T-Rex Overview

Support Licence Objects available Table of awk equivalents

This is T-Rex version 1.00.

T-Rex is an independently developed Delphi component for parsing textfiles.

It gives you the full power of awk's pattern-action style parsing, slotted into Delphi's event-handler paradigm.

If you've never used awk before, don't worry! We won't assume any prior knowledge on your part.

If you have used awk before, you'll know how quick and easy it is. With T-Rex, pattern recognition simply becomes an event, and you write the corresponding action in an event handler.

Using T-Rex is really easy.

Step 1: Create a `T_Rex` nonvisual component on your form.

Step 2: Point it at the textfile you want to parse, using `Filespec`.

Step 3: Specify the patterns that you are looking for in `MatchPattern`, using the same standard regular expressions that you're used to in the Delphi IDE (and if you need a tutorial on them, relax!, it's included in this helpfile).

Step 4: Write a handler for the `OnMatch` event.

Step 5: Tell T-Rex to `Scan` your file.

That's it!

This program is produced by a member of the Association of Shareware Professionals (ASP). If you are unable to resolve a shareware-related problem with an ASP member, ASP may be able to help. Click on the ASP logo for details.

This is $Revision:: 1.4 $ of this helpfile.

Objects Available in T-Rex

T-Rex offers a single nonvisual component,  TRex, which scans a specified input file, parses it for matches against the regular expressions you specify, and triggers events when matches are found.

The regular expressions are stored in TRex's MatchPattern property, which is an object of type TRexList. The TRexList object is available for your use whether or not you use the TRex component. A TRexList is a modified TStrings object. The regular expressions that you specify live in the Strings property. When you add a regular expression, the object compiles the string representation into a finite state recognizer for it, and stores it in the corresponding Objects property of TRexList.
The object that is stored is of type TRegExp. You can create your own objects of type TRegExp if you want to perform your own match processing on a single regular expression.

Compilation of a regular expression to a finite state recognizer is comparatively slow. However, T-Rex does this once only for each regular expression.

Recognition of a match by a compiled finite state recognizer is very fast indeed.

If you specify match patterns at design time, they are compiled as the TRex component is created. If you specify them at runtime, they are compiled when you call MatchPattern.Add or MatchPattern.Assign. If you create a TRegExp object, the regular expressions are compiled at create time.

Finite State Recognizer

A recognizer for a pattern *P* is a program that takes as input a string *s* and answers "yes" if *s* matches *P* and "no" otherwise. A finite-state recognizer works in the same way as the familiar railroad syntax diagram found in language reference manuals.

The awk language

Acknowledgements

Computer users spend a lot of time doing simple, mechanical data manipulation: changing the format of data, checking its validity, finding items with some property, adding up numbers, and the like.

Awk is a programming language, named for its creators, Alfred Aho, Peter Weinberger and Brian Kernighan, that makes it possible to handle such tasks with very short programs, often only one or two lines long. An awk program is a sequence of patterns and actions that tell what to look for in the input data and what to do when it's found. Awk searches for lines matched by any of the patterns; when a matching line is found, the corresponding action is performed. A pattern can select lines by combinations of regular expressions and normal string and numeric comparisons. Actions may perform arbitrary processing on selected lines; the action language looks like C, but there are no declarations.

Awk's brevity of expression makes it valuable for prototyping larger programs. One starts with a few lines, then refines the program until it does the desired job, experimenting with designs by trying alternatives quickly.

In principle, it's straightforward to translate an awk program into another language such as Delphi when the design is right. In practice, though, some of awk's nicest features are tough to implement in Delphi. T-Rex is designed to ease this process.

Adapted from the preface to *The awk progamming language*.

Regular Expressions

Quick Survey Examples Tutorial adapted from *The awk progamming language*.

If you've used DOS wildcards, then you're already familiar with the concept of a regular expression: it's a notation for matching and specifying strings.

Unfortunately, DOS wildcards are not all that powerful. There's no way, using DOS's ***** and **?**, to write an expression that will match all and only valid Delphi identifiers, or valid floating-point constants. Using regular expressions, you can do all of this and more.

The regular expression metacharacters are

\ . ^ $ [ ] | ( ) * + ?

They are called metacharacters because they have special meanings. (DOS wildcards have two metacharacters, ***** and **?**. Both have different meanings in regular expressions.)

A regular expression consisting of a single nonmetacharacter matches itself. Thus, a single letter or digit (for example **A**) is a basic regular expression that matches the one-character string 'A'.

