
PPREP 1.0 - Reference Manual



by *Wolfgang Lief*

Last updated: 2 September 2009

PPREP

Abstract

This manual was automatically generated from the source code of version 1.0.240 (20090902) of the PPREP (Portable Pre-Processor) script file pre-processing and execution system.

PPREP is written in portable Object Pascal and should be working on any platform where a suitable Pascal compiler is available (e.g. FreePascal (<http://www.freepascal.org>) or Delphi).

Copyright

Copyright © 2007-2009
Airborne Research Australia/Wolfgang Lief
Hangar 60, Parafield Airport, South Australia

As this document is generated automatically, the (re-)usage conditions of the underlying source code apply. Please refer to the header of the main source file of the fpx library 'libfpx.pas' for more information.

Contents

1	History and Rationale	5
2	PPREP as a standalone program	5
3	PPREP as a library procedure	6
4	Commands and Features	6
4.1	Intrinsic Operations	6
4.1.1	'{ }' - Variable Reference	6
4.1.2	Conditional Operators	6
4.2	Functions	7
4.2.1	\$_CALL - PPREP Call Line	7
4.2.2	\$_COPY - Substring Copy	7
4.2.3	\$_DATE - Timestamp Function	8
4.2.4	\$_DDS - Fix Up Directory Separators	8
4.2.5	\$_EVAL - Numerical Evaluation	8
4.2.6	\$_EXIST - File Existence Function	8
4.2.7	\$_EXITCODE - Command Line Exit Code Function	9
4.2.8	\$_EXT - File Name Extension	9
4.2.9	\$_FILEDATE - File Timestamp Function	9
4.2.10	\$_FILELIST - Directory Listing Function	9
4.2.11	\$_FILEREAD - Read Table Cell(s) From a Text File	9
4.2.12	\$_FSIZE - File Size Function	10
4.2.13	\$_HMS - Convert Time in Seconds to HMS	10
4.2.14	\$_HOST - Host Name	10
4.2.15	\$_INCLUDE - Lines From a Text File	10
4.2.16	\$_LCUT - Cut Left of Marker	10
4.2.17	\$_LOCASE - Lowercase Function	11
4.2.18	\$_LOGIC - Logical Evaluation	11
4.2.19	\$_MASK - Apply Mask to Expression	11
4.2.20	\$_MATCH - Select a 'Best Match' from File	11
4.2.21	\$_NSET - Variable Check	12
4.2.22	\$_NUMBER - Number Formatting Function	12
4.2.23	\$_OS - Operating System	12
4.2.24	\$_PATH - PPREP Executable Path	12
4.2.25	\$_RANGE - Range Generator Function	12

4.2.26	\$@RCUT - Cut Right of Marker	13
4.2.27	\$@SEC - Convert Time in HMS to Seconds	13
4.2.28	\$@SELECT - Selector Function	13
4.2.29	\$@SORT - Sort Strings	13
4.2.30	\$@SUBST - Substitute String With Replacement	13
4.2.31	\$@TABLE - Table Access Function	14
4.2.32	\$@UPCASE - Uppercase Function	14
4.2.33	\$@UTM - Convert WGS84 Coordinates to UTM	14
4.2.34	\$@WGS - Convert UTM Coordinates to WGS84	14
4.2.35	\$@XMLREAD - Read a Value from an XML File	14
4.3	Directives	15
4.3.1	\$ALIAS - Variable Definition	15
4.3.2	\$BRACKETS - Define Bracket Characters	15
4.3.3	\$C - Literal Copy	15
4.3.4	\$CLEAR - Clear Target Buffer	16
4.3.5	\$CLEART - Clear Table(s)	16
4.3.6	\$DELETE - File Deletion	16
4.3.7	\$DUMP - Dump Target Buffer to File	16
4.3.8	\$END - Terminate Processing	16
4.3.9	\$ERROR - Error Termination	16
4.3.10	\$EXECUTE - Execute External Program	17
4.3.11	\$FILTER - Define Line Filter	17
4.3.12	\$INCLUDE - Include External File	17
4.3.13	\$IF - Conditional Structure	17
4.3.14	\$KEEPBLANK - Define Action for Blank Lines	18
4.3.15	\$LOOP - Cyclic Structure	18
4.3.16	\$MAXRECURSE - Maximum Recursion Level	19
4.3.17	\$PATH - Check and Prepare Path	19
4.3.18	\$PREFIX - Define Command Prefix Character	19
4.3.19	\$RENAME - File Renaming/Moving	19
4.3.20	\$SELECT - Alternatives Structure	20
4.3.21	\$SET - Variable Assignment	20
4.3.22	\$STARTT - Start Table Definition	20
4.3.23	\$SUBSTITUTE - Variable Substitution	21
4.3.24	\$UNSET - Variable Deletion	21

