xpcfiles: Cross Platform File Copy Version 1

CML00029-01

Code Magus Limited (England reg. no. 4024745)
Number 6, 69 Woodstock Road
Oxford, OX2 6EY, United Kingdom
www.codemagus.com

December 15, 2020

Contents

1 Introduction

Program `xpcfiles` (`SRDXPCPY` on OS/390) copies files with multiple fixed format records suitable for processing on one platform, so that they are suitable for processing on other platforms. The files can contain records with multiple record formats, but the content of any particular record must be able to distinguish the layout of that record.

In order for `xpcfiles` to be able to copy a file and convert its records, certain configuration data is required. This configuration data is kept in a separate file and contains logic and references to copybooks. Within the configuration file there is a section for each record type. These sections, referred to as `layouts`, assign a mapping to the record by way of selecting 01-level items in a copybook together with an indication of which fields in the copybooks are actually present in each case. In order to know whether or not a particular `layout` applies to a record, a predicate is included in each `layout`. This predicate is restricted in that it only comprises of a conjunction of equalities and in-equalities in which fields are compared to literals.

Together with the layouts in the configuration file, there are `input` and `output` options which pertain to the source and target systems on which the file originates (`input`) and on which the the file will be processed (`output`). These platforms may be different to the platform on which the `xpcfiles` program ultimately executes.

2 Processing

The program processes files by converting a single input file into a single output file for each execution of the program. Together with the names of the input and output files, a configuration file is required. This configuration file contains the `layout` sections as well as any options required to describe the `input` and `output` platforms as well as any `global` options.

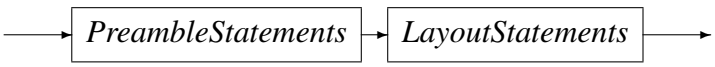
The configuration file is processed by parsing its contents and building the corresponding data structures. Any copybooks referenced in the layout statements are also parsed, once in the context of the `input` options and once in the context of the `output` options. The predicate is parsed into its abstract syntax tree and the literals are converted into the concrete representations implied by the `input` options and the types of the data items to which they are being compared (either for equality or in-equality). This preparation of the literals expedites record identification when the input file is processed.

3 Configuration

This section describes the configuration file's statement syntax and meaning.

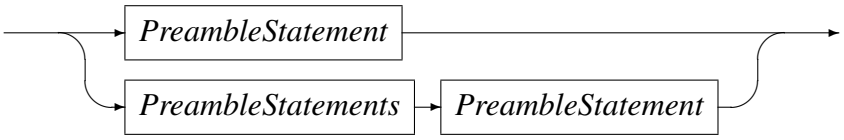
A configuration file contains the details of how to identify records in the file and the options for the input and output processing platforms.

CrossPlatformConfig



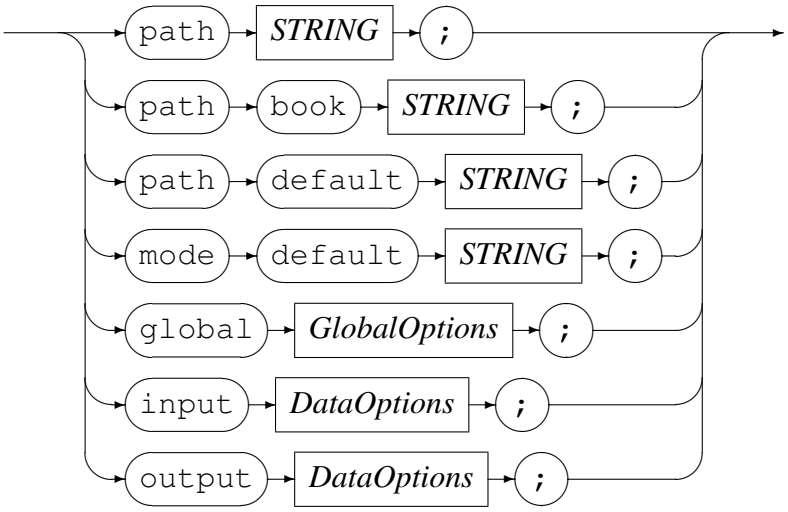
There are two parts to the configuration *CrossPlatformConfig*: *PreambleStatements* specify the various options; and the *LayoutStatements* describe various records and how they are identified.