Period . and backslash \

In a regular expression, a period **.** matches any single character. The backslash is the quoting character: it turns of the special meaning of the metacharacter. The backslash has a second meaning: it allows you to specify common non-printing characters such as tab and carriage return in a way that is easy to see in the Property Editor.

Examples...

Anchor metacharacters ^ and $

In a regular expression, a caret **^** matches the beginning of a string, a dollar-sign **$** matches the end of a string. These metacharacters are called anchors because they "anchor" the pattern to one or other end of the string to be matched.

Examples...

Character Classes

A regular expression consisting of a group of characters enclosed in brackets is called a character class; it matches any one of the enclosed characters. For example, **[AEIOU]** matches any of the characters A E I O or U.

More...

Examples...

Parentheses ( )

Parentheses are used in a regular expression to specify how components are grouped, much as they are in arithmetic expressions.

Alternation

The alternation operator **|** is used to specify alternatives: if *r* and *s* are two regular expressions, then *r***|***s* matches any string matched by *r* or by *s*.

Examples...

Concatenation

There is no explicit concatenation operator. If *r* and *s* are regular expressions, then *rs* matches any string of the form *xy* where *x* matches *r* and *y* matches *s*. The expressions *r* and *s* needs to be in parentheses if they have alternation operators inside them, because concatenation binds tighter than alternation.

Examples...

Repetition

The symbols ***** **+** and **?** are used to specify repetitions in regular expressions. If *r* is a regular expression, then

*r** matches any string consisting of zero or more consecutive substrings matched by *r*,

r+ matches any string consisting of one or more consecutive substrings matched by *r*,

r? matches the null string, or any string matched by *r*.

The expression *r* needs to be in parentheses if there is an alternation operator inside *r*, because repetition binds tighter than alternation.

Examples...

Precedence

The alternation operator **|** has the lowest precedence, then concatenation, and finally the repetition operators * + and ?. As with arithmetic expressions, operations of higher precedence are done before lower ones. These conventions often allow parentheses to be omitted: **ab|cd** is the same as **(ab)|(cd)** and **^ab|cd*e$** is the same as **(^ab)|(cd*e$)**.

Period and Backslash Examples

`...`	matches any three consecutive characters
`\.`	matches a period
`........\....`	matches a DOS filename of exactly 8.3 characters; for example `datafile.txt` but not `data.txt` nor `datafile.db`.

The backslash **** in the second and third example is the quoting character. It is there to preserve the literal meaning of the period, which would otherwise be taken as the metacharacter **.** which matches any character.

The period corresponds to the DOS wildcard **?**.

Escapes

Non-printing characters can be specified using the backslash as shown below:

`\b`	backspace (ASCII 8)
`\f`	formfeed (ASCII 12)
`\n`	linefeed (ASCII 10)
`\r`	carriage return (ASCII 13)
`\t`	horizontal tab (ASCII 9)

In DOS textfiles, a newline is indicated by a carriage return, optionally followed by a linefeed. To capture this, use `\r\n?`,

Anchor Metacharacters ^ and $

Here are some examples of the use of **^** and **$**:

`^C`	matches a C at the beginning of a string
`C$`	matches a C at the end of a string
`^C$`	matches the string consisting of the single character C
`^.$`	matches any string containing exactly one character
`^...$`	matches any string containing exactly three characters
`\.$`	matches a period at the end of a string

Character Classes

Ranges

Ranges of characters can be abbreaviated in a character class by using a hyphen. The character immediately to the left of the hyphen defines the beginning of the range; the character immediately to the right defines the end. Thus, **[0-9]** matches any digit, and **[a-zA-Z][0-9]** matches a letter followed by a digit. If it appears first or last, a hyphen in a character class denotes itself, so the character classes **[-+]** and **[+-]** match either a + or a -.

Complemented Classes

A complemented character class is one in which the first character after the **[** is a **^**. Such a class matches any character not in the group following the caret. Thus, **[^0-9]** matches any character except a digit; **[^a-zA-Z]** matches any character except an upper or lower-case letter.

Character Class Examples

Examples

`^[ABC]`	matches an A, B or C at the beginning of a string
`^[^ABC]`	any character at the beginning of a string, except A, B or C
`[^ABC]`	matches any character other than A, B or C
`^[^a-z]$`	matches any single-character string, except a lower-case letter

Inside a character class, all characters have their literal meaning, except for the quoting character \, **^** at the beginning, and **-** between two characters. Thus **[.]** matches a period (and so is an alternative notation to **\.**) and **^[^^]** matches any character except a caret at the beginning of a string.

