

The Trainer's Friend, Inc.

6790 E. Cedar Ave., Suite 201
Denver, Colorado 80224
U.S.A.

Telephone: (303) 393-8716

E-mail trainers@trainersfriend.com
Internet www.trainersfriend.com

File RePackager

User's Guide

Version 1.1

The following terms that may appear in these course materials are trademarks or registered trademarks:

Trademarks of the International Business Machines Corporation:

AS/400, IBM, Language Environment, MVS, TSO, WebSphere, z/OS, z/VM, z/Architecture, zSeries, z9, z10

Trademarks of Microsoft Corp.: Microsoft, Windows, Windows XP, Windows Vista

Trademark of American National Standards Institute: ANSI

Registered Trademarks of Institute of Electrical and Electronic Engineers: IEEE, POSIX

Registered Trademark of The Open Group: UNIX

Trademark of Sun Microsystems, Inc.: Java

Registered Trademark of Linus Torvalds: LINUX

Registered Trademark of Unicode, Inc.: Unicode

Trademarks held on behalf of World Wide Web Consortium: W3C, XHTML, XSL, WebFonts

Trademark of The Trainer's Friend, Inc.: File Repackager

Table of Contents

| | |
|---|----|
| Introduction to File RePackager | 3 |
| Introduction | 4 |
| File characteristics | 8 |
| DD Statement Parameters for HFS Files | 13 |
| PATH and FILEDATA | 13 |
| PATHOPTS | 14 |
| PATHMODE | 15 |
| Order of Processing | 17 |
| Syntax rules | 19 |
| Control statements | 20 |
| OPTIONS | 26 |
| xFILE commands: IFILE, OFILE | 28 |
| Describing Input and Output files using DD statements and commands | 33 |
| Codepage Basics | 39 |
| Unicode | 42 |
| CCSIDs | 45 |
| Codepages in File RePackager | 46 |
| Codepages supported in File RePackager | 47 |
| File RePackager Samples | 51 |
| QSAM -> QSAM, output has longer records than input, padded with default (spaces) | |
| QSAM -> QSAM, output has longer records than input, padded with zeros | |
| QSAM -> QSAM, output has longer records than input, padded with default (spaces), convert from EBCDIC to ASCII | |
| QSAM -> QSAM, convert from ASCII to EBCDIC | |
| QSAM -> QSAM, output has longer records than input, padded with c'3's, convert from EBCDIC to ASCII | |
| QSAM file to z/OS UNIX file | |
| z/OS UNIX file in, copy to member in PDS | |

| | |
|--------------------------|----|
| Messages and Codes | 55 |
| Abend Codes | 56 |
| Return Codes | 57 |
| Messages | 58 |

Introduction to File RePackager

File RePackager from The Trainer's Friend

- ◆ Introduction

- ◆ File characteristics

- ◆ DD Statement Parameters for HFS Files

 - ✗ PATH and FILEDATA

 - ✗ PATHOPTS

 - ✗ PATHMODE

- ◆ Order of Processing

Introduction

File RePackager is a collection of modules that provide a flexible, powerful mechanism to copy and transform files:

The inputs and outputs can have any of these characteristics:

Sequential files or member of a PDS or PDSE

- * Fixed length or variable length records**
- * Character string data can be encoded in any supported codepage**

z/OS UNIX files

- * Type TEXT, with record delimiters of NL, LF, CR, LFCR, or CRLF**
- * Type BINARY with length prefixes of 1- or 2- or 4-byte binary integer**
- * Type BINARY with fixed length records**
- * Type RECORD (z/OS 1.11 and later)**
- * In all cases, character string data can be encoded in any supported codepage**

You can mix and match the inputs and output, so that you can transform the structure of records, the size of the records, and the encoding of character data in the records.

You can skip a number of input records before you start copying, and you can specify a number of output records after which you stop writing.

File RePackager supports a number of EBCDIC and ASCII code pages. See chapter three for the complete list of supported code pages.

Introduction, 2

The potential applications, then, include these:

Build test files from existing data

Work with files encoded in different codepages than you normally use in your installation

Accept files from non-EBCDIC systems and convert them for processing on the mainframe

Create versions of mainframe files that can be transferred to non-mainframe platforms for processing there

So, for example, these capabilities might be helpful in migrating to or from the mainframe.

In any event, there are, of course, restrictions / limitations:

- * Executable files (load modules and program objects) cannot be copied
- * Commands must be entered in a supported EBCDIC code page

Introduction, 3

The File RePackager accomplishes its work based on commands you enter.

The file with DDname TTFMDS contains the commands. This file can be any sequential file, member or a PDS/E, or z/OS UNIX file with a record size ≥ 80 . If the file is empty, the default action of copying the input data file to the output data file, unchanged, is performed. (Note: an empty file is not the same as a missing file. To signify an empty file, use DD * with no lines or DSN=NULLFILE on the DD statement.)

A log of the progress through the process is written to a data set using a DDname of TTFLOG; this is expected to be a SYSOUT file. This log contains notes about the various tasks performed, diagnostic messages, and informational messages such as the number of records read and written.

The primary input file has a DDname of TTFIN, and the primary output file has a DDname of TTFOUT. Both must be valid DD statements for the task at hand.

The data files (TTFIN and TTFOUT) may be specified in JCL (using DSNNAME for MVS files or PATH for z/OS UNIX files) or, if the files are z/OS UNIX files, you may specify a PATH parameter on the control statement and omit the DD statement. There is a discussion of the differences on the next page.

Supporting modules, such as codepage translation tables and configuration type information, are stored in the library containing the main program, TTFUIN1.

Introduction, 4

Here's a closer look at options for processing the data files (as opposed to the commands and log files). That is, the files referenced by TTFIN and TTFOUR.

If you specify either of these files using a DD statement, you can process both MVS data sets and z/OS UNIX files.

If you are processing z/OS UNIX files specified through DD statements, the program will use the QSAM interface. This means:

- * DD statements will have to have a PATH parameter
- * DD statements will have to include LRECL, RECFM, and possibly BLKSIZE parameters
- * For the output file, TTFOUR, you may need FILEDATA, PATHDISP, PATHMODE, and / or PATHOPTS parameters (discussed in more detail shortly)
- * DD statements for input files (TTFIN) that are z/OS UNIX files, usually require a FILEDATA parameter in addition to PATH

Alternatively, if you specify PATH on a command (IFILE or OFILE), the utility will use kernel services for I/O. In this case, you may need to supply additional information on the command, and the related DD statement will be ignored.

File Characteristics

To better understand the options and possibilities, we need to describe the characteristics of both MVS files and z/OS UNIX files. This will make it possible to explain what's possible and what does not make sense for this utility.

MVS Files

Generally speaking, MVS files contain records that are variable or fixed in length. In z/OS terms, this is described as RECFM (the record format). So RECFM=V or RECFM=F, respectively. The size of a record is called its LRECL (logical record length) and is an integer specifying how many bytes long the record is.

Files with fixed length records all have the same LRECL. The maximum record length is 32760 bytes. (There is support for tape files to use block sizes larger than 32760 bytes, but this utility does not currently support this option.)

Variable length records contain records of differing lengths. To find the length of any record, each record is stored with a length prefix, called an RDW (Record Descriptor Word). The prefix is four bytes long: 2 bytes containing the length of the Data + the length of the prefix; followed by 2 bytes used for flags when working with spanned records.

Since this utility does not support spanned records, we can safely assume variable length records have an RDW with two bytes (binary integer) length and two bytes of binary zeros. The maximum length of data is 32756, since we need to allow for the RDW (actually, there is a hidden BDW [Block Descriptor Word] for each physical block, so the true maximum data length is 32752 bytes).