PreambleStatements



The preamble comprises a list of *PreambleStatements*. A *PreambleStatement* has one of the following forms:

PreambleStatement



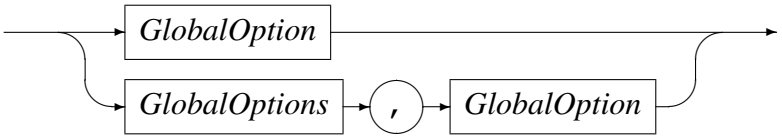
In addition to the various option statements, there are `path` statements and a `mode` statement in the preamble section of the configuration file.

The `path` and `path book` statement supplies a string which is used as a mask for supplying the full file name for the various copybooks that are referred to in the `layout` statement. This string value is expected to contain a `%s sprintf` style sequence where the supplied copybook name is edited into the string to form the full name of the copybook file.

The `path default` statement supplies a string, also expected to contain a `%s sprintf` style sequence where the file name of the `default` clause is used to form a full file name of the file containing the default record for that particular layout.

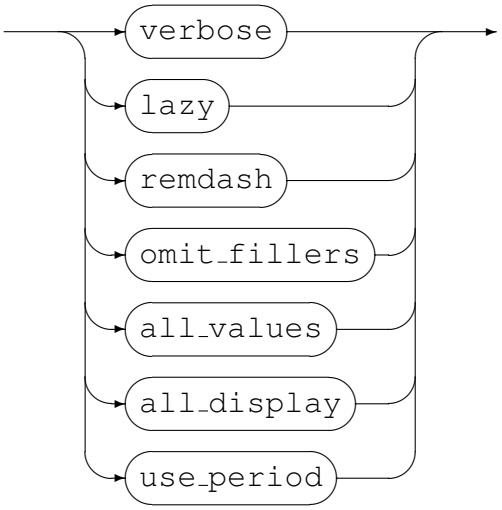
The `mode default` statement supplies the open mode string that will be used to open each of the default value files. This mode string has a default value of `rb,type=record` and does not have to be specified if this default value will satisfy the requirements.

GlobalOptions



The *GlobalOptions* supplies the list of global options used for processing the file.

GlobalOption



The `verbose` global option specifies that, for diagnostic purposes, additional messages will be produced when parsing and preparing the configuration file and the copybooks referred to in the configuration `layout` sections.

The `lazy` keyword specifies that the copybooks will only be loaded on demand. That is when the fields contained within them are referenced. It does not pertain to the evaluation of the `layout` statements themselves.

The `remdash` global option indicates that all dashes in the field names in the copybooks will be converted to underscore characters as the copybooks are parsed into symbols table entries. This means that all reference to field names in the `layout` statements, whether within the `include` or `exclude` statements, or within the predicate (i.e in the `when` clause), must contain underscore characters rather than the dashes.

The `omit_fillers` global option indicates that filler fields will not appear in the records regardless of whether or not the parents of the filler fields are explicitly included the layout.

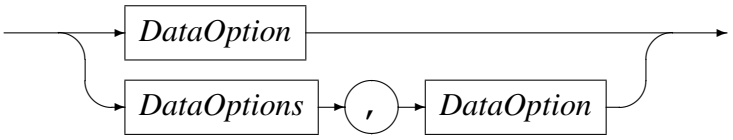
The `all_values` global option indicates that the alpha-numeric fields should be processed with all byte values being valid. The default behaviour without this option checks

each byte in the source field to make sure that it is a valid graphic character with respect to the code EBCDIC Latin/1 Open Systems IBM Code Page 1047, or the ISO-8 IBM Code Page 819 (depending on whether the `ebcdic` or `ascii` option is in effect). Without this option being in effect, if non-graphic characters appear in a field, then the field is not converted and keeps the same values as in the source field (with the exception of any required padding or truncating as determined by the respective field lengths). With the `all_values` option, the bytes of the field are not required to match the graphic characters in the corresponding code pages and all values are translated if required.