Alternation and Concatenation

The regular expression

```
(Asian|European|North American) (male|female) (black|blue)bird
```

matches twelve strings ranging from *Asian male blackbird* to *North American female bluebird.*

Repetition

Here are some examples of the use of ***** **+** and **?**

`B*`	matches the null string or B or BB, and so on
`AB*C`	matches AC or ABC or ABBC, and so on
`AB+C`	matches ABC or ABBC or ABBBC, and so on
`ABB*C`	also matches ABC or ABBC or ABBBC, and so on
`AB?C`	matches AC or ABC
`[A-Z]+`	matches any string of one or more upper-case letters
`(AB)+C`	matches ABC or ABABC or ABABABC, and so on

The regular expression **.***, meaning zero or more repetitions of any character, corresponds to the DOS wildcard *****.

The Delphi IDE uses the notation `A{0-2}` to mean 0, 1 or 2 occurrences of A. T-Rex does not support this notation. Instead, use `A?A?`.

Regular Expression Examples

Here are some useful string-matching patterns:

This...	matches an input line that consists of...
`^[0-9]+$`	only digits
`^[0-9][0-9][0-9]$`	exactly three digits
`^(\+|-)?[0-9]+\.?[0-9]*$`	a decimal number with an optional sign and optional fraction
`^[-+]?[0-9]+[.]?[0-9]*$`	also a decimal number with an optional sign and optional fraction
`^[-+]?([0-9]+[.]?[0-9]*|[.][0-9]+)([Ee][-+]?[0-9]+)?$`	a floating-point number with an optional sign and an optional exponent
`^[A-Za-z]$\|^[A-Za-z][0-9]$`	a letter or a letter followed by a digit (variable name in old-fashioned Basic)
`^[A-Za-z][0-9]?$`	also a letter or a letter followed by a digit

Since + and . are metacharacters, they have to be preceded by backslashes in the third example to match literal occurrences. These backslashes are not needed within character classes, so the fourth example shows an alternate way to describe the same numbers.

Regular Expressions: Quick Survey

Metacharacter	Meaning
`.`	Any character
`\`	Quoting character: . matches any character \. matches period
`^`	Beginning of string
`$`	End of string
`[ ]`	Character class; [^A] means any other than A; [A-Za-z] means a range
`|`	Alternation: A|B matches A or B
`( )`	Grouping: (A|B)C matches AC, BC; A|BC matches A, BC
`*`	Zero or more occurrences: CA* matches C, CA, CAA etc
`+`	One or more occurrences: CA+ matches CA, CAA, CAAA etc
`?`	Zero or one occurrence: CA? matches C, CA
`{ }`	Not supported. (In the IDE it is used to specify a number of occurrences, for example `{2}` or `{0-2}`.) Instead of `A{0-2}` use `A? A?`.

Table of awk equivalents

If you've programmed in awk, then you may only need this table to get you started with T-Rex. Grey indicates features that are in Tawk that are not in standard awk.

Patterns	
BEGIN	Code before calling Scan
BEGINFILE	OnBOF event
END	Code after Scan returns
ENDFILE	OnEOF event
NR==1	OnBOF event
boolean expression	if-test in BeforeLineMatch event
/regular expression/	regular expression in MatchPattern, handled in OnMatch event
pattern && pattern	if-test in OnMultipleMatch event
pattern || pattern	if-test in OnMultipleMatch event
!pattern	flag in OnMultipleMatch event; if-test in AfterLineMatch event
pattern,pattern	Use a switch and an if-test in OnMatch event
Actions	
exit	SeekEof
next	SeekEoln
Input/Output	
close(FILENAME)	SeekEof
Built-in variables	
FILENAME	Filename
FNR	FileLineNumber
FS	InputSeparator
FPAT	TokenPattern
NF	TokenCount
NR	ScanLineNumber
OFS	OutputSeparator
RS	LineSeparator
RSTART	returned by TRegExp.Match
RLENGTH	returned by TRegExp.Match
$n	Token.Strings[n]
Built-in string functions	
match()	MatchImmediate
sub()	SubstImmediate
gsub()	SubstImmediate
split()	Place, or place another, TRex component to your form. Pass ScanText the string to be split. Access the elements of Token in an AfterLineMatch event handler.