File Characteristics, 2

There is actually a third possible record format, U, Undefined. Undefined format records are variable length records without any RDW or BDW blocks. Undefined records are used to hold load modules and program objects (executable programs), and these are not supported for this utility. (RECFM=U)

Unfortunately, if you process z/OS UNIX files using QSAM (as the utility does), the system tells the program that the records are RECFM U, and does not normally supply useful information about LRECL. This is why you need to specify RECFM and BLKSIZE or LRECL (and possibly both) on DD statements that point to z/OS UNIX files.

z/OS UNIX Files

Files of this type are stored in the Hierarchical File System (HFS) of directories, sub-directories, and files in the UNIX style. IBM provides an enhanced version called z/OS File System (zFS). Application programs really can't tell the difference (differences are internal and relate to system issues) so we may use the term "HFS" as a shorthand for "HFS or zFS" and "z/OS UNIX files".

File Characteristics, 3

z/OS UNIX Files, continued

There are several variations of how data is stored in the HFS:

Binary files are strings of bytes with no delimiters. The only known "size" is the total number of bytes. These are used for executable programs and files such as images (bitmaps, JPEGs, etc.) and other media storage. These are not supported as either input or output by the utility. They are simply not of interest here ... except that ...

Sometimes files are stored as binary that actually have structure to them: it's just that no one tells the file system about it. This is how you can store data that has a mixture of text (character string) data and binary types (such as packed decimal, floating point, and binary integer numbers). The utility can work with files of this nature, but it needs to know how to extract records from the byte stream. You tell the utility about the embedded record structure using one of these parameters on the utility control statement:

| | |
|--------------------------------------|---|
| RECLLEN=<i>n</i> | records are actually fixed length, <i>n</i> bytes long (the utility currently supports $1 \leq n \leq 131072$ (that is, 128K)) |
| RECPREFIX={1 2 4} | records have a prefix that contains the length of the data; this prefix is a 1-, 2-, or 4-byte binary integer (<u>not</u> formatted like an RDW) |
| RECDELIM={CR LF CRLF LFCR NL} | identifies some character combination that marks the end of each record: CR - Carriage Return LF - Line Feed NL - New Line |

Each of these use the appropriate code page values

Note that if the file's code page is not the default, you must code the **cp=** parameter before any **recdelim=** parameter in order to use the correct code page values. Also observe that if the data includes bit patterns that match the delimiter, you will not be able to correctly determine record boundaries.

File Characteristics, 4

z/OS UNIX Files, continued

Text files are an alternative way of organizing data in HFS files. For these kinds of files the data is presumed to be all text (character string). Record boundaries are usually determined by a delimiter (which varies across operating systems). This utility supports text files where record boundaries are determined by the RECDELIM parameter (or RECDELIM and RECLLEN combined).

However, note that if you process such a file using QSAM or other z/OS UNIX facilities, only the EBCDIC delimiter of x'15' (NL) is recognized. So if you wish to have output in an ASCII or Unicode code page, you should handle the file as if it is a Binary file or a Record file.

Record files are the third way data may be organized in z/OS UNIX files (requires z/OS 1.11 or later). In this method, records are stored with a four byte length prefix. Currently the first byte is reserved (must be binary zeros); the length prefix contains the length of the data record (it does not include its own length).

Note that storing data as record files allows you to include both text and binary format fields in each record.

File Characteristics, 5

When you convert from one file to another file with a different length, you can specify FILL on the OFILE statement to indicate a character to use to pad a record to make it long enough; if the output record is shorter than the input record, you can specify an OPTION parameter that says its OK to do record truncation.

Note that if records are to be truncated, the utility first attempts to squeeze the record by deleting trailing characters that match the FILL specification. After the squeezing is done, if the resulting record is still too short, the OPTION statement TRUNCATE is used to determine what to do. Note that sequential files (and members of PDS/Es) that have a RECFM of V are not padded: they just have their RDWs adjusted.

The default is to use the LRECL of the input as the LRECL for the output, except adjustments are made for any RDW, prefix, or record delimiter character(s).

Also note, for data in HFS files, you may also specify RECLLEN and one of RECPREFIX or RECDELIM; RECLLEN includes only the length of the data; when reading, the utility will also allow for the additional prefix or delimiter; when writing, the utility will adjust for the prefix or delimiter. Note that you cannot have both RECPREFIX and RECDELIM for the same file.

DD Statement Parameters for HFS Files

When you point to an HFS file using JCL DD statements, you must have a PATH parameter: `PATH='/...'`. This parameter value is case sensitive and tells the system the data is in an HFS file.

You'll typically need a FILEDATA parameter, also:

```
FILEDATA={TEXT|BINARY|RECORD}
```

as discussed earlier.

Note: when you code PATH on a utility control statement, the FILEDATA parameter only takes a single character: `FILEDATA={T|B|R}`.

In addition to PATH and FILEDATA, there are some other parameters you should be aware of for use when you code DD statements for working with HFS files. If you code IFILE or OFILE commands instead of DD statements, these parameters are not seen and they are ignored (you will take the program's defaults).

The PATHOPTS DD statement parameter is used to tell OPEN what options to use. And the PATHMODE DD statement parameter is used to assign permission bits to a file when it is first created (so this would be for TTFOUT only, and not even then if you are reusing a file).

PATHOPTS

- ❑ Use this JCL DD statement parameter to give OPEN necessary information

Syntax

```
PATHOPTS=({ORDONLY|OWRONLY|ORDWR}  
           {OAPPEND|OCREAT[,OEXCL]|OSYNC|OTRUNC})
```

Where

- ◆ You can specify up to 7 of these, in any order (but only one from the first group of three)
 - X **ORDONLY** - open file for read only
 - X **OWRONLY** - open file for write only
 - X **ORDWR** - open for read and write
 - X **OAPPEND** - each write should add records to the end (if not specified, and file opened OWRONLY, writes overlay existing data)
 - X **OCREAT** - if file does not exist, create it; if some directory in the path does not exist, it will not be created and the open will fail; if file exists and OEXCL is also specified, the job step fails; if file exists and OEXCL is not specified, the existing dataset will be used
 - X **OEXCL** - conditional allocation: if the file exists, fail the step; if the file does not exist, create it; this option is ignored unless OCREAT is also specified
 - X **OSYNC** - do not return from callable services that write to this file until the actual physical I/O has been done
 - X **OTRUNC** - removes all data from the file (sets the file size to zero) if the file named on the PATH is an existing, regular file that has been successfully opened with ORDONLY or OWRONLY also specified

PATHMODE

- ❑ Use this JCL DD statement parameter to set permissions for a file if PATHOPTS includes OCREAT

Syntax

PATHMODE=(*permission*[,...])

Where

- ◆ *permission* is a list of one to 14 values, in any order, from this list:
 - X **SIRUSR** - owner can read file
 - X **SIWUSR** - owner can write to file
 - X **SIXUSR** - owner can search, if a directory, execute if a file
 - X **SIRWXU** - owner can read, write, search / execute
 - X **SIRGRP** - group member can read file
 - X **SIWGRP** - group member can write to file
 - X **SIXGRP** - group member can search, if a directory, execute if a file
 - X **SIRWXG** - group members can read, write, search / execute
 - X **SIROTH** - others can read file
 - X **SIWOTH** - others can write to file
 - X **SIXOTH** - others can search, if a directory, execute if a file
 - X **SIRWXO** - others can read, write, search / execute
 - X **SISUID** - system should set UID to owner's when run
 - X **SISGID** - system should set GID to owning group's when run

PATHMODE, continued

Notes

- ◆ Although they are available, the docs say you shouldn't use **SISUID** and **SISGID** - these settings are lost when the file is written to
- ◆ If you do not code **PATHMODE** when creating an HFS file using **JCL**, the default permissions are **000** - meaning no one has any access to the file

If you use **IFILE** instead of **TTFIN** the utility defaults to read only for the **PATHOPTS** value; **PATHMODE** is irrelevant when reading.