The `all_display` global option indicates that the alpha-numeric fields should be processed with all graphic byte values being valid, and that the non-graphic byte values should be translated to either the question mark character ("`?`") or the period character ("`.`"). The default is to use the question mark character unless the `use_period` option is also in effect.

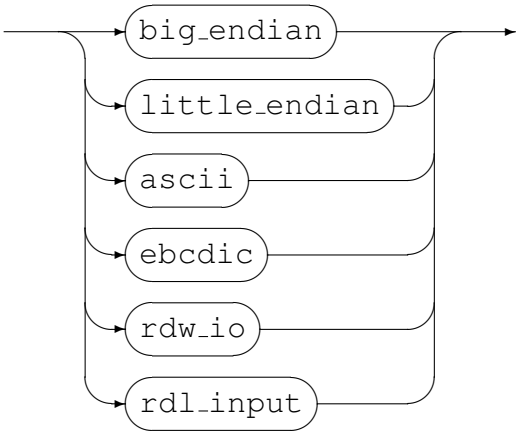
The same values apply to the `input` and `output` options. In the discussion here, these are grouped as *DataOptions*. The *DataOptions* appear as a list of comma-separated values, each being a `DataOption`.

DataOptions



The various values of the *DataOption* describe some facet of the `input` or `output` file's data. Some of the *DataOptions* are mutually exclusive.

DataOption



The options `big_endian` and `little_endian` are mutually exclusive in the same `input` or `output` statement. This is true even if more than one `input` or more than one `output` statement is used.

The `big_endian` option on the `input` statement indicates that binary data on the plat-

form suitable for processing the input file is represented in the big-endian format. That is, with the most significant byte of the binary value stored in the lowest byte address of the binary field. The `little_endian` option on the `input` statement indicates that binary data on the platform suitable for processing the input file is represented in the little-endian format. That is, with the most significant byte of the binary value stored in the highest byte address of the binary field. Similarly, when the options `bigendian` and `littleendian` are used on an `output` statement, they describe the format of the binary representation of the platform intended to process the converted file.

The options `ascii` and `ebcdic` are mutually exclusive in the same `input` or `output` statement, even if more than one `input` or `output` statement is used.

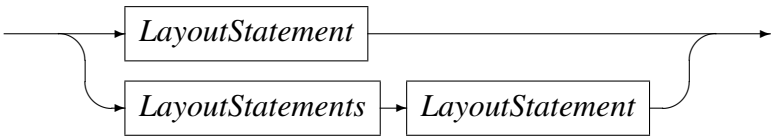
When used on the `input` (resp. `output`) statement, the `ascii` option indicates the character set collating sequence used to represent single byte display data on the platform on which the data is intended to be produced (resp. consumed) is based on the ASCII, or more accurately the ISO-8 character set with its graphics from ISO-8859-1.

Similarly, when used on the `input` (resp. `output`) statement, the `ebcdic` option indicates the character set collating sequence used to represent single byte display data on the platform on which the data is intended to be produced (resp. consumed) is based on the EBCDIC character set using code page 1047, or Latin 1/Open Systems.

The `rdw_io` option when used on the `input` statement (resp. `output` statement) indicates that the input file (resp. output file) is to be read (resp. written) using the `rdw_io` library. This method of reading and writing the file interprets the records of the file as though they were variable length records in which the records are prefixed with record descriptor words.

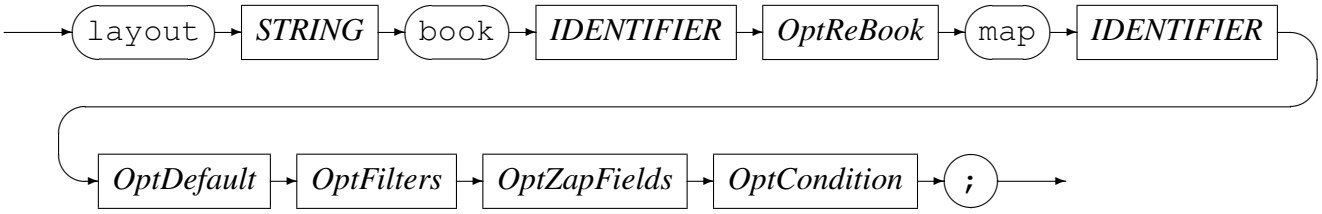
The `rdl_read` option when used on the `input` statement indicates that the input file record lengths are determined by the record contents themselves. In this case a read-ahead mechanism is used to read the data and from the input stream and once the correct layout has been determined, the stream is advanced by the length of the map associated with the layout. This is useful when the records have different lengths, RDWs are not used and either end-of-record markers (such as ASCII CR-LF sequences) cannot be trusted because of the presence of other binary data or they are also do not appear in the data.