TRex Component

See also Properties Methods Events Tasks

Unit

T_Rex

Description

TRex is a non-visual component that specifies a textfile to be parsed at the patterns that are to be searched for inside it.

When a pattern is matched, the component initiates an event that notifies the program of the match. The program's event-handler then performs whatever processing is required.

The Filespec property identifies the file (or files) to be scanned. For example, setting the filespec to `*.txt` will cause all of the files matching that pattern to be scanned in turn. You can also scan a block of text in memory, such as the contents of a TMemo control.

Specify the patterns to be searched for as regular expressions in the MatchPattern property.

The OnMatch event is the most usual event to handle. Your OnMatch event handler is called every time the current input line of the textfile matches one of the patterns in MatchPattern. The `Expression` parameter passed to the OnMatch handler tells your program which pattern was matched.

An alternative event to OnMatch is OnToken. The OnToken event handler is called every time TRex identifies a complete token in the input stream. You have of course complete control over what constitutes a token.

There are also OnBOF and OnEOF events which are primarily useful when scanning multiple files. They notify your program when a new file has been opened (which you might want to know about to reset counters, etc) and when the end of the file has been reached (which you might want to use to report progress or perform cleanup).

To initiate the scan process, call Scan or ScanText.

TRex provides a TRegExp object, which you may need to create if you wish to perform your own pattern matching.

In addition to these properties, methods, and events, this component also has the properties and methods that apply to all components.

Using the TRex Component

TRex Reference

Purpose

TRex  is a nonvisual component that automates the scanning of textfiles for patterns that you specify as regular expressions

To Specify the File to be Parsed

Either at design time or at runtime assign a value to Filespec. This can be either a filename or a wildcard specification. If it is a wildcard specification then each of the matching files will be scanned in turn. If you want to scan multiple files that cannot be captured with a single wildcard specification, assign Filespec and call Scan multiple times.

To Specify the Patterns to be Matched

Either at design time or at runtime, put one or more regular expressions in the MatchPattern property. Assigning a value that is not a valid regular expression will raise an EInvalidRegularExp exception.

To Specify the Action to be Taken on a Match

Write a handler for the OnMatch event.

To Control the Scanning Process

To begin the scan of the input file, call the Scan method. To abandon further processing of an input line once the OnMatch event handler has seen all it needs to see, call the SeekEoln method. To determine progress through the file, query the FileLineNumber property or compare the FilePosition property with the FileSize property. To abandon further processing of an entire file, call the SeekEof method.

To Scan a Block of Text in Memory

To begin the scan, call the ScanText method.

See Also

Overview of T-Rex

Properties

Run-time only

Key properties

FileLineNumber

Filename

FilePosition

FileSize

Filespec

InputLine

InputSeparator

LineSeparator

MatchPattern

MaxInputLine

OutputSeparator

ScanLineNumber

TokenCount

TokenPattern

Token

Methods

 Key methods

GetText

 Scan

 ScanText

 SeekEoln

SeekEof

Events

 Key events

- AfterLineMatch
- BeforeLineMatch
- OnBOF
- OnEOF
-  OnMatch
  - OnMultipleMatch
  -  OnToken

Filespec Property

TRex Reference

Declaration

```
property Filespec: TFilename;
```

Description

Identifies the file or files to be scanned. If more than one file matches the Filespec, then each file will be scanned in turn. The name of the file currently being scanned is available in the Filename property.

Filename Property

TRex Reference

Declaration

```
property Filename: TFilename;
```

Description

Runtime and read-only. The name of the file currently being scanned. This normally is only of interest if Filespec specifies more than one file.

Awk equivalent

FILENAME variable

FileSize Property

TRex Reference

Declaration

```
property FileSize: Longint;
```

Description

Runtime and read-only. The size, in bytes, of the file or other input stream currently being scanned. Normally used together with FilePosition to report the progress of a scan. Returns zero if no scan is active: beware of divide-by-zero errors!

Awk equivalent

None

FilePosition Property

TRex Reference

Declaration

```
property FilePosition: Longint;
```

Description

Runtime and read-only. The number of bytes of the file or other input stream that have already been processed. Normally used together with FileSize to report the progress of a scan. Returns zero if no scan is active. Returns zero in the OnBOF event. Returns FileSize in the OnEOF event.