5	Appendix	21
5.1	Examples	21
5.2	Changelog	22

1 History and Rationale

PPREP is an extended/improved Object Pascal re-write from scratch of the well utilized and proven preprocessors 'Aprep' (Fortran 90) and its predecessor 'prep' (Fortran 77), which was developed by the research team around Dr J. Hacker at Flinders University, Adelaide during the 1980s and 90s.

The decision to re-implement the system in a different language was made to ensure ongoing maintenance and portability of the code, while at the same time incorporating additional features not available (or not easily available on all platforms) in the Fortran environment. The choice of FreePascal as the underlying programming language make PPREP scripting available to a wide range of platforms, including, apart from mainstream end-user systems like Linux, MacOS, and Windows, quite a number of more exotic and/or embedded systems e.g. MorphOS, WinCE, or good old DOS.

While a couple of limitations of its predecessors have been relaxed in PPREP (e.g. no limits on string lengths), there is one major structural change which affects end-user scripts:

prep and Aprep where 'block-oriented' preprocessors which treated a text file as a single block and performed all operations they were directed to do in the whole file, e.g. '\$SET x y' would change all occurrences of '{x}' in the file into 'y', without exception. Only after all statements were processed, it would be checked if there are any unsatisfied references left (and terminated if there were). As a side-effect, unresolved references in inactive (e.g. non-taken '\$IF' branches) could lead to error termination.

PPREP on the other hand strictly is a 'stream-oriented' processor which processes the input file as a stream of lines one after the other. The consequences are:

- any command on any given line can only have effects 'downstream', i.e. on lines still pending to be processed
- any references not resolvable with the 'knowledge' of the program when it encounters a line lead to error termination
- erroneous statements in inactive parts of the script will not be evaluated and thus not lead to error termination

2 PPREP as a standalone program

The program 'pprep.pas' is a driver for the fpxPrep library. It pre-loads the input string list from a file, calls the 'PreProcess' procedure and writes the output into a file, or optionally to the console.

Syntax:

```
pprep {input file} {output file}
```

Parameters:

{input file} (filename) – Name of the input file
{output file} (filename) – Name of the output file ('-' redirects output to the console)

Note:

While it is fully legal for input and output to be the same file, this is not recommended.

3 PPREP as a library procedure

The main functionality of PPREP is realized as the Object Pascal unit 'fpXPrep.pas' which makes the procedure interface

```
'procedure PreProcess(var aStringList:TStringList);'
```

available to any Pascal programs using the standard 'Classes' library (e.g. any Object Pascal program written in FreePascal, Delphi, or MacPascal). On invocation the TStringList object 'aStringList' should contain the text lines to be processed. On return, 'aStringList' contains the final state of the target buffer after PPREP has finished (i.e. if no \$CLEAR commands have been executed, the full extent of the preprocessed text).

Currently a PPREP module is incorporated as an integral part of the *rasp* airborne sensor processing package.

4 Commands and Features

The following list is a (hopefully) complete, alphabetically sorted (within their context groups) list of the available commands and features.

4.1 Intrinsic Operations

4.1.1 '{ }' - Variable Reference

Syntax:

```
{name}
```

Description:

Preprocessor variables are referenced by their name enclosed in '{ }' brackets. Variable reference brackets can be nested and will be evaluated from the innermost outwards at processing time. For example, the statement

```
{name{number}}
```

will first evaluate the variable 'number', and assuming that its value is '42', then the variable 'name42'.

Note:

There is no limit to nesting depth and in conjunction with the \$ALIAS command direct or indirect endless recursion is possible. This is normally limited to 64 levels through the default setting of the \$MAXRECURSE command, which needs to be increased to allow higher nesting depth.