If you use **OFFILE** instead of **TTFOUT**, the utility defaults to write only for **PATHOPTS** and **rw-rw-rw** for **PATHMODE**

PATHMODE and **PATHOPTS** are only recognized if you use **DD** statements for a z/OS UNIX file.

Order of Processing

It might be useful to understand that all the commands from TTFMDS are processed first, then the data files are opened and the information from the files is used to complete the utility's internal control blocks.

Here's a sample step that runs File RePackager:

```
//STEP1      EXEC  PGM=TTFUIN1
//TTFLOG      DD   SYSOUT=*
//TTFMDS      DD   *
             ifile cp=819,path=/u/scomsto/utility/source/...,
             recdelim=crlf,filedata=t
//TTFOUT      DD   DISP=OLD,DSN=library(member)
```

- ◆ Notice the commands file (DD name TTFMDS): these are the instructions used to give the utility any special processing instructions

✗ In the absence of any such instructions, the default processing is simply to copy the input file (pointed at by DDname TTFIN) to the output file (pointed at by DD name TTFOUT)

- ◆ In the example above, we don't need TTFIN since the command ifile fully specifies the input file

□ The next chapter covers the syntax rules for the utility commands

This page intentionally left almost blank.

Control Statements

Syntax rules

Control statements

◆ OPTIONS

◆ xFILE commands: IFILE, OFILE

✗ Describing Input and Output files using DD statements and commands

Syntax Rules

☐ **Commands are coded in the invariant EBCDIC code set:**

◆ **A-Z, a-z, 0-9, and these special characters:**

- X - space
- X < - less than
- X (- left parenthesis
- X + - plus sign
- X & - ampersand
- X * - asterisk
- X) - right parenthesis
- X ; - semi-colon
- X - - dash / hyphen
- X / - forward slash
- X : - colon
- X , - comma
- X % - percent
- X > - greater than
- X ? - question mark
- X = - equal sign
- X ' - single quote, apostrophe
- X " - double quote
- X . - period, decimal point, full stop

Note

◆ **Code pages 1026 and 1155, both EBCDIC Turkish pages, do not map double quote the same way as all other EBCDIC code pages**

Syntax Rules, 2

- The command file should not have sequence numbers in the records
- The command file may contain blank lines anywhere
- Comment lines begin with an asterisk (*) in column 1 and may appear anywhere in the command file
- The general syntax of a command is:

operator operand[,operand...]

- ◆ The 'operator' is the command; it must begin in column 2 or later
 - ◆ Operands are separated by commas, no extra spaces are allowed; however, spaces in quoted strings are supported
 - ◆ All operands are keyword (*reserved_word=value*), so they may appear in any order on a statement
- The logical end of a line is designated by the first occurrence of:
 - * Physical end of input record, or
 - * An unquoted space after at least one operand
 - Any text past the logical end of line is treated as a comment

Syntax Rules, 3

- ❑ The names of operators (commands) and operands (the keywords) are case-insensitive

- ❑ Values chosen from pre-determined options are also case-insensitive
 - ◆ All other values are case sensitive

 - ◆ Specifically, the values in PATH operands are case sensitive

Command Continuation

□ Commands may be continued using these rules:

- ◆ A command must have an operator and at least [the beginning of] one operand on the first line
- ◆ If an operand string ends with comma-space (', ') or slash-space ('/ '), it indicates this command is continued on the following line
- ◆ Continuation lines operand values begin with the first non-blank character after column 1; this allows a continuation line to be commented out by placing an asterisk in column 1
- ◆ Keywords are not allowed to be split across lines
- ◆ If a quoted string needs to be continued, close the string on the first line, follow with a comma-space, then start with a quote on the next line; the contents of the string will be merged; for example:

```
FIELD LIT=C'<p style="color: blue">Here is',  
'<strong>the</strong> place to come.'
```

X Would be seen as:

```
FIELD LIT=C'<p style="color: blue">Here is <strong>the</strong> place to come.'
```

Control Statements

- The commands recognized by the TTFUIN1 parser are these:
 - ◆ **OPTIONS** - specifies any processing options in effect for this run
 - ◆ **IFILE** - specifies the code page the input file is encoded in
 - ◆ **OFILE** - specifies the code page the output file is to be encoded in

Notes:

- ✗ If the IFILE or OFILE (or both) commands are omitted, the default code page is assumed for that file
- ✗ All the xFILE commands may also include information on file and record characteristics

Control Statements, 2

Although these commands can be included in almost any order, these rules should be followed:

◆ If there is an **OPTIONS** command, it should come first

We start, then, with the **OPTIONS** command

Next we discuss the **xFILE** commands, which both use the same parameters except for two special cases

The OPTIONS command

Syntax

```
OPTIONS [RUN=NO]
        [,TRUNCATE={T|S|P}]
        [,TRANSSUB={N|B|X'nn'}]
        [,TRANSACT={I|R|S}]
        [,STARTAFTER=n][,STOPAFTER=m]
```

Notes

- ◆ **RUN=NO** requests the commands be analyzed but no files read from, or written to; this allows you to have the utility check the commands for you to see how the commands have been interpreted
- ✗ Note that the input file and output file will be OPENed, just not processed

The OPTIONS command, 2

◆ **TRUNCATE={T|S|P}** - action to take if output record too large

✗ T - truncate record and continue; S - skip record (do not write out) and continue; P - stop run (this is the default); in all cases, a log entry is made identifying the problematic record.

◆ **TRANSSUB={N|B|X'*nn*'}** - character to substitute when an invalid character is detected in a translation operation (default: X'01')

✗ N represents a null (x'00); B represents a blank (space) (x'40' for EBCDIC codepages, x'20' for ASCII code pages)

◆ **TRANSACT={I|R|S}** - action to take if an invalid (nonsupported) character is encountered in a translation operation

✗ I - ignore (that is, don't check for invalid characters); R - report (log), replace with substitution character, and continue; S - report and stop run; (default: R)

◆ **STARTAFTER=*n*** says to skip '*n*' records in the input file before writing output records; default is zero (0)

◆ **STOPAFTER=*m*** says to stop after writing '*m*' records out; a value of zero (0), which is the default, says to continue until end of input file is encountered

xFILE Commands

Syntax

```
xFILE [CP=nnnnn] [PATH=/...]  
      [FILEDATA={T|B|R}] [RECLEN=n]  
      [RECPREFIX={1|2|4}] [FILL={N|B|X'nn'}]  
      [RECDELIM={CR|LF|CRLF|LFCR|NL}]
```

Notes

- ◆ **IFILE** - for the primary data source file
- ◆ **OFILE** - for the data output file
- ◆ **All operands are optional; those specified may be specified in any order, separated by commas (although you must have at least one operand)**
- ◆ **CP=nnnnn** designates the code page the file is encoded in; code pages are specified as a string of 1-5 decimal characters (0-9); leading zeros may be omitted; the list of supported code pages is found in Chapter 3 of this document
 - ✗ File RePackager comes with a default code page of **01140** (EBCDIC USA with Euro support); instead of changing this (which is possible), you may want to override the default just for a particular run
 - ✗ Code pages are specified as a string of 1-5 decimal characters (0-9); leading zeros may be omitted; the list of supported code pages is found in Chapter 3 of this document

xFILE Commands, continued

- ◆ **PATH=/** - name of a z/OS UNIX file to use; if this parameter is missing, the file pointed at by DDname related to this file is used (which may be a z/OS UNIX file (in which case it will be processed using QSAM) or a classic MVS file (sequential or member of a PDS or PDSE)); value must not be in quotes
 - ✗ Identifying a file using PATH= causes z/OS UNIX kernel processing to be used for I/O, thus more information is needed:
- ◆ **FILEDATA={T|B|R}** - only meaningful if PATH specified; is file a Text file (only contains text strings and possibly delimiters), a Binary file (contains any data, nothing is recognized as a delimiter; may be fixed length) or a Record file (any kind of data, 4 byte length prefix)?
- ◆ **RECPREFIX={1|2|4}** - indicates each record has a binary prefix specifying the length of the following data, and the prefix is 1- or 2- or 4-bytes long
- ◆ **RECDELIM={CR|LF|CRLF|LFCR|NL}** - for FILEDATA=T only: what characters delimit records; these characters are encoded in the file's codepage
- ◆ **RECLLEN=*n*** - for z/OS UNIX files; signifies fixed length records; if RECPREFIX also specified, prefix contains *n* in all cases; if RECDELIM is specified, the delimiter string is appended to all records; cannot have both RECPREFIX and RECDELIM