Awk equivalent

None

InputLine Property

TRex Reference See also

Declaration

```
property InputLine: String;
```

Description

Runtime only. Gives the first 255 characters of the current input line. This will be the line as read from the input file, unless you have caused the input to be recomputed by assigning a new value to Token or Tokencount

Awk Equivalent

$0

See also

GetText

InputSeparator Property

Declaration

```
property InputSeparator: String;
```

Description

The default value of InputSeparator is '\w', which is shorthand for "whitespace". When InputSeparator has this specific value, input tokens or fields are separated by blanks and/or tabs, and leading blanks and tabs are discarded.

When InputSeparator has any other value, leading blanks and tabs are *not* discarded.

The way T-Rex splits a line into tokens can be changed by assigning a string to InputSeparator. This string is interpreted as a regular expression. The leftmost longest nonnull and nonoverlapping substrings matched by that regular expression become the token separators. For example,

```
InputSeparator := ',[ \t]*|[ \t]+'
```

makes every string consisting of a comma followed by blanks and tabs, and every string of blanks and tabs without a comma, into a token separator.

Because InputSeparator is interpreted as a regular expression, something indirect, such as `'[|]'` is needed to set it to a metacharacter. Because the sequence '\w' has a special meaning, something indirect, such as `'[\\]w'`, is also needed to set the input separator to this sequence.

Note that `\w` is not acceptable in a regular expression to mean whitespace. It is a special sequence used in this context only. To specify the equivalent in a regular expression, use `[ \t]+`.

Awk equivalent

`FS` variable. T-Rex differs in its choice of '\w' for the default value (because the awk value, a single space, is hard to see in the Property Editor). T-Rex also differs in that it treats one-character values as regular expressions.

See also
TokenPattern

LineSeparator Property

Declaration

```
property LineSeparator: String;
```

Description

The default value of InputSeparator is '\l', which is shorthand for "line separator". When LineSeparator has this specific value, one line of input is taken to end at a carriage return, optionally followed by a linefeed, that is, the regular expression `\r\n?`.

The way T-Rex splits a the file into lines can be changed by assigning a new value to LineSeparator. This string is interpreted as a regular expression.

Because InputSeparator is interpreted as a regular expression, something indirect, such as `'[|]'` is needed to set it to a metacharacter. Because the sequence '\l' has a special meaning, something indirect, such as `'[\\]l'`, is also needed to set the input separator to this sequence.

Note that `\l` is not acceptable in a regular expression to mean a carriage return optionally followed by a linefeed. It is a special sequence used in this context only. To specify the equivalent in a regular expression, use `\r\n?`.

Awk equivalent

RS variable. This implementation does not support setting LineSeparator to the null string. In awk this is taken to mean that records are separated by a blank line. Instead, use `\r\n? *\r\n?` (two successive carriage returns with optional linefeeds, possibly with spaces on the blank line).

See also
TokenPattern

MaxInputLine Property

TRex Reference

Declaration

```
property MaxInputLine: word;
```

Description

Specifies the maximum length of an input line, including the line terminator such as a CR/LF sequence. Cannot be changed while the file is being scanned. Any input line longer than MaxInputLine will be silently split. Must be a multiple of 1024 characters.

Awk equivalent

None: normally a fixed implementation-defined limit.

OutputSeparator Property

TRex Reference

Declaration

```
property OutputSeparator:string;
```

Description

T-Rex uses the value OutputSeparator to separate tokens when the InputLine is recomputed. This happens when a value is assigned to Token or TokenCount.

Awk Equivalent

OFS variable

MatchPattern Property

TRex Reference

Declaration

```
property MatchPattern:  TStrings;
```

Description

A list of strings containing regular expressions to be matched against. The first regular expression is numbered 0. When an input line contains one or more instances of a pattern that matches on of the patterns in the list, the OnMatch event occurs.

Awk Equivalent

Regular expression between / / characters in a pattern-action clause.

TokenCount Property

TRex Reference

Declaration

```
property TokenCount: word;
```

Description

Runtime only. Gives the number of tokens or fields in the current input line. Assigning a lower value causes then input line to be truncated to the specified number of tokens. Assigning a higher values causes new (null) tokens to be added to the end of the line.

Awk Equivalent

NF variable

TokenPattern Property

TRex Reference

Declaration

```
property TokenPattern: String;
```

Description

One way to identify the tokens in your input is to use InputSeparator to define what separates one token from another. That is what standard awk does. But sometimes it is easier to define a token by what it contains rather than by what it does not contain. Setting TokenPattern instead of InputSeparator allows you to define what pattern a token must match. The separators then become whatever does not match TokenPattern. Only one of the two properties may be set.

Awk equivalent

None in standard awk: FPAT variable in Tawk.