In order to process files which contain curly brackets '{ }', the reserved characters used to delineate variable references can be re-defined using the '\$BRACKETS' command.

4.1.2 Conditional Operators

Syntax:

```
{Operand_1} {Operator} {Operand_2}
```

Description:

The logical values 'true' and 'false' are represented in PPREP as the single characters 'Y' and 'N', respectively. A set of conditional operators allows generation, combination, and modification of such values.

Multiple operators can be combined on a single line, provided that proper bracketing (using '(')') brackets) is used to resolve any ambiguities of which operand belongs to which operator.

The following evaluators are available:

Evaluator	Description
<code>\$EQ</code>	Evaluates to 'Y' if Op1 and Op2 are identical
<code>\$NE</code>	Evaluates to 'Y' if Op1 and Op2 are different
<code>\$IN</code>	Evaluates to 'Y' if Op1 is a space-delimited substring of Op2
<code>\$OR</code>	Evaluates to 'Y' if Op1 or Op2 are 'Y'
<code>\$AND</code>	Evaluates to 'Y' if Op1 and Op2 are 'Y'
<code>\$XOR</code>	Evaluates to 'Y' if only one of Op1 or Op2 is 'Y'
<code>\$NOT</code>	Evaluates to the logical inversion of Op2
<code>\$GT</code>	Evaluates to 'Y' if Op1 is lexically greater than Op2
<code>\$LT</code>	Evaluates to 'Y' if Op1 is lexically smaller than Op2
<code>\$NUL</code>	Evaluates to 'Y' if Op2 is blank/empty

Note:

When using the default date format ('YYMMDD_hhmm'), lexical comparison is equivalent to temporal comparison.

Note:

In order to process files which contain the dollar character '\$', the reserved character used as a function/command prefix can be re-defined using the '\$PREFIX' command.

4.2 Functions

PPREP functions are statements of the form '\$@{name}({expression})'. When a function is being processed, the whole statement is replaced by the function result string (i.e. a direct string/string replacement). Functions can be nested and are evaluated from the innermost set of brackets outwards. Unlike directives (see below), functions only have no effects outside the line they are on.

Note:

In order to process files which contain the dollar character '\$', the reserved character used as a function/command prefix can be re-defined using the '\$PREFIX' command.

4.2.1 \$@CALL - PPREP Call Line

Syntax:

`$@CALL()`

Description:

Returns the call line used to invoke PPREP (e.g. this way PPREP can find out if it was called via the PPREP front-end or as an implicit part of another program. Furthermore it makes it possible to pass parameters from the command line to the script).

4.2.2 \$@COPY - Substring Copy

Syntax:

`$@COPY({Expression},{Start},{Length})`

Description:

Returns a sub-string of `{Expression}` with `{Length}` characters beginning at the `{Start}`th character. the `{Length}` parameter is optional and defaults to all remaining characters of the string. If `{Start}` is a negative number, the start character is the `{Start}`th to the left from the end of the string.

4.2.3 `$@DATE` - Timestamp Function

Syntax:

```
$@DATE({Time Format})
```

or

```
$@D({Time Format})
```

Description:

Returns a time stamp string of the current time. If nothing or `'*` is specified as the format, the default format `'YYMMDD_hhmm'` is used.

4.2.4 `$@DDS` - Fix Up Directory Separators

Syntax:

```
$@DDS({Filename})
```

Description:

Fixes up the directory separators in `{Filename}` to be valid on the system PPRE is currently running on (i.e. use `'/'` on Linux and `'\'` on Windows). If `{Filename}` is already correct, this function will do nothing.

4.2.5 `$@EVAL` - Numerical Evaluation

Syntax:

```
$@EVAL('{Numerical Expression}')
```

Description:

Evaluates a numerical expression and returns a single number as the result. Apart from the basic operations `'+,-,*,/'` `$@EVAL` will interpret brackets, the exponentiation operator `'^'` and a number of predefined functions (currently implemented are `ABS`, `SQRT`, `SQR`, `FRAC`, `TRUNC`, `ROUND`, `LOG10`, `TAN`, `SIN`, `COS`, `ATAN`, `ASIN`, `ACOS`, `LN`, `EXP`, and the constant `PI`).