LayoutStatements



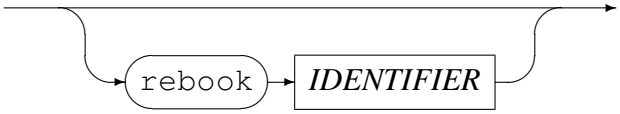
Following the preamble is a list of `layout` statements.

LayoutStatement



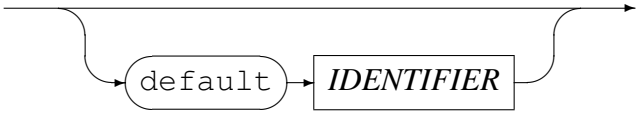
Each `layout` statement starts with the keyword `layout` and is terminated with the "`;`" character. A single `layout` statement describes one record type in the file to be copied. A record type is a single 01-level record item with a single combination of fields under it. That is, a layout in which there is no ambiguity regarding the layout of the fields of the record. The copybook containing this 01-level item is supplied by the `book` clause and the 01-level item within the book is nominated using the `map` clause.

OptReBook



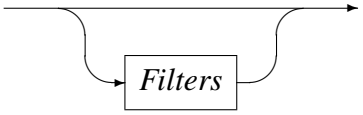
Optionally, the output layout of the record may be slightly different from the input layout of the record. In order to specify this, an alternate output book can be specified. The alternate output copybook is specified on the optional `rebook` clause.

Optdefault



Optionally, the output of a particular layout can be overlaid with the contents of a file. When the `default` clause is used, the contents of the first record of the indicated file are used to overlay the output buffer after the buffer has been initialised with initialisation image byte value, and before the input buffer fields have been converted into their output buffer values. The supplied identifier is edited into the value of the `path default` string to form the file name of the file to be opened.

OptFilters

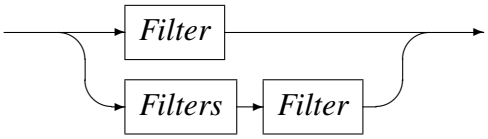


The *Filters* are also optional and when not specified, all the fields on the nominated 01-level item are assumed to be present in the record when the `layout` statement is applicable (except for any filler items when the `omit_fillers` option is in effect for

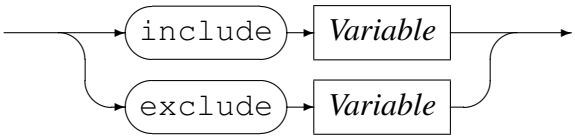
the input file).

When present, the *Fillers* modify the fields that are included as a result of selecting the `layout` statement and the corresponding 01-level item.

Filters



Filter

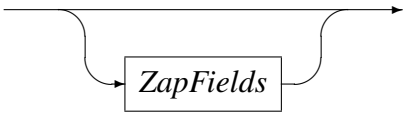


There can be any number of *Filter*s with each *Filter* clause starting with one of the keywords `include` or `exclude`.

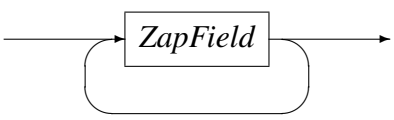
Filters are processed in the order in which they are listed in the `layout` statement. A *Filter* names a variable within the selected 01-level item using a dotted notation to fully qualify the item that should be included or excluded. For an `include` clause, the named variable is assumed to be included in the record described by the `layout` statement. The meaning of including a covering field is to include all elementary items under the covering field. For an `exclude` clause, the named variable is assumed to be excluded from the record described by the `layout` statement even if the field was explicitly included because of a prior `include` clause (directly or indirectly). And the same is true of included fields which had been explicitly excluded by a previous `exclude` clause (directly or indirectly).