FILEDATA, RECPREFIX, RECDELIM, and RECLLEN all require PATH; the supported combinations of parameters are discussed later

xFILE Commands, continued

- ◆ **FILL={N|B|X'*nn*'}** - what character should be used to fill omitted positions; used when an output record is composed of fields that have gaps in their starting positions: Nulls, Blanks, or a specific character in hexadecimal; only valid for OFILE command; ignored if output is QSAM variable length record
 - ✗ N represents a null (x'00); B represents a blank (space) (x'40' for EBCDIC codepages, x'20' for ASCII code pages)
 - ✗ This value is also used as a pad character to make sure data fills a specified record length, if needed; the default fill character is B (blank, in the codepage of the output file)
 - ✗ This value is also used when trying to fit an output record into the available space: trailing characters equal to this value are trimmed until the record will fit; if such trimming still does not allow the output record to fit, the OPTIONS command TRUNCATE value determines what action to take

xFILE Commands, continued

- Here we summarize how the utility interprets and handles various combinations of parameters on these statements
 - ◆ **PATH** - file is found (IFILE) or to be stored (OFILE) in the HFS; related DD statement will not be opened
 - ◆ **RECLen, RECPREFIX, REDELIM, and FILEDATA** all require **PATH**
 - ◆ **RECPREFIX** and **RECDELIM** are always mutually exclusive
 - ◆ **PATH** requires **FILEDATA** (a value will be assigned if there is no **FILEDATA** parameter)
 - ✗ If **FILEDATA=T**, **RECDELIM** is required; the assumption is records contain text delimited by the **RECDELIM** character(s); **RECPREFIX** is not allowed
 - ✗ If **FILEDATA=T** and **RECDELIM** and **RECLen** are specified, all records are assumed to be the same size, contain only text, and the length includes the delimiter character(s)
 - ✗ If **FILEDATA=B**, then must have one of these situations:
 - **RECLen** is set; all records have the same length
 - **RECPREFIX** is set; records are variable length, determined by the prefix value
 - **RECLen** and **RECPREFIX** are both set: all records have the same length, which is **RECLen** plus the length of the prefix; for input specify the **RECLen** value as the length of the records plus the length of the prefix
 - **RECLen** and **RECDELIM** are set: all records have the same length, which is **RECLen** plus the length of the delimiter(s)

xFILE Commands, continued

- Here we summarize how the utility interprets and handles various combinations of parameters on these statements, continued
 - ◆ **PATH requires FILEDATA (a value will be assigned if there is no FILEDATA parameter) - continued**
 - FILEDATA=RECORD is new with z/OS 1.11; we support this using the xFILE parameter FILEDATA=R; RECPREFIX is not allowed for this setting, nor is RECDELIM
 - ✗ If FILEDATA=R without RECLen, records are prefixed by a 4-byte length code containing the length of the data
 - ✗ If FILEDATA=R with RECLen, records are prefixed by the 4-byte length code and its value is the same for all records (RECLen is the data length: it does not include the prefix)
 - ◆ **PATH requires at least one of RECLen, RECPREFIX, RECDELIM, or FILEDATA=R; if not, an error is signalled**
 - ◆ **If PATH is specified without FILEDATA:**
 - ✗ It is an error if both RECDELIM and RECPREFIX are present
 - ✗ If RECDELIM is specified, but not RECLen nor RECPREFIX, FILEDATA is set to T
 - ✗ Otherwise, FILEDATA is set to B

Note:

FILEDATA as a JCL DD statement parameter must have one of these values: TEXT, BINARY, RECORD; DD statements are case sensitive

FILEDATA as an operand on a File RePackager command must have one of these values: T, B, R; commands are case insensitive

Describing Files

All supported file types may be used as input and output; here is a discussion of the JCL and / or utility command parameters that must / may be specified for the supported file types

- ◆ Remember TTFIN and TTFOUT are the names of DD statements, while IFILE and OFILE are the names of commands in the TTFMDS file

- ☐ For **input of MVS, RECFM=F**, code TTFIN with DISP=SHR and DSN=*dsname*; you do not need IFILE except to request a non-default codepage

- ☐ For **input of MVS, RECFM=V**, code TTFIN with DISP=SHR and DSN=*dsname*; you do not need IFILE except to request a non-default codepage

- ☐ For **input of HFS, filedata of TEXT**
 - ◆ Code TTFIN with PATH, FILEDATA=TEXT, and LRECL set to an integer equal to or greater than the largest record in the file

 - ◆ Or code IFILE with PATH and FILEDATA=T; you may also specify RECLLEN, but it is not needed

 - ✗ For either case, code CP on IFILE if code page is not the default code page; in this case, you must also code RECDELIM on IFILE to specify delimiting character(s)
 - There is no default value for RECDELIM

Describing Files, 2

- For **input of HFS**, filedata of **BINARY**, **fixed length** records, **no prefix**
 - ◆ Code TTFIN with PATH, FILEDATA=BINARY, and LRECL= n
 - ◆ Or code IFILE with PATH, FILEDATA=B, and RECLLEN= n
 - ✗ For either case, code CP on an IFILE if code page is not the default code page for any text strings in the data and you want code page translation done

- For **input of HFS**, filedata of **BINARY** records, **with prefix**
 - ◆ Omit TTFIN; code IFILE with PATH, FILEDATA=B, and RECPREFIX={1|2|4}; if records are fixed length may also code RECLLEN= n , where n includes the record length plus the length of the prefix
 - ✗ Code CP if code page is not the default code page for text strings and you want code page translation done

- For **input of HFS**, filedata of **BINARY**, **fixed length** records, **no prefix**, with **delimiter**
 - ◆ Omit TTFIN; code IFILE with PATH, FILEDATA=B, RECLLEN= n , and RECDELIM={CR|LF|CRLF|LFCR|NL}
 - ✗ Code CP if code page is not the default code page for any text strings in the data

Describing Files, 3

- For **input of HFS**, filedata of **RECORD** (z/OS 1.11 and later only)
 - ◆ Code TTFIN with PATH, FILEDATA=RECORD and LRECL set to an integer equal to or greater than the largest record in the file
 - ◆ Or code IFILE with PATH, FILEDATA=R, and RECLEN=*n*
 - ✗ For either case, code CP on an IFILE if code page is not the default code page for any text strings in the data and you want code page translation done

- For **output of MVS, RECFM=F**, code TTFOUT with DSN=*dsname*, and either DISP=OLD or DISP=(,CATLG) with LIKE= or SPACE, RECFM, LRECL, and BLKSIZE; you do not need OFILE except to request a non-default codepage or to specify a FILL character if length is increasing

- For **output of MVS, RECFM=V**, code TTFOUT with DSN=*dsname*, and either DISP=OLD or DISP=(,CATLG) with LIKE= or SPACE, RECFM, LRECL, and BLKSIZE; you do not need OFILE except to request a non-default codepage; note: no padding is done for this kind of file

- For **output of HFS**, filedata of **TEXT**
 - ◆ Code TTFOUT with PATH, FILEDATA=TEXT and LRECL set to an integer equal to or greater than the largest record in the file
 - ◆ Or code OFILE with PATH and FILEDATA=T; you may also specify RECLEN
 - ✗ For either case, code CP on OFILE if code page of file is not the default code page or to specify a FILL character if length is increasing; must also code RECDELIM on OFILE to specify delimiting character(s)