Note: If `{Numerical Expression}` contains any brackets (either stand-alone, or as part of a function) it needs to be enclosed in quotes in order to separate the PPREP function brackets from the numerical brackets.

Note: The arguments and/or results of trigonometric functions are assumed to be in degrees.

4.2.6 `$@EXIST` - File Existence Function

Syntax:

```
$@EXIST({Filename})
```

or

```
$@E({Filename})
```

Description:

Returns 'Y' if the file {Filename} exists, 'N' otherwise.

4.2.7 `$_EXITCODE` - Command Line Exit Code Function**Syntax:**

```
$_EXITCODE()
```

Description:

Returns the exit code (integer number) of the last command executed through the '`$_EXECUTE`' command.

4.2.8 `$_EXT` - File Name Extension**Syntax:**

```
$_EXT({Filename})
```

Description:

Returns the extension part of a filename (i.e. the characters after the rightmost '.')

4.2.9 `$_FILEDATE` - File Timestamp Function**Syntax:**

```
$_FILEDATE({Filename},{Time Format})
```

or

```
$_FD({Filename},{Time Format})
```

Description:

Returns a time stamp string of the last modification time of a file. If nothing or '*' is specified as the format, the default format 'YYMMDD_hhmm' is used.

4.2.10 `$_FILELIST` - Directory Listing Function**Syntax:**

```
$_FILELIST({Path})
```

or

```
$_FL({Path})
```

Description:

Returns a space separated list of file names satisfying the {Path} specification (Wildcard expression, e.g. '*.las', can be used)

4.2.11 `$_FILEREAD` - Read Table Cell(s) From a Text File**Syntax:**

```
$_FILEREAD({filename},{column},{row},{Delimiter})
```

Description:

Returns the a specified element from a text file containing columns/rows of entries. `{column}`, `{row}`, `{Delimiter}` are optional parameters describing the structure of the file (default is space-delimited). If `{column}` or `{row}` are omitted, or set to zero, the respective whole row or column is returned (if both are zero, the whole file is returned as a single line).

4.2.12 `$_FILESIZE` - File Size Function

Syntax:

`$_FILESIZE({Filename})`

Description:

Returns the size of the file `{Filename}` in bytes, or '0' if it does not exist.

4.2.13 `$_HMS` - Convert Time in Seconds to HMS

Syntax:

`$_HMS({Seconds})`

Description:

Returns the corresponding 6-digit hour/minute/second time string.

4.2.14 `$_HOST` - Host Name

Syntax:

`$_HOST()`

Description:

Returns the host name (if set) of the computer PPREP is currently running on.

4.2.15 `$_INCLUDE` - Lines From a Text File

Syntax:

`$_INCLUDE({filename},{start_line},{nlines})`

Description:

Reads `{nlines}` lines from the text file `{filename}`, beginning with line `{start_line}` and returns them as a space-separated list of quoted strings. The parameters `{nlines}` and `{start_line}` are optional and default to 'until the end of the file' and 'from the beginning of the file', respectively.

Note:

Blank lines in `{filename}` are ignored (i.e. not counted and not returned).

4.2.16 `$_LCUT` - Cut Left of Marker

Syntax:

`$_LCUT({Marker},{Expression})`

Description:

Returns the part of `{Expression}` which is to the left of the leftmost occurrence of `{Marker}`.

4.2.17 `$@LOCASE` - Lowercase Function**Syntax:**

```
$@LOCASE({String})
```

Description:

Replaces a string with its lower case equivalent.

4.2.18 `$@LOGIC` - Logical Evaluation**Syntax:**

```
$@Logic({Statement})
```

or

```
$@L({Statement})
```

Description:

Returns 'Y' if {Statement} is true, 'N' otherwise.

4.2.19 `$@MASK` - Apply Mask to Expression**Syntax:**

```
$@MASK({Mask},{Expression})
```

Description:

Returns {Expression} with all occurrences of {Mask} removed.

