber group number*.

**length of local TSAP** Length in bytes of *local TSAP* field from 0 through 16.

**local TSAP** List of digits corresponding to local TSAP (Transport Service Access Point).

**length of remote TSAP** Length in bytes of *remote TSAP* field from 0 through 16.

**remote TSAP** List of digits corresponding to remote TSAP (Transport Service Access Point). If the remote site is DPS 7/7000 or DPS 8, the remote TSAP must be 0x40, 0x01, *site name*.

**network calling data** List of digits corresponding to network call data.
**For UFT, this field is not used.**

The local address and remote address fields for the local and remote network addresses must comply with the format for the type of network (X25 or Ethernet):

- for X25, depending on the connection:
  - SVC connection,
    The local and remote network addresses are expressed as a string of 1 to 15 decimal digits.
    For example: `138061000` (TRANSPAC address)
  - PVC connection,
    The remote network addresses are expressed as a string of 8 characters.
    For example: `PVC00001` (TRANSPAC PVC)
- for Ethernet:
  List of 6 hexadecimal bytes separated by a comma.
  For example: `0x08, 0x00, 0x20, 0x04, 0x01, 0x02.`
- for FULL–IP:
  The local and remote addresses must be filled with valid NSAPs defined in the stack configuration:
  - the local address being a valid local NSAP,
  - the remote address being a remote NSAP defined with an entry in the RIB table.
    See “How to Add a RIB Entry” in *OSI Services Reference Manual*.
- for NULL–IP:
  The local address must be filled in with the local Ethernet address.
  The remote address must be filled in with the remote Ethernet address.
- for NetShare:
  The local and remote addresses must be filled in with the valid NetShare addresses. See the NetShare User’s Guide.

# Examples

## Access to a Remote Host via TRANSPAC

The following description enables connection to a remote host `SITE 001` via TRANSPAC:

```
SITE 001 uftp: 0x10  ufts: 0x10
class of transport (0, 2, 4): 2
alternate class (0, 2, 4): 2
expedited data (0, 1): 1
flow control (0, 1): 1
credit (1 to 15): 3
checksum control for class 4 (0, 1): 0
tpdu size: 1024
nsdu size: 1024
npdu size: 1024
input network window (1 to 127): 3
output network window (1 to 127): 3
type of network (1-X25 2-ETH 3-INTER 4-NULLINTER): 1
type of connection (0 SVC, 1 PVC): 0
local address: 138020100
remote address: 191193254
length of network facilities (1 to 10): 2
network facilities (list of digits): 0x03, 0x00
length of local TSAP (0 to 16): 6
local TSAP (list of digits): 0x40,1,2,3,4,5
length of remote TSAP (0 to 16): 6
remote TSAP (list of digits):
0x40,0x01,0x02,0x03,0x04,0x05
network calling data (list of digits):
```

## Access to a Remote Host via ETHERNET

The following description enables connection to a remote host `SITEA` via Ethernet:

```
SITEA ufts: 0x10  uftp: 0x10
class of transport (0, 2, 4): 4
alternate class (0, 2, 4): 2
expedited data (0, 1): 1
flow control (0, 1): 1
credit (1 to 15): 3
checksum control for class 4 (0, 1): 0
tpdu size: 1024
nsdu size: 1024
npdu size: 1024
input network window (1 to 127): 3
output network window (1 to 127): 3
type of network (1-X25 2-ETH 3-INTER 4-NULLINTER): 2
type of connection (0 SVC, 1 PVC): 0
local address: 0x08,0x00,0x38,0x20,0x04,0x03
remote address: 0x08,0x00,0x38,0x20,0x04,0x01
length of network facilities (1 to 10):
network facilities (list of digits):
length of local TSAP (0 to 16): 6
local TSAP (list of digits): 0x40,1,2,3,4,5
length of remote TSAP (0 to 16): 6
remote TSAP (list of digits):
0x40,0x01,0x02,0x03,0x03,0x04
network calling data (list of digits):
```

## Access to a Remote Host by Specifying NSAP of Remote Host

The following description enables connection to a remote host `DoradeFullIP` via FULL–IP:

```
DoradeFullIP uftp: 0x10  ufts: 0x10
class of transport (0,2,4): 4
alternate class (4): 4
expedited data (0,1): 1
flow control (0,1): 1
credit (1 to 15): 3
checksum control for class 4 (0,1): 0
tpdu size: 1024
nsdu size: 1024
npdu size: 1024
input network window (1 to 127): 3
output network window (1 to 127): 3
type of network (1-X25 2-ETH 3-INTER 4-NULLINTER): 3
type of connection (0 CVC,1 CVP): 0
local address: 0x1a
remote address: 0x2a
length of network facilities (1 to 10): 0
network facilities (list of digits):
length of local TSAP (0 to 16): 6
local TSAP (list of digits): 0x40,1,2,3,4,5
length of remote TSAP (0 to 16): 6
remote TSAP (list of digits): 0x40,1,2,3,4,5
network calling data (list of digits):
```

The values of the network dependent fields like `class of transport` depend on whether the NSAP is associated with Ethernet or the X25 network.

# Glossary

This glossary contains the abbreviations, keywords and phrases that can be found in this document.

**API**
Application Program Interface: Functional interface allowing a high level language application program to use specific data or functions of the operating system.