Describing Files, 4

For **output of HFS**, filedata of **BINARY, fixed length** records, **no prefix**

◆ Code TTFOUT with PATH, FILEDATA=BINARY, and LRECL= n

◆ Or code OFILE with PATH, FILEDATA=B, and RECLLEN= n

✗ For either case, code CP on an IFILE if code page is not the default code page for any text strings in the data and you want code page translation done

For **output of HFS**, filedata of **BINARY** records, **with prefix**

◆ Omit TTFOUT; code OFILE with PATH, FILEDATA=B, and RECPREFIX={1|2|4}; if records are fixed length may also code RECLLEN= n , where n includes the record length plus the length of the prefix

✗ Code CP if code page is not the default code page for text strings and you want code page translation done

For **output of HFS**, filedata of **BINARY, fixed length** records, **no prefix, with delimiter**

◆ Omit TTFOUT; code OFILE with PATH, FILEDATA=B, RECLLEN= n , and RECDELIM={CR|LF|CRLF|LFCR|NL}

✗ Code CP if code page is not the default code page for any text strings in the data

Describing Files, 5

- For **output of HFS**, filedata of **RECORD** (z/OS 1.11 and later only)
 - ◆ Code TTFOUT with PATH, FILEDATA=RECORD and LRECL set to an integer equal to or greater than the largest record in the file
 - ◆ Or code OFILE with PATH, FILEDATA=R, and possibly RECLEN=*n*
- ✗ For either case, code CP on an OFILE if code page is not the default code page for any text strings in the data and you want code page translation done or to specify a FILL character if length is increasing (in which case, RECLEN must have been specified)

This page intentionally left almost blank.

Code Page Support

- Codepage Basics**
- Unicode**
- CCSIDs**
- Codepages in File RePackager**
- Codepages supported in File RePackager**

Codepage Basics

When the modern mainframe appeared in the mid-1960's, design decisions were made that would influence computing for decades to come.

One of the early considerations was how to represent the common English alphabet electronically. Things started out this way: when you pressed a key on a keyboard, the keyboard is designed to send a pattern of eight bits (electronic 1's and 0's), a byte. The selection of what pattern of bits are sent by what characters are somewhat arbitrary, although some order was designed into the mapping also.

On the IBM mainframe, the mapping originally used was called EBCDIC (Extended Binary Coded Decimal Interchange Code). Each character is assigned to one byte, and a byte has 8 bits, which allows for 256 distinct values. Theoretically, this allowed for 256 characters. However, for historical and practical reasons some bit patterns have to be reserved for control functions (for example, tab, new line, even 'space').

This was fine until people wanted to use mainframes in countries where the local language wasn't English. These languages had characters in their alphabets not in the classic English alphabet. And many languages had their own unique characters.

To provide for data entry using these languages, new bit patterns had to be supplied for the extra characters. Things got a little out of control for a while and we ended up with many flavors of EBCDIC. Some might work for multiple languages, but most flavors had different code points for the national alphabetic and for some of the punctuation characters.

The mapping between abstract characters and bit patterns is called a code page (or "codepage" - one word or two, both work).

Codepage Basics, 2

Now EBCDIC was used on mainframes and, later, on the IBM AS/400 series of machines. But other standards were popping up all over. The most popular non-mainframe code page is what we now call ASCII (American Standard Code for Information Interchange).

ASCII was originally a 7-bit code that was later expanded into what is now called "Latin-1". Like EBCDIC, ASCII too has several variants.

Neither of these codepage families were particularly useful for Asian customers, however. The CJK (Chinese-Japanese-Korean) family of languages, for example, have thousands of characters, way more than can be represented in 256 values. Separate standards grew up around those languages, usually using 16 bits (2 bytes) per character, which allows for 65,536 distinct values.

In 1946 a collection of national governments formed the International Standards Organization (ISO) to develop a set of standards to be agreed on and used by all its members (and anyone who wanted to do business with its members). To date, ISO has published over 17,500 standards in virtually every field of human endeavor.

One area of interest to us, is in codepages. The ISO provided a place to formalize all existing standard codepages, to formally name the various codepages and to record precisely what mappings constituted each codepage. There were still many codepages, but at least there was agreement as to what was in each codepage.

Unicode

In 1991 a consortium of industry leaders incorporated a non-profit organization funded by all of these companies under the name The Unicode Consortium, with the charge to bring some sense to this chaotic situation. Out of this has come the Unicode system: a Universal code page, with the goal of establishing agreed upon character mappings for every human written language.

The starting point is the ISO standard named ISO-10646 and, in fact, these two bodies have worked hard to ensure these two standards are kept in sync. For they are both works in progress. Periodically new characters are added and older characters might be reassigned or at least have their properties re-specified.

The most recent version of the Unicode standard is 5.2.0 (released October 2009) and it contains assignments of bit patterns to 107,296 characters! Clearly 16-bit characters are not enough! So the Unicode consortium came up with an interesting approach.

Start with a number of bits that should allow for all we will ever need. That number was decided to be 21 bits. A 21-bit integer has 1,114,112 distinct values. That should do it!

Because computers work optimally with units and powers of 2, we find a 32-bit integer (four bytes) is a nice container. So let's start there. We call this 21-bits-of-value-in-32-bits-of-space UTF-32. UTF stands for Unicode Transformation Format. Every included character is assigned to a number in the theoretical range 0 to 1,114,112.

Unicode, 2

The reality is, we're using less than one tenth of that capacity so far. Now we try and see how we can get more efficient.

The first step is to use UTF-16. This is a 16 bit character. But, as we've seen, this would only allow for 65,536 values. Here's where things get clever: a range of this set of number is set aside for use in surrogate pairs. In particular, the range of values from 55,296 to 56,319 is used for "high surrogate" values and the range from 56,320 to 57,343 is used for "low surrogate" values.

In hexadecimal, these are x'D800'-x'DBFF' and x'DC00'-x'DFFF'. (Note that each range contains 1024 values.)

We remove these ranges from being eligible for representing single characters and establish this rule: whenever a UTF-16 value is found to be in the high surrogate range, it must be followed by a value from the low surrogate range, and the combination of the two 16-bit values is mapped to a single character.

So using surrogate pairs, you can get $1,024 \times 1,024 = 1,048,576$ values. That is, using UTF-16, most commonly used characters take up two bytes (16 bits). But some characters require four bytes (32 bits) to be represented.

The mapping between characters represented in UTF-32 and their UTF-16 representations turns out to be purely mechanical: a single computer instruction can convert from one to the other simply.

So now, instead of using 4 bytes for every character, we can get by with just 2 bytes for most characters and 4 bytes for some (less frequently used) characters.

But there's one more step...

Unicode, 3

It turns out the first 127 values in UTF-32 are the same as the first 127 values in ASCII. So if you are using ASCII, and if you are only using common Western characters, you are, in a sense, using Unicode. In fact, there is a third UTF: UTF-8. In UTF-8, every Unicode character is represented in 1, 2, 3, or 4 bytes! Code points 0-127 in ASCII are the same as code points 0-127 in UTF-8.

The details are beyond the scope of this document, but it turns out there is a mechanical procedure to map between UTF-8, UTF-16, and UTF-32 formats. We don't need to know the details here, but we do need to know:

- * Unicode data may come in UTF-8, UTF-16, or UTF-32

- * The size of a Unicode character is always four bytes in UTF-32, usually two bytes (but sometimes four) in UTF-16, 1-4 bytes in UTF-8.

- NOTE:** Version 1 of File RePackager does not support Unicode files; it is intended that Version 2 will provide full support of UTF-8, UTF-16, and UTF-32 codepages.

CCSIDs

Character set names use terms like "CCSID" - Coded Character Set ID (an integer), "CP" - Code Page, plus various aliases. It can all be quite confusing.