The next section of the `layout` is an optional zap deck which must be applied to the input and output buffers if the current layout applies to that buffer.

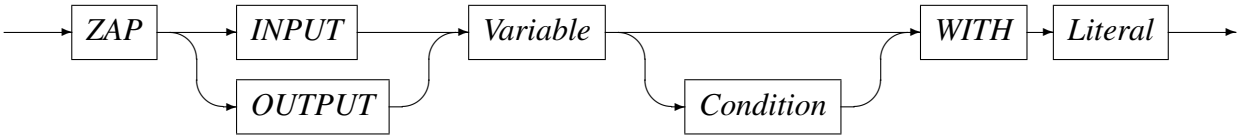
OptZapFields



ZapFields



ZapField



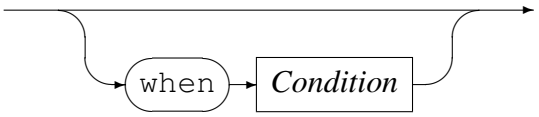
During processing of the input file, once a `layout` is determined to be applicable to an input record, and before any fields are moved from the input buffer to the output buffer, any applicable `INPUT` zaps are applied to the input buffer. The zaps are determined to be applicable if the predicate specified by the *Condition* evaluates to true over the input buffer. If the predicate is not present, then the update is applied to the input buffer unconditionally for the layout in which it is defined.

If there are any output buffer zaps, then these are applied to the output buffer once the fields from the input record have been moved to the output buffer.

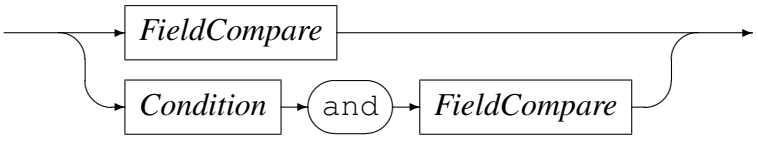
Following the *Filters* in the `layout` statement is an optional predicate which, when it evaluates to true, indicates that the layout is applicable to the current record. The predicate is introduced with the `when` clause and comprises of a single equality or in-equality or a conjunction of equalities and/or in-equalities. Each equality or in-equality must compare a *Variable* to a *Literal*.

If the predicate is not provided, then it behaves as though one is present which always evaluates to true. In order to provide a default layout without a predicate in the same configuration with other layouts which do contain `when` clauses make sure the default `layout` is the first layout in the configuration file.

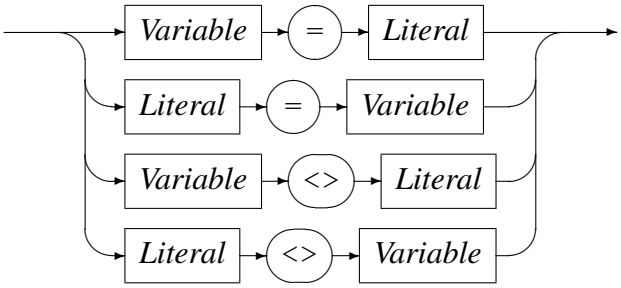
OptCondition



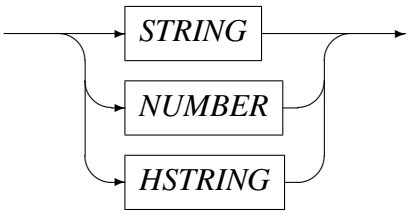
Condition



FieldCompare



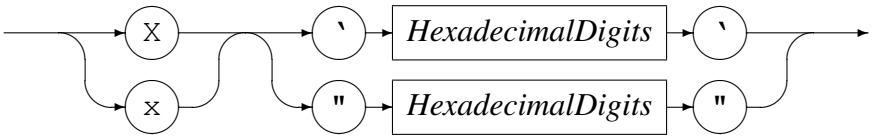
Literal



A *Literal* may be a string, number or a hexadecimal string. A string is a sequence of printable characters enclosed in apostrophes or quotation marks. A string delimited by apostrophes cannot contain apostrophes and a string delimited by quotation marks cannot contain quotation marks. If such string values are required, or if non-printable characters are to be included in the string, then the hexadecimal format of the string must be used.