4.2.20 `$@MATCH` - Select a 'Best Match' from File**Syntax:**

```
$@MATCH({Filename},{Match_Start},{Match_End},{Return},{Expression},{Delimiter})
```

Description:

Returns the column {Return} from the line of {Filename} which best matches the {Expression}. {Match_Start} and {Match_End} describe which columns should be matched against, and which mode should be used. {Delimiter} is an optional parameter (the default is space-delimited) describing the character used to delineate the columns in {filename}.

The following cases are possible:

{Match_End}-{Match_Start}	Elements in {Expression}	Match Type
0	1 (numeric)	Closest
0	1 (non-numeric)	Exact
1	1 (numeric)	Most Central 1D
3	2 (numeric)	Most Central 2D

Example

```
$@MATCH(dem_config.txt,3,6,1,138.0 -35)
```

will perform a look-up in 'dem_config.txt' and return the name of the DEM tile in which the coordinates 138/-35 are located most centrally.

```
$@MATCH(photo_times.txt,2,2,1,52456)
```

will return the name of the photo taken closest to GPS time 52456.

4.2.21 `$@NSET` - Variable Check**Syntax:**`$@NSET({name})`**Description:**

Returns 'Y' if {name} is undefined, 'N' otherwise.

4.2.22 `$@NUMBER` - Number Formatting Function**Syntax:**`$@NUMBER({Expression},{nDigits})`

or

`$@N({Expression},{nDigits})`**Description:**

Converts {Expression} into an integer number (non-numeric values evaluate to zero) and formats that into a string of {nDigits} digits, cutting off high digits or padding with leading zeroes if required.

4.2.23 `$@OS` - Operating System**Syntax:**`$@OS()`**Description:**

Returns the operating system ID of the computer PPREP is currently running on (Possible values are 'LINUX', 'WIN', and 'MAC').

4.2.24 `$@PATH` - PPREP Executable Path**Syntax:**`$@PATH()`**Description:**

Returns the fully qualified path of the currently running executable, including the trailing delimiter (and on Windows systems the leading drive letter).

4.2.25 `$@RANGE` - Range Generator Function**Syntax:**`$@RANGE({From},{To},{Step})`

or

`$@RANGE({From},{To})`

or

`$@R({From},{To},{Step})`

or

`$@R({From},{To})`**Description:**

Generates a space-delimited list of numerical indices between `{From}` and `{To}`, using the optional step width `{Step}` (Default = 1). The sign of the step will be automatically adjusted for sequence direction.

4.2.26 `$@RCUT` - Cut Right of Marker

Syntax:

```
$@RCUT({Marker},{Expression})
```

Description:

Returns the part of `{Expression}` which is to the right of the leftmost occurrence of `{Marker}`.

4.2.27 `$@SEC` - Convert Time in HMS to Seconds

Syntax:

```
$@SEC({HMS})
```

Description:

Returns the corresponding second count for a given 6-digit hour/minute/second time string.

4.2.28 `$@SELECT` - Selector Function

Syntax:

```
$@SELECT({Item_1} {Item_2} ... {Item_n},{Index})
```

or

```
$@S({Item_1} {Item_2} ... {Item_n},{Index})
```

Description:

The selector function evaluates to the n-th space-delimited sub-field as designated by `{Index}`.

4.2.29 `$@SORT` - Sort Strings

Syntax:

```
$@SORT({Expression})
```

Description:

Returns `{Expression}` with all space-delimited substrings sorted in alphabetical order.

4.2.30 `$@SUBST` - Substitute String With Replacement

Syntax:

```
$@SUBST({mask},{replacement},{expression})
```

Description:

Returns `{expression}` with all occurrences of `{mask}` replaced by `{replacement}`.

4.2.31 `$$TABLE` - Table Access Function**Syntax:**`$$TABLE({TableName},{RowName},{ColName})`

or

`$$T({TabelName},{RowName},{ColName})`**Description:**

This function returns an element identified by a table name, and row and column labels. If the requested row is not found, the row labeled 'def' (if present) is selected instead.

4.2.32 `$$UPCASE` - Uppercase Function**Syntax:**`$$UPCASE({String})`

or

`$$U({String})`**Description:**

Replaces a string with its upper case equivalent.