Since File RePackager is developed to run on z/OS, we decided to use this official IBM site when deciding what code pages to support:

http://www.ibm.com/software/globalization/ccsid/ccsid_registered.jsp

Note that CCSIDs are five-digit numbers, but when you specify a codepage to the utility you can omit leading zeros.

Codepages in File RePackager

This is all just background, however, unless you want to modify any of the codepage tables we supply, or if you want to add your own codepage tables. Details on how to do this appear in the Installation and Customization Guide.

One of the major design points in File RePackager is to support data in encoded different codepages, allowing you to convert between any two of the supported codepages.

When you tell the utility about the input file, you specify the codepage it is encoded in and when you describe the output file you specify what codepage the output should use for characters. The utility automatically copies in the relevant codepage conversion table and ensures all text data in the output will be encoded in the desired codepage.

The utility command statements must be coded in a supported EBCDIC codepage (actually, almost all EBCDIC code pages will work for this, except the Japanese and Turkish code pages).

Codepages Supported in File RePackager

| <u>CCSID</u> | <u>Name(s) and Notes</u> |
|--------------|--|
| 00037 | EBCDIC USA |
| 00367 | US ASCII-7, also US ANSI X3.4 ASCII |
| 00819 | ASCII Latin-1; also known as ISO-8859-1; also ISO Latin-1 Western European ASCII |
| 00923 | ASCII Latin-9: also know as ISO-8859-15 |
| 00924 | EBCDIC Latin-9; codepage 1047 + the Euro, but there are some other code point differences. |
| 01047 | EBCDIC: Latin-1/Open Systems |
| 01140 | EBCDIC: 00037 + Euro |

To say File RePackager "supports" these codepages means that you can have any of these codepages as input and any as output; if the input and output codepages are different, the utility will automatically convert string fields in the input codepage to string fields in the output codepage as output records are built.

Note: telling the File RePackager to convert between codepages is only viable if all the data in the source records is character; any data stored in packed decimal, binary integer, or internal floating point formats will not be treated correctly.

Special note: if numeric data was created on the mainframe as COBOL data with an S in the picture and no usage clause, e.g.: PIC S9(5)V99, without COMP, COMP-1, COMP-2, COMP-3, BINARY, or PACKED-DECIMAL the last digit will be represented as an alphabetic character or brace ({ or }) instead of a numeric digit. This will cause an error in translation also.

Sample TTFLOG outputs

```
TTFU000I FileRePackager Version v.r.m logging started.

TTFE001I CPU-id: cpuid. License-id: dddddd-dddddd.
TTFE002I This copy of the software is licensed for use only by xxx
TTFE002I      at the location known as: YYYYYYYYYYYYYYYY.
TTFE003I Use by any other organization or individual is illegal and
          expressly forbidden.
TTFE003I Anyone discovering an unauthorized use is requested to
          report it to The Trainer's Friend at 303-393-8716.

TTFE004I Copyright 2010 by The Trainer's Friend, Inc.
TTFE004I Call The Trainer's Friend at 303-393-8716 for support.

TTFU019I Input commands:
TTFU031I :   ifile cp=819
TTFU031I :   ofile fill=x'f3'

TTFU051I Run characteristics:
TTFU004I * Input file: RECFM=FB, LRECL=      80, BLKSIZE= 27,920
TTFU008I * Output file: RECFM=FB, LRECL=     100, BLKSIZE= 27,900
TTFU063I * Character strings will be translated from code page
          00819 to code page 01140
TTFU058I * All input data records will be processed.
TTFU052I End of run characteristics.

TTFU053I Processing Started.
TTFU054I Processing Ended.

TTFU065I Records read from input file:   144
TTFU066I Records written to output file: 144

TTFU900I Final completion code:   0
```

Sample TTFLOG outputs, continued

```
TTFU000I FileRePackager Version v.r.m logging started.

TTFE001I CPU-id: cpuid. License-id: dddddd-dddddd.
TTFE002I This copy of the software is licensed for use only by xxx
TTFE002I      at the location known as: YYYYYYYYYYYYYYYY.
TTFE003I Use by any other organization or individual is illegal and
          expressly forbidden.
TTFE003I Anyone discovering an unauthorized use is requested to
          report it to The Trainer's Friend at 303-393-8716.

TTFE004I Copyright 2010 by The Trainer's Friend, Inc.
TTFE004I Call The Trainer's Friend at 303-393-8716 for support.

TTFU019I Input commands:

TTFU051I Run characteristics:
TTFU014I * Input file:  RECFM=FB, LRECL=      80, BLKSIZE= 27,920
TTFU016I * Output file: RECFM=FB, LRECL=      74, BLKSIZE= 27,972
TTFU064I * No code page translations will be done.
TTFU058I * All input data records will be processed.
TTFU052I End of run characteristics.

TTFU053I Processing Started.
TTFU054I Processing Ended.

TTFU065I Records read from input file:  144
TTFU066I Records written to output file: 144

TTFU900I Final completion code:  0
```

Sample TTFLOG outputs, continued

```
TTFU000I FileRePackager Version v.r.m logging started.

TTFE001I CPU-id: cpuid. License-id: dddddd-ddddd.
TTFE002I This copy of the software is licensed for use only by xxx
TTFE002I      at the location known as: YYYYYYYYYYYYYY.
TTFE003I Use by any other organization or individual is illegal and
          expressly forbidden.
TTFE003I Anyone discovering an unauthorized use is requested to
          report it to The Trainer's Friend at 303-393-8716.

TTFE004I Copyright 2010 by The Trainer's Friend, Inc.
TTFE004I Call The Trainer's Friend at 303-393-8716 for support.

TTFU019I Input commands:
TTFU031I : ifile   PATH=/u/scomsto/utility/source/src_ina_record,
TTFU031I : FILEDATA=R
TTFU031I : ofile   PATH=/u/scomsto/utility/target/out_ina_record,
TTFU031I : FILEDATA=R

TTFU051I Run characteristics:
TTFU014I * Input file:  z/OS UNIX file.
TTFU016I * Output file: z/OS UNIX file.
TTFU064I * No code page translations will be done.
TTFU058I * All input data records will be processed.
TTFU052I End of run characteristics.

TTFU053I Processing Started.
TTFU054I Processing Ended.

TTFU065I Records read from input file:   212
TTFU066I Records written to output file: 212

TTFU900I Final completion code:  0
```

Section Preview

File RePackager Samples

- ◆ QSAM -> QSAM, output has longer records than input, padded with default (spaces)
- ◆ QSAM -> QSAM, output has longer records than input, padded with zeros
- ◆ QSAM -> QSAM, output has longer records than input, padded with default (spaces), convert from EBCDIC to ASCII
- ◆ QSAM -> QSAM, convert from ASCII to EBCDIC
- ◆ QSAM -> QSAM, output has longer records than input, padded with c'3's, convert from EBCDIC to ASCII
- ◆ QSAM file to z/OS UNIX file
- ◆ z/OS UNIX file in, copy to member in PDS

Samples

- In this section we provide sample job steps that accomplish representative work using File RePackager.
 - ◆ For new (output) data sets, we only code the minimal parameters to illustrate the point
 - ◆ We have not shown any JOBLIB or STEPLIB datasets you might need

1. QSAM, Fixed length record file, default codepage

-> QSAM Fixed length record file with records longer than the input file, default codepage:

```
//REPACK EXEC PGM=TTFUIN1
//TTFLOG DD SYSOUT=*
//TTFMDS DD DSN=NULLFILE
//TTFIN DD DISP=SHR,DSN=...
//TTFOUT DD DSN=...,DISP=(,CATLG),LRECL=nnnn,
// UNIT=SYSALLDA,SPACE=...
```