A hexadecimal string is represented by a literal of the form:

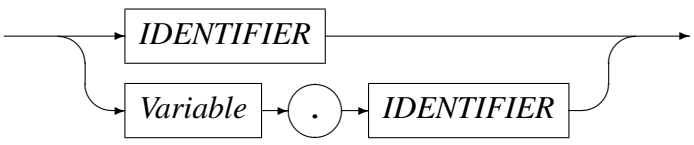
HSTRING



Where *HexadecimalDigits* is a sequence of an even number of *HexadecimalDigit* digits: `0` ... `9`, `A` or `a` ... `F` or `f`.

A number is a sequence of digits preceded by an optional "+" or "−" sign and with an optional decimal point. At least one digit is required and if the decimal point is used, then at least one digit must precede the decimal point and at least one digit must follow the decimal point.

Variable



Wherever a *Variable* can appear, it must fully qualify the item starting from the 01-level

under which the *Variable* appears. If the *Variable* is an indexed item, then the full set of index values required to index the item must be present. An index to an item is specified as a comma-separated list of integers in parentheses which must follow the *Variable* name.

4 Processing Binary Data in Alpha-numeric Fields

Not all alpha-numeric items contain data that is required to be graphic or to contain graphic characters. Such items might contain a bit map, graphic image, or special values that indicate the same condition independently of the encoding scheme of the hosting platform. An example of the latter is a trailer record where the key or record type value uses high values to indicate a trailer record. There are two means of indicating that such a field should be excluded from translation. The first method is an extension of the underlying COBOL grammar, this augments the type of the field to indicate that item does not have an exposed collating sequence and uses the phrase `COLLATING IS HIDDEN` as shown in the following example:

```
01 TRAILER-RECORD.
   05  TRL-RECORD-TYPE    PIC X(02) COLLATING IS HIDDEN.
   05  FILLER             PIC X(392).
```

However, this method makes the copybook unsuitable for a COBOL compiler, and so an alternative method of applying the attribute to the type of field is to place a specially formatted comment in the copybook. The downside of this is that should there be a syntax error in the comment, then it cannot be detected as the comment may be interpreted as a normal comment. The specific structured comment is one that starts with `attribute set` as in the following example:

```
01 TRAILER-RECORD.
   05  TRL-RECORD-TYPE    PIC X(02).
*attribute set HDR-RECORD-TYPE["TRANSLATE"] = "NO".
   05  FILLER             PIC X(392).
```

5 Execution

On UNIX and Windows based systems the command runs under a shell with the command being entered either in a script file or at the command prompt. On OS/390 (where the program is called `SRDXPCPY`) the command can be executed as a batch program or under `IKJEFT01` as a command processor. Regardless of where the command is executed, the program is invoked in a similar manner and with the same command line arguments available.

- `-c, --config-file=<config-file>` Name of layout configuration file (mandatory).
- `-i, --input-file=<input-file>` Name of the input fixed format file (mandatory).
- `-j, --input-mode=<input-mode>` Mode to use to open input file.
- `-o, --output-file=<output-file>` Name of the output fixed format file (mandatory).
- `-k, --output-mode=<output-mode>` Mode to use to open output file.
- `-n, --input-use-rdwio` Use the RDW I/O library for reading the input file.
- `-e, --input-use-rdlread` Use the RDLREAD library for reading the input file.
- `-u, --output-use-rdwio` Use the RDW I/O library for writing the output file.
- `-l, --record-length=<record-length>` Record length to assumed for files.
- `-v, --verbose` Verbose printing during processing.
- `-?, --help` Show a brief help message.
- `--usage` Display brief usage message.