4.2.33 `$$UTM` - Convert WGS84 Coordinates to UTM**Syntax:**`$$UTM({Latitude} {Longitude})`**Description:**

Returns the corresponding UTM coordinates in the form of `{Easting}` `{Northing}` `{Zone}`.

4.2.34 `$$WGS` - Convert UTM Coordinates to WGS84**Syntax:**`$$WGS({Easting} {Northing} {Zone})`**Description:**

Returns the corresponding WGS84 coordinates in the form of `{Latitude}` `{Longitude}`. The `{Zone}` parameter is optional - if omitted or specified as '0', the current default zone is used. 'S' or 'N' postfixes to the zone can be specified to designate southern or northern hemisphere, respectively. If omitted, southern hemisphere UTM conventions will be used.

4.2.35 `$$XMLREAD` - Read a Value from an XML File**Syntax:**`$$XMLREAD({filename},{key_1},...,{key_n})`**Description:**

Returns a 'value' field from an XML file. The requested field is located by a sequence of one or more keys, i.e. the file is advanced in sequence to

the respective first occurrence of `{key_1}`, `{key_2}`, up to `{key_n}`. The closest following string enclosed in inverse angle brackets `'>{result}<'` is returned.

Note:

The keys `{key_i}` are case sensitive and an exact match is required.

4.3 Directives

PPREP directives are statements of the form `'${name} {parameter_list}'`. Directives control the PPREP control flow and usually have effects for all subsequent lines (but unlike in PPREP's predecessor 'prep', there are no backward influences). Except in conditional statements, there can only be one directive stated on each input line.

Note:

In order to process files which contain the dollar character '\$', the reserved character used as a function/command prefix can be re-defined using the '\$PREFIX' command.

4.3.1 \$ALIAS - Variable Definition

Syntax:

```
$ALIAS {Statement}
```

Description:

Assign a value to a variable before any variable and function references have been resolved, i.e. they will not be resolved at the location of \$ALIAS statement, but when they are referenced.

4.3.2 \$BRACKETS - Define Bracket Characters

Syntax:

```
$BRACKETS {Expression}
```

Description:

The two character string `{Expression}` defines alternative variable identifiers to be used in place of the standard 'curly brackets'.

Note:

The characters used for left and right bracket need to be different and non-blank. An empty `{Expression}` resets the bracket definition to 'curly brackets'.

4.3.3 \$C - Literal Copy

Syntax:

```
$C {Statement}
```

Description:

Copy `{statement}` to the output without any modification.

4.3.4 `$CLEAR` - Clear Target Buffer**Syntax:**`$CLEAR`**Description:**

Clear the contents of the target buffer (i.e. all output lines generated so far), while keeping all variable references generated by `$SET` commands and `$LOOP` structures.

4.3.5 `$CLEART` - Clear Table(s)**Syntax:**`$CLEART {Table Name}`**Description:**

Will remove the named table from memory (i.e. so that it can be re-assigned). If no name is given, all tables will be cleared.

4.3.6 `$DELETE` - File Deletion**Syntax:**`$DELETE {Name}`**Description:**

Deletes a named file from disk.

4.3.7 `$DUMP` - Dump Target Buffer to File**Syntax:**`$DUMP {Filename}`**Description:**

Dump the current contents of the target buffer to a file (i.e. all lines which have been generated since the start of `PPREP` or the last `$CLEAR` command).

4.3.8 `$END` - Terminate Processing**Syntax:**`$END`**Description:**

Terminates processing and discards the rest of the file.

4.3.9 `$ERROR` - Error Termination**Syntax:**`$ERROR {Message}`**Description:**

Force error termination with the specified message.

4.3.10 `$EXECUTE` - Execute External Program**Syntax:**

```
$EXECUTE {CommandLine}
```

Description:

Submits a command line to the operating system for execution in the same processing thread as PPREP (i.e. PPREP will be suspended until the external command has finished).

Note:

As resolution of the command line depends entirely on the configuration of the underlying operating system and command shell, no assumptions about the default path or other environmental constraints of the called program should be made, i.e. preferably the full pathname of an executable should be used in order to avoid ambiguities (e.g. `'c:\windows\notepad.exe'` instead of just `'notepad'`).