**ASCII**
American Standard Code for Information Interchange.

**COSP**
Connection Oriented Session Protocol.

**COTP**
Connection Oriented Transport Protocol.

**DPX**
Distributed Processor for UNIX.

**DSA**
Distributed Systems Architecture.

**DSA–62**
Bull proprietary file tranfer protocol.

**DSP**
Domain Specific Part.

**EBCDIC**
Extended Binary Code Decimal Interchange Code.

**Ethernet**
A baseband LAN specification (IEEE 802.3) using the CSMA–CD technique.

**FCB**
UFT Control Block.

**FDDI**
Communications adapter interface with a Fiber Distributed Data Network.

**GCOS**
General Comprehensible Operating System.

**HPAD**
Host PAD, Server side in the PAD client/server model.

**IDP**
Initial Domain Part.

**iFOR/LS**
Information For Operation Retrieval/License System.

**IP**
Internet Protocol.

**ISO**
International Standards Organization: Originator of Open Systems Interconnection reference model (ISO–IS 7498).

**LLC**
Logical Link Control. Protocol governing the assembly of transmission frames and their exchange between data stations, independent of the medium access control protocol.

**LPP**
Licensed Program Product.

**MAC**
Media Access Control**.**

**NFS**
Network File System: Protocol developed by Sun Microsystems allowing users to directly access files on other systems in a network.

**NSAP**
Network Service Access Point: A chain of 15 hexadecimal characters identifying the NSAP of a remote machine. It must be an even number of 40 characters maximum.

**NSDU**
Network Service Data Unit.

**OPP**
Optional Program Product.

**OSI**
Open Systems Interconnection: Reference model defined in OS–IS 7498.

**OTM**
Open Terminal Manager.

**PAD**
Packet Assembler Disassembler. Functional device enabling un–equipped Data Terminal Equipments to access a packet switching network.

**PID**
Process Identifier.
Logical number allocated by the system to a running process (indicated with the *"ps"* command).

**PVC**
**Permanent Virtual Circuit.** A virtual circuit which is permanently established between two DTEs. It ties up a logical channel permanently.

**RIB**
Routing Information Base, Network database which contains all the required routing information to remote NSAPs. It gives the remote SNPA and local subnetwork to use for NSAPs or groups of NSAPs. The SNPA is found by using the Subnet table.

**SMIT**
System Management Interface Tool (IBM): Menu–driven, resident command–building system management facility.

**SNPA**
Sub Network Point of Attachment: Information for accessing the system within the domain (Transpac or Ethernet address).

**SSAP**
Session Service Access Point

**SVC**
Switched Virtual Circuit. A virtual circuit which exists only for the duration of the call, acting like a connection over the normal telephone network, requested by a virtual call and released when the call is cleared.

**TCP**
Transport Control Protocol. Protocol used in ARPA Internet (U.S. Department of Defense standards for inter–networks).

**TCP–IP**
TCP and IP are the two fundamental protocols of the Internet protocol suite. (Acronym for this suite). TCP provides reliable transfer of data, while IP transmits.

**Token Ring**
Access procedure used with a sequential topology.

**TPAD**
Terminal PAD. Client side in the PAD client/server model.

**TPAD–HPAD**
Reference of the Bull application level that implements X.3, X.28, X.29, and Y/13 recommendations.

**TPDU**
Transport Protocol Data Unit.

**TRANSPAC**
French public packet–switched network offering connections in packet mode (X.25) or character mode (X28–X3).

**TSAP**
Transport Service Access Point.

**UCB**
User Control Block.

**UFT**
Unified File Transfer.

**UFTF**
Unified File Transfer on a Foreign Host.

**UNIX**
Portable operating system, implemented in “C” language.

**X25**
In data communication, a recommendation of the CCITT which defines the interface between DTE and packet-switching network.

# Index

## Symbols



## A



## B



## C



## D



## E

## F



## G



## H



## I



## L



## M



## N



## O



## P



## Q



## R



## S



## T



## U

## W



## X

# Vos remarques sur ce document / Technical publication remark form

**Titre / Title :** Bull DPX/20 Unified File Transfer (UFT) Reference Manual

**Nº Reférence / Reference Nº :** 86 A2 10CB 03

**Daté / Dated :** March 1996

ERREURS DETECTEES / ERRORS IN PUBLICATION

AMELIORATIONS SUGGEREES / SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Vos remarques et suggestions seront examinées attentivement
Si vous désirez une réponse écrite, veuillez indiquer ci-après votre adresse postale complète.

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please furnish your complete mailing address below.

NOM / NAME : _____ Date : _____

SOCIETE / COMPANY : _____

ADRESSE / ADDRESS : _____

_____

Remettez cet imprimé à un responsable BULL ou envoyez-le directement à :

Please give this technical publication remark form to your BULL representative or mail to:

**Bull Electronics Angers S.A.**
**CEDOC**
Atelier de Reprographie
331 Avenue Patton
49004 ANGERS CEDEX 01
FRANCE

PLACE BAR CODE IN LOWER LEFT CORNER

**Bull Electronics Angers S.A.**
**CEDOC**
Atelier de Reprographie
331 Avenue Patton
49004 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
**86 A2 10CB 03**

Bull

Utiliser les marques de découpe pour obtenir les étiquettes.
Use the cut marks to get the labels.

**DPX/20**
AIX
Unified File
Transfer (UFT)
Reference Manual

86 A2 10CB 03

**DPX/20**
AIX
Unified File
Transfer (UFT)
Reference Manual

86 A2 10CB 03

**DPX/20**
AIX
Unified File
Transfer (UFT)
Reference Manual

86 A2 10CB 03