6 Example—Coverting Data From an ASCII Base to an EBCDIC Base

This section demonstrate a simple example. Below is a configuration file which can be used to copy files suitable for processing on a little endian binary representation, ASCII based system to files suitable for processing on a big endian EBCDIC based system. The file contains two record types ("`Sample Record A`" and "`Sample Record B`"). The two different records are distinguishable by the value of the record type fields (`A-RECORD-TYPE` of record `A-SAMPLE-RECORD` and `B-RECORD-TYPE` of `B--SAMPLE-RECORD`).

```
# File: asc2ebc.conf
#
# This file is the configuration file for testing the
# xpcfiles program.
#
path "/home/stephen/xpcfiles.test/%s.cbb";
```

```
global lazy, verbose, omit_fillers;
input little_endian, ascii;
output big_endian, ebcdic;

layout "Sample Record A" book sample map A-SAMPLE-RECORD
   include A-SAMPLE-RECORD
      when A-SAMPLE-RECORD.A-RECORD-TYPE = "A";

layout "Sample Record B" book sample map B-SAMPLE-RECORD
   include B-SAMPLE-RECORD
      when B-SAMPLE-RECORD.B-RECORD-TYPE = "B";
```

In this example, the copybook referred to would be called `sample.cbb` and would have the fully qualified path name of:

`/home/stephen/xpcfiles.test/sample.cbb`

and could look something like:

```
      * File: sample.cbb
      *
      * This is a sample copy book for testing the xpcfiles program.
      *
       01  A-SAMPLE-RECORD.
           03 A-RECORD-TYPE    PIC X VALUE "A".
           03 A-ALPHA-FIELD-1  PIC X(20).
           03 A-NUMERIC-FIELD-2 PIC S9(5).
           03 A-NUMERIC-FIELD-3 PIC S9(5) COMP-3.
           03 A-NUMERIC-FIELD-4 PIC S9(4) COMP.
           03 A-NUMERIC-FIELD-5 PIC S9(5) COMP.
           03 A-NUMERIC-FIELD-6 PIC S9(8) COMP.
      *
      01   B-SAMPLE-RECORD.
           03 B-RECORD-TYPE    PIC X VALUE "B".
           03 B-NUMERIC-FIELD-1 PIC S9(8) COMP.
           03 B-NUMERIC-FIELD-2 PIC S9(4) COMP.
           03 B-NUMERIC-FIELD-3 PIC S9(5) COMP-3.
           03 B-NUMERIC-FIELD-4 PIC S9(5).
           03 B-NUMERIC-FIELD-5 PIC S9(5) COMP.
           03 B-ALPHA-FIELD-6  PIC X(20).
```

The following illustrates the usage of this command using the above configuration file:

```
xpcfiles --config-file=asc2ebc.conf \
   --input-file=sample_ascii_in.bin \
   --output-file=sample_ebcdic_out.bin
```

7 Example—Conditionally Zapping Fields in the Input File

Assume a file described by copybook `TESTBOOK.cpy`:

```
01  input_map.
    03  field1 pic x.
    03  field2 pic x.
    03  field3 pic s9(7) comp-3.
```

In this file of 6 byte records, some of the values of field-3 are not valid:

```
      00..__..__05..__..__10..__..__15..__..__20..__..__25..__..__30..
 0000: 414100000000C41410000001C41410000002C41420000003C41420000004C4142
 0000: A.A.........A.A.........A.A.......,.A.B.......<.A.B.......L<A.B.
 0020: 4141414141410000005C41410000006C41410000007C
 0032: A.A.A.A.A.A.......\*A.A.......l%A.A.......|@
```

This causes formatting errors (`Error unpacking value for FIELD3`) when trying to interpret the file using the copybook `TEXTBOOK.cpy` as shown in this following output from the `cmlprint` tool (see record 6):

```
Start of File = example.dat. Using object types = testtype.objtypes
Seq = 1, File = example.dat
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "A"
      03 FIELD3 = 0

Seq = 2, File = example.dat
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "A"
      03 FIELD3 = 1

Seq = 3, File = example.dat
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "A"
      03 FIELD3 = 2

Seq = 4, File = example.dat
Type = layout
```

```
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "B"
      03 FIELD3 = 3

Seq = 5, File = example.dat
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "B"
      03 FIELD3 = 4

Seq = 6, File = example.dat
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "B"
Error unpacking value for FIELD3.
      03 FIELD3 = X"41414141"

Seq = 7, File = example.dat
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "A"
      03 FIELD3 = 5

Seq = 8, File = example.dat
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "A"
      03 FIELD3 = 6

Seq = 9, File = example.dat
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "A"
      03 FIELD3 = 7
```