4.3.11 `$FILTER` - Define Line Filter**Syntax:**

```
$FILTER {Expression}
```

Description:

After the line filter has been engaged with the `$FILTER` command, PPREP will ignore all subsequent lines not starting with the character sequence `{Expression}`. Lines beginning with `{Expression}` will be processed as normal (with `{Expression}` removed). An empty string as `{Expression}` will reset the filter to process all lines again.

Example: Filtering manual LaTeX code from Pascal source

```
$FILTER // // $INCLUDE fpxPres.pas // $FILTER
```

4.3.12 `$INCLUDE` - Include External File**Syntax:**

```
$INCLUDE {Filename} {Name_1} {Name_2} ... {Name_n}
```

Description:

Includes an external file and optionally makes a name list available to a `$SUBSTITUTE` command.

4.3.13 `$IF` - Conditional Structure**Syntax:**

```
$IF {Condition} $THEN {Statement}
```

or

```
$IF {Condition} $THEN {Statement} $ELSE {Statement}
```

or

```
$IF {Condition} $THEN
```

```
    {Statements}
```

```
$ENDIF
```

or

```
$IF {Condition} $THEN
    {Statements}
$ELSE
    {Statements}
$ENDIF
```

Description:

If {Condition} evaluates to 'Y', the first alternative of the conditional structure is selected. For any other value, the second alternative (i.e. the part after the \$ELSE statement) will be processed. If any further characters follow after the \$THEN, the \$IF statement is assumed to be a 'single-line if', i.e. no \$ENDIF is necessary. When the condition line terminates after the \$THEN, the statement is considered to be the beginning of a 'block if', which needs to be terminated by an \$ENDIF statement.

Note:

There is no limit to nesting depth of blocked if structures. PPREP will not check for illegal intersections between '\$LOOP', '\$SELECT', and '\$IF' structures (e.g. a loop starting inside a conditional construct, but ending outside), and will produce unpredictable output should they occur.

4.3.14 \$KEEPBLANK - Define Action for Blank Lines

Syntax:

```
$KEEPBLANK {Expression}
```

Description:

If {Expression} is 'Y', PPREP will not drop blank lines from the output.

4.3.15 \$LOOP - Cyclic Structure

Syntax:

```
$LOOP {Name_1} {Value_1_1} {Value_1_2} ... {Value_1_n}
$& {Name_2} {Value_2_1} {Value_2_2} ... {Value_2_n}
.
    {further synchronized loop variables}
.
$& {Name_m} {Value_m_1} {Value_m_2} ... {Value_m_n}
$INDEX {IndexName}
.
    {further statements (including nested loops)}
.
$ENDLOOP
```

Description:

A loop structure is unrolled by assigning a sequence of values to a variable. The '\$&' command can be used to have additional variables unrolling in parallel to the main loop variable. The '\$INDEX' command assign the string representation of the current loop index at the current nesting level to a variable.

Note:

There is no limit to loop nesting depth. PPREP will not check for illegal intersections between '\$LOOP', '\$SELECT', and '\$IF' structures (e.g. a loop starting inside a conditional construct, but ending outside), and will produce unpredictable output should they occur.

4.3.16 \$MAXRECURSE - Maximum Recursion Level

Syntax:

```
$MAXRECURSE {Level}
```

Description:

Allows to set the 'Emergency Brake' maximum allowed recursion level in variable/function definitions to a different value from the default of 64. While excessive recursion/nesting of loops, conditional structures, and include files would sooner or later lead to an 'out of memory' termination, variable and function expansion could lead to infinite recursion (e.g. \$ALIAS endless_loop {endless_loop} would hang up the program as soon as it encounters the reference {endless_loop}. With the default setting PPREP will abort all processing after expanding variable and function references for a given line for 64 times and still not resolving everything.

4.3.17 \$PATH - Check and Prepare Path

Syntax:

```
$PATH {Path}
```

Description:

Checks if {Path} designates a valid directory, and if it does not exist, creates it (creating all higher path elements as necessary in the process).

4.3.18 \$PREFIX - Define Command Prefix Character

Syntax:

```
$PREFIX {Expression}
```

Description:

The single character string {Expression} defines an alternative prefix character to be used in place of the standard '\$' sign.

Note:

An empty {Expression} resets the prefix definition to '\$'.