2. QSAM, Fixed length record file, default codepage

-> QSAM Fixed length record file with records longer than the input file, pad with character zeros, default codepage:

```
//REPACK EXEC PGM=TTFUIN1
//TTFLOG DD SYSOUT=*
//TTFMDS DD *
        ofile fill=x'f0'
//TTFIN DD DISP=SHR,DSN=...
//TTFOUT DD DSN=...,DISP=(,CATLG),LRECL=nnnn,
// UNIT=SYSALLDA,SPACE=...
```

3. QSAM, Fixed length record file, default codepage

-> QSAM Fixed length record file with records longer than the input file, output records in ASCII Latin-1 codepage:

```
//REPACK EXEC PGM=TTFUIN1
//TTFLOG DD SYSOUT=*
//TTFMDS DD *
    ofile cp=819
//TTFIN DD DISP=SHR,DSN=...
//TTFOUT DD DSN=...,DISP=(,CATLG),LRECL=nnnn,
// UNIT=SYSALLDA,SPACE=...
```

4. QSAM, Fixed length record file, ASCII Latin-9 codepage

-> QSAM Fixed length record file with records same size as the input file, output records in EBCDIC:

```
//REPACK EXEC PGM=TTFUIN1
//TTFLOG DD SYSOUT=*
//TTFMDS DD *
    ifile cp=819
//TTFIN DD DISP=SHR,DSN=...
//TTFOUT DD DSN=...,DISP=(,CATLG),LIKE=...
```

5. QSAM, Fixed length record file, default codepage

-> QSAM Fixed length record file with records larger than the input file; output file in ASCII, padded with '3's

```
//STEP1 EXEC PGM=TTFUIN1
//TTFLOG DD SYSOUT=*
//TTFMDS DD *
    ofile cp=819,fill=x'33'
//TTFIN DD DISP=SHR,DSN=...
//TTFOUT DD DISP=(,CATLG),DSN=...,
// LRECL=100,
// LIKE=...
```

Notice that any fill=x'..' value needs to be the value you want as it is represented in the codepage of the output file

6. QSAM, Fixed length records in, HFS file out, of type TEXT

```
//STEP1      EXEC  PGM=TTFUIN1
//TTFLOG      DD   SYSOUT=*
//TTFCMDS     DD   *
//TTFOUT      DD   PATH='/u/scomsto/utility/target/...',
//    PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//    PATHMODE=(SIRWXU,SIRWXG,SIROTH),
//    FILEDATA=TEXT,RECFM=F,LRECL=nn
//TTFIN       DD   DISP=SHR,DSN=...
```

- ◆ Note: empty TTFCMDS file
- ◆ On the TTFOUT DD statment, you need to specify an LRECL at least as large as the records in your input file; need to specify RECFM or OPEN will fail

7. HFS file in, member of a library out

```
//STEP1      EXEC  PGM=TTFUIN1
//TTFLOG      DD   SYSOUT=*
//TTFCMDS     DD   *
//    ifile cp=819,path=/u/scomsto/utility/source/...,
//    recdelim=crlf,filedata=t
//TTFOUT      DD   DISP=OLD,DSN=library(member)
```

- ◆ Note the output has DISP=OLD: the library exists and is being updated by adding or replacing a member
- ◆ Also, in this case, the input is an ascii file with CRLF delimiter, probably a file from a Windows system uploaded in binary
- ◆ Finally note there is no TTFIN DD statement, since the parameters are supplied on the IFILE command

Messages and Codes

Abend Codes

Return Codes

Messages

Abend Codes

We have worked hard to prevent File RePackager from abending ("blowing up"), but some things are out of our control. So here is our general advice:

System Abends (Sxxx in job listing): These abends are likely due to errors in your JCL or utility control statements coding, so check your JCL and utility control statements carefully and look at the utility log file; if you can't resolve the problem, contact The Trainer's Friend by phone or email to talk about solving the issue.

Phone: 303-393-8716

Email: steve@trainersfriend.com

User Abends (Udddd in job listing): all user abends are generated by I/O errors against the various files:

U230 - I/O error on commands file

U231 - I/O error on log file

U232 - I/O error on data input file

U233 - I/O error on data output file

In the above cases, it's possible they were caused by non-hardware faults, so try to read from the files using some other program (IEBGENER or similar IBM utility) and, again, check your JCL for erroneous or conflicting parameters. Otherwise, talk to the people in your shop who are responsible for hardware error monitoring.

Return Codes

At the end of every run, the utility emits a single integer, a return code, that provides a high level piece of information on how the run went:

| <u>RC</u> | <u>Means</u> |
|-----------|--|
| 0 | No errors were detected, run was successful |
| 4 | A warning diagnostic was issued, but the utility was run using default or deduced values |
| 8 | An error was encountered that prevents running the actual copy / transformation; nonetheless, error checking continued on the input commands |
| 12 | An error occurred that prevented further processing |
| 16 | Unable to open the log file; the application terminates |
| 17 | Error closing the log file |
| 1000 | Unable to open the commands file |
| 1001 | Unable to open the input file |
| 2001 | Unable to open an HFS input file |
| 1002 | Unable to open the output file |
| 1010 | Unable to set FILEDATA=RECORD for output file |
| 1020 | Error closing the commands file |
| 1021 | Error closing the input file |
| 1022 | Error closing the output file |

Note that open errors are usually due to incorrectly coded DD statements.

TTFE007S Unable to obtain CPU id.

The callable service CSRSI failed. This can happen on an older system; the system is not supported by the File RePackager.

TTFE008W This license expires on *yyyy/mm/dd*; call The Trainer's Friend to update / renew.

This warning message starts appearing about 30 days before the product license expires.

TTFE009S This license expired on *yyyy/mm/dd*; call The Trainer's Friend to update / renew.

The license has expired. The product will not run until you update, renew, or upgrade your copy.

TTFU012S LRECL of commands file less than 80.

The TTFIN file must be a sequential file (including member of PDS or PDSE, or a z/OS UNIX file) with records 80 bytes or longer.

**TTFU014I * Input file: RECFM=xxx, LRECL= *nn,nnn*, BLKSIZE= *nn,nnn*
or z/OS UNIX file.**

The input data file has been successfully opened.

TTFU015S Input file RECFM VBS is not supported.

Spanned records are not supported for input

**TTFU016I * Output file: RECFM=FB, LRECL= *nn,nnn*, BLKSIZE= *nn,nnn*
or z/OS UNIX file.**

The output data file has been successfully opened.

TTFU017S Output file RECFM VBS is not supported.

Spanned records are not supported for output

TTFU019I Input commands:

Marks the beginning of command processing

TTFU020E * Unable to load file: TTFUCPS

This file is necessary for processing. Ensure it is in the STEPLIB for the step

TTFU021E * Unable to load file: TTFCDFLT

This file is necessary for processing. Ensure it is in the STEPLIB for the step

TTFU023E * Unable to load file: Tfffitt

This is the codepage translation file; the 'fff' is the base 36 id for the from file and the ttt represents the base 36 id for the to file. This file is necessary for processing. Ensure it is in the STEPLIB for the step

TTFU024E * Unable to load file: TTFPARSE.

This is the main parsing subroutine. This file is necessary for processing. Ensure it is in the STEPLIB for the step

TTFU028S Insufficient storage; increase REGION size.

or, from the Parsing routine:

TTFP028S Insufficient storage; increase REGION size.

An attempt to dynamically obtain storage failed.

TTFU031I : *__ content __*

Display utility command record image; blank lines and comments are not displayed

TTFU034W > Col. 1 must be blank or asterisk.

Neither commands nor continuations may begin in column 1

TTFU036W > Command name too long; ignored.

Command names are 15 characters or less in length

TTFU038W > No command found.

Should not see this, due to earlier checks. (See TTFU067W)

TTFU039W > No operand found.

After a command was found, the operand string was blank

TTFU041E Reached unexpected point: xxxxx;

contact Trainers Friend: 303-393-8716

(All on one line.) Possible bug in utility. Contact us.