Ordinarily, this would also cause `xpcfiles` processing to terminate with a suitable error message. Ordinarily, this is normal and desirable behaviour.

```
Cross Platform Copy.
Copyright (c) 2000-2003 by Stephen Donaldson. All rights reserved.
Copyright (c) 2001-2004 by Code Magus Limited. All rights reserved.

mailto:stephen@codemagus.com, http://www.codemagus.com
Build: Apr 24 2008, 16:18:01

path "./%s.cpy";

global lazy;
input ascii, little_endian;
output ebcdic, big_endian;

layout "Sample Layout"
   book TESTBOOK map input_map
   include input_map
   when input_map.field1 = 'A';

Error unpacking value for FIELD3.
Error converting buffer for record number = 6.
Offset of error = 2.

Current input buffer:

        00..__..__05..__..__10..__..__15..__..__20..__..__25..__..__30..
  0000: 414241414141
  0000: A.B.A.A.A.A.

Output buffer so far:

        00..__..__05..__..__10..__..__15..__..__20..__..__25..__..__30..
  0000: C1C20000004C
  0000: .A.B......L<

Terminating conversion.
```

There are situations where values do not form part of the field domain and which are not considered errors. Such situations occur, for example, in DB2 extracts where the missing field value is indicated by the presence of a specific value in another field (typically, the null indicator). In the example above we use a zap to make the value of `field3` value before the conversion process, and in this example the condition under which this occurs is if `field2` has the value of '`B`':

```
Cross Platform Copy.
Copyright (c) 2000-2003 by Stephen Donaldson. All rights reserved.
Copyright (c) 2001-2004 by Code Magus Limited. All rights reserved.

mailto:stephen@codemagus.com, http://www.codemagus.com
```

```
Build: Apr 24 2008, 16:18:01

path "./%s.cpy";

global lazy;
input ascii, little_endian;
output ebcdic, big_endian;

layout "Sample Layout"
   book TESTBOOK map input_map
   include input_map
   zap input input_map.field3 when input_map.field2 = 'B' with -1
   when input_map.field1 = 'A';
```

The conversion now no longer values on invalid `field3` values, and the converted file also formats without errors. Notice in the file, that wherever `field2` has a value of 'B', `field3` has a value of -1:

```
Start of File = binary(outfile.dat,recfm=f,reclen=6,mode=rb).
Using object types = testtype.objtypes.

Seq = 1, Length = 6
File = binary(outfile.dat,recfm=f,reclen=6,mode=rb)
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "A"
      03 FIELD3 = 0

Seq = 2, Length = 6
File = binary(outfile.dat,recfm=f,reclen=6,mode=rb)
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "A"
      03 FIELD3 = 1

Seq = 3, Length = 6
File = binary(outfile.dat,recfm=f,reclen=6,mode=rb)
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "A"
      03 FIELD3 = 2

Seq = 4, Length = 6
```

```
File = binary(outfile.dat,recfm=f,reclen=6,mode=rb)
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "B"
      03 FIELD3 = -1

Seq = 5, Length = 6
File = binary(outfile.dat,recfm=f,reclen=6,mode=rb)
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "B"
      03 FIELD3 = -1

Seq = 6, Length = 6
File = binary(outfile.dat,recfm=f,reclen=6,mode=rb)
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "B"
      03 FIELD3 = -1

Seq = 7, Length = 6
File = binary(outfile.dat,recfm=f,reclen=6,mode=rb)
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "A"
      03 FIELD3 = 5

Seq = 8, Length = 6
File = binary(outfile.dat,recfm=f,reclen=6,mode=rb)
Type = layout
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "A"
      03 FIELD3 = 6

Seq = 9, Length = 6
File = binary(outfile.dat,recfm=f,reclen=6,mode=rb)
Type = layout
```

```
Title = Test Book Layout

   01 INPUT_MAP
      03 FIELD1 = "A"
      03 FIELD2 = "A"
      03 FIELD3 = 7
```