4.3.19 \$RENAME - File Renaming/Moving

Syntax:

```
$RENAME {OldName} {NewName}
```

Description:

Rename a file on disk. By specifying full pathnames, this command can also move files to a different directory, as long as the target directory is on the same logical partition as the origin.

4.3.20 \$SELECT - Alternatives Structure

Syntax:

```
$SELECT {Selector}
$CASE {Option_1}
    {Statements}
$CASE {Option_2}
    {Statements}
$CASE {...}
    {Statements}
$CASE {Option_n}
    {Statements}
$CASE *
    {Statements}
$ENDSELECT
```

Description:

The {Selector} is checked against the various options {Option_n}. If a match is found, the respective statement block will be evaluated. If no match is found and a \$CASE * statement is present, its statement block will be evaluated.

Note:

There is no limit to nesting depth of \$SELECT structures. PPREP will not check for illegal intersections between '\$LOOP', '\$SELECT', and '\$IF' structures (e.g. a loop starting inside a conditional construct, but ending outside), and will produce unpredictable output should they occur.

4.3.21 \$SET - Variable Assignment

Syntax:

```
$SET {Name} {Value}
```

Description:

Assign a value to a variable after all variable and function references have been resolved (use \$ALIAS if such references need to be included in the variable).

4.3.22 \$STARTT - Start Table Definition

Syntax:

```
$STARTT {Table Name}
    {ColName_1} {ColName_2} ... {ColName_n}
```

```
'def'      {Default_1} {Default_2} ... {Default_n}
{RowName_1} {Cell_1_1} {Cell_1_2} ... {Cell_1_n}
{RowName_2} {Cell_2_1} {Cell_2_2} ... {Cell_2_n}
.
.
{RowName_k} {Cell_k_1} {Cell_k_2} ... {Cell_k_n}
$ENDT
```

Description:

Define a named table. The first line of the table definition has to be a space-delimited list of column names. Subsequent lines have to start with the row name and then a space-delimited list of cell values. The row name 'def' can be used to define default values which will be selected by the table look-up subroutine when no appropriate row can be found.

Note:

Cell values are stored in the table before variable references and functions are resolved, i.e. any references will be resolved when the cell values are accessed (as a side effect, any errors in individual cells will not manifest themselves unless they are accessed).

4.3.23 \$SUBSTITUTE - Variable Substitution

Syntax:

```
$SUBSTITUTE {Name_1} {Name_2} ... {Name_n}
```

Description:

Replaces local symbols in the current include file with those in a substitution list made available by the current \$INCLUDE command. If the list contains more entries than are to be substituted, the additional values are silently ignored. If the list contains fewer entries than the \$SUBSTITUTE command requires, the missing entries are not substituted but used literally. This behaviour allows easy implementation of default settings in include files which can optionally be overridden by the caller.

4.3.24 \$UNSET - Variable Deletion

Syntax:

```
$UNSET {Name}
```

Description:

Removes the entry for a named variable so that any subsequent reference to it will result in error termination. The main usefulness of this command is during program development and debugging in order to exclude unwanted side-effects from 'residual' definitions. 'Unsetting' of non-existing variables is allowed.

5 Appendix

5.1 Examples

Examples and reference output for all PPREP functions and directives can be found in the 'examples' subdirectory of the source tree.

5.2 Changelog

- 20090901 WL - New function `$@XMLREAD`
- 20090831 WL - New function `$@SEC`
 - New function `$@HMS`
- 20090826 WL - New function `$@INCLUDE`
 - Reorganized source code/manual alphabetically
 - New function `$@COPY`
- 20090824 WL - New function `$@MATCH`
 - New function `$@WGS`
 - New function `$@UTM`
- 20090819 WL - New directive `$@FILTER`
 - New directive `$@BRACKETS`
 - New directive `$@PREFIX`
 - New directive `$@KEEPBLANK`
 - New function `$@SORT`
 - Changed manual generation to use PPREP instead of specialized doc-filter program
- 20090818 WL - New function `$@FILEREAD`
 - New function `$@LCUT`
 - New function `$@RCUT`
 - re-wrote some structural elements in function evaluation to enable a shared codebase with *rasp*