TTFU042E > Operand string too long. Ignored.

More than 32756 bytes of operand string. Rest of process is canceled.

TTFU050E Errors found, application terminated.

TTFU050E 'E' level diagnostics encountered. Syntax checking completed.

Syntax checking found 'E' level errors (check the log), completed syntax checking, but did not run the utility.

TTFU051I Run characteristics:

At this point, the input commands have been gathered and parsed. Now, the utility reports on its understanding of the run request. This could be useful if you find unexpected results.

TTFU052I End of run characteristics.

This line delimits the end of the analysis phase.

TTFU053I Processing Started.

This line marks the beginning of the run phase.

TTFU054I Processing Ended.

This line indicates the run has completed.

TTFU055S Processing Canceled.

This message is issued instead of TTFU053I and TTFU054I when the analysis phase detects errors that indicate the run cannot be successful. Read the previous log messages.

TTFU058I * All input data records will be processed.

TTFU059I * Skip first *nnn* input data records.

TTFU060I * Stop after *nnn* data records written.

Based on OPTIONS STARTAFTER and STOPAFTER paramters

TTFU062E ==> Missing info for translate table ==> Tfffft

A codepage translation requires an entry in the table that summarizes all the translate tables. Someone may have added a codepage translation table without updating TRANINFO. See the chapter on codepage support in the Installation and Customization Guide.

**TTFU063I * Character strings will be translated from code
page *nnnnn* to code page *nnnnn***

or

TTFU064I * No code page translations will be done

Information about translation to be done in this run

TTFU065I Records read from input file: *nnnnnnnnnnnn*

TTFU066I Records written to output file: *nnnnnnnnnnnn*

Summary information.

TTFU067W > Unknown command; ignored.

The first word on a utility command record was not recognized.

TTFU070W >Multiple OPTIONS commands present.

First encountered command will be used.

TTFU071W >Multiple IFILE commands present.

First encountered command will be used.

TTFU072W >Multiple OFILE commands present.

First encountered command will be used.

Only one of each of these commands is allowed in any run.

TTFU076W Invalid character x'xx' found at location *nn* in record number *m*

TTFU077W Replaced by substitution character.

A character was found in the input data that cannot be translated into the code page of the output; the substitution character has replaced the untranslatable character in the output

Check that any cp= parameter you have specified is correct for the input file you are working with

TTFU078E Run stopped.

An error was encountered (such as an untranslatable character in message TTFU076W) and the OPTIONS command specified the run should be stopped when this error is found.

TTFU083S Severe error from parse routine.

Generated when return code from TTFPARSE is 12.

TTFU086W Utility run skipped due to options value.

Generated when RUN=NO coded on OPTIONS command

TTFU094I Output record larger than file LRECL.

TTFU095W Based on OPTIONS TRUNCATE value, a truncated record will be written.

TTFU093I Bad record is input record number: *n*

TTFU094I Output record larger than file LRECL.

TTFU096W Based on OPTIONS TRUNCATE value, the record will not be written out.

TTFU093I Bad record is input record number: *n*

TTFU094I Output record larger than file LRECL.

TTFU097E Based on OPTIONS TRUNCATE value, the run will be stopped.

TTFU093I Bad record is input record number: *n*

There is not enough room in the output record, even after truncating trailing blanks, to hold the generated data; the response depends on the setting of TRUNCATE= on the OPTIONS statement (default is to stop the run).

TTFU098S Output LRECL too small. Re-run with larger LRECL.

Increase the LRECL value for TTFOUT and re-run the job.

TTFU099E Input HFS record larger than buffer. Run stopped.

Reading from input record, and record size is larger than 128KiB. This is not supported

TTFU100W Output HFS record larger than prefix size allows. The record will not be written out.

The output file is a z/OS UNIX file with record prefixes; the size of the current record is larger than the prefix size can hold; for 1 byte prefix, output record size > 256; for a 2 byte prefix, output record size > 65536; for a 4 byte prefix the output record size is larger than the output buffer. The record is skipped and processing continues.

TTFP201W > Operand name too long.

An operand name longer than 15 characters was encountered; this can occur if an operand does not have an = sign after it.

TTFP202E > Operand namespace exhausted.

More than 24 operand names in a single statement.

TTFP203W > Unrecognized operand name: *nnnnnn*; ignored.

Operand name misspelled or unsupported.

TTFP204E > Operand value too long.

Value longer than allowed maximum for the operand or total size of all values larger than 1200 bytes.

TTFP206E > Unsupported CodePage requested: *nnnnn*

Check the list of supported code pages

TTFP207W > Encountered name with no value.

An operand was found on a command such as **name=**, or **name=** with a space or no value following the equals sign

TTFP209E > PATH not found, so FILEDATA, RECLLEN, RECDELIM, RECPREFIX not allowed.

The named parameters all relate to z/OS UNIX files

TTFP210W > Invalid RECPREFIX value (valid: 1,2,4); ignored.

Fix the value in RECPREFIX

TTFP211E > Invalid RECLLEN value: not positive integer.

Fix the value in RECLLEN

TTFP212W > Invalid FILEDATA value (valid: T,B,R)

Fix the value in FILEDATA

**TTFP213E >Invalid RECDELIM value (valid:CR,LF,CRLF,LFCR,NL).
Copy will not be run.**

Fix the value in RECDELIM

TTFP214W > Invalid value for V=. Only Y is valid; ignored.

Fix the value in V

TTFP215W > Multiple RECDELIM parameters; first used.

Include the value you really want

TTFP216W > Multiple RECLLEN parameters; first used.

Choose the value you want to use.

TTFP218W > Multiple RECPREFIX parameters; first used.

Specify the value you mean to use.

TTFP219W > RECPREFIX and RECDELIM are mutually exclusive; first used.

Specify the correct value

TTFP221W > Multiple PATH parameters; first used.

Only one PATH parameter per xFILE command.

TTFP222W > Multiple FILEDATA parameters; first used.

Specify the value you mean.

TTFP225E > FILEDATA=R conflicts with RECPREFIX.

FILEDATA=R implies a [hidden] leading prefix

**TTFP226E > FILEDATA=B requires at least one of RECLLEN, RECPREFIX,
RECDELIM.**

For binary files, the utility needs some way to determine the record boundaries

TTFP227E > FILEDATA=T conflicts with RECPREFIX.

Text files use delimiters to determine record boundaries.

TTFP228E > FILEDATA=T requires RECDELIM.

Text files use delimiters to determine record boundaries.

**TTFP229E > PATH requires at least one of RECLEN, RECPREFIX,
RECDELIM.**

This occurs if none of the above parameters are specified; the utility has no way of knowing how to determine record boundaries.

TTFP230W > PATH speciifed with no FILEDATA; FILEDATA set to {B|T}

Utility chooses FILEDATA if it is not specified and PATH is. T is chosen if RECDELIM is also specified, otherwise B is used.

TTFP231W > Invalid value for FILL character; space used.

Only N (for nulls), B (for blanks) or x'xx' (hex representation of a single character) are allowed.

TTFP232W > Multiple FILL parameters; first used.

Specify the value you mean to use.

TTFP235W > RUN specified incorrectly. RUN=NO assumed.

Specify RUN=NO or omit.

TTFP240W > Multiple STARTAFTER parameters; first used.

Specify the value you mean.

TTFP241W >Multiple STOPAFTER parameters; first used.

Specify the value you mean.

TTFP242E > Invalid STARTAFTER value: not positive integer.

Fix the value in STARTAFTER

TTFP243E > Invalid STOPAFTER value: not positive integer.

Fix the value in STOPAFTER

TTFP244W > > Invalid TRUNCATE option; TRUNCATE=P assumed.

Run should be OK, but change the value to eliminate the message

TTFU900I Final completion code: *nn*

Looking for 0 or at most 4.

This page intentionally left almost blank.