Token Property

TRex Reference

Declaration

```
property Token [index: integer]: String;
```

Description

As T-Rex parses a file, it splits every input line into tokens, using InputSeparator or TokenPattern to decide what constitutes a token. This array property gives you access to each of the tokens on the line.

The array is zero-based: the first token in the line is Token[0].

Assigning a new value to a token causes InputLine to be recomputed. You may assign a value to a nonexistent field. Suppose, for example, that there are currently 3 tokens on the input line (in other words, TokenCount = 3). Then executing

```
Token[5] := 'MyValue'
```

causes three new tokens to be created:

```
Token[3] = ''
Token[4] = ''
Token[5] = 'MyValue'
```

In the process, TokenCount will be recomputed and set to 6.

Awk equivalent

$1, $2, $3 ... $NF.

FileLineNumber Property

Declaration

```
property FileLineNumber: word;
```

Description

Runtime only. Incremented by one each time a line is read from the input file. Set to zero each time a new file is opened.

Awk equivalent

FNR variable

See also

ScanLineNumber

ScanLineNumber Property

TRex Reference

Declaration

```
property ScanLineNumber: word;
```

Description

Runtime only. Incremented by one each time a line is read from the input file. Unlike FileLineNumber, it is not reset to zero when a new file is opened.

Awk equivalent

NR variable

Scan Method

Declaration

```
procedure Scan;
```

Description

Opens each of the files specified in Filespec in turn. Reads each file one line at a time. Splits each line into tokens (as specified by InputSeparator or TokenPattern). Scans each line for a match against one or more of the patterns in MatchPattern.

Awk equivalent

Executing an awk program.

ScanText Method

TRex Reference Example

Declaration

```
procedure ScanText(Text: PChar);
```

Description

Processes Text, one line at a time. Splits each line into tokens (as specified by InputSeparator or TokenPattern). Scans each line for a match against one or more of the patterns in MatchPattern.

Ignores Filespec, if set.

Awk equivalent

Executing an awk program.

ScanText Example

Assume your form contains a TMemo component called Memo1 and a TRex component called Rex1. To scan the contents of the memo component,

```
Rex1.ScanText(Memo1.GetText);
```

GetText Method

TRex Reference

Declaration

```
function GetText: PChar;
```

Description

Returns a pointer to the input line. Intended for use when the input line is longer than 255 characters.

Awk equivalent

```
$0
```

SeekEoln Method

TRex Reference

Declaration

```
procedure SeekEoln;
```

Description

Available only inside an event handler at runtime. Instructs T-Rex to abandon all further processing of this line and proceed to the next one. No further OnToken or OnMatch events will occur for this line. The OnMultipleMatch and AfterLineMatch events will not occur for this line.

SeekEoln raises an ESeekEoln exception, which permits all nested procedure calls in your own code to be cleanly unwound. When SeekEoln is called from an appropriate point in your program, the component handles the exception itself so it does *not* halt your program.

It only makes sense to call SeekEoln in an OnMatch or OnToken event handler. Calling it from other points in your program will result in an unhandled SeekEoln exception.

The Delphi IDE reports exceptions before calling the exception handler, unless you tell it not to. The IDE will report this exception with the message "This is not an error" if you have Halt on Exception switched on, even when you correctly call SeekEoln from an OnMatch or OnToken event handler. This only happens inside the IDE, and only if you have Hold on Exception switched on.

Awk equivalent

```
next
```

SeekEof Method

TRex Reference

Declaration

```
procedure SeekEof;
```

Description

Available only inside an event handler at runtime. Instructs T-Rex to abandon all further processing of this file and proceed to the next one, if any. No OnMultipleMatch event will occur. Will trigger an immediate AfterLineMatch event followed by an OnEof event

SeekEof raises an ESeekEof exception, which permits all nested procedure calls in your own code to be cleanly unwound. If you run your application inside the Delphi IDE, the IDE will report this exception with the message "This is not an error" if you have Halt on Exception switched on. The component handles the exception itself so it does *not* halt your program. But the IDE reports all exceptions, handled or not, before the handler is called, unless you tell it not to.

Awk equivalent

```
close(FILENAME)
```

AfterLineMatch Event

TRex Reference

Declaration

```
property AfterLineMatch:  TNotifyEvent;
```

Description

Occurs after all processing of a line is complete and all OnMatch, OnToken and OnMultipleMatch events have occurred.

Awk equivalent

An action clause with a blank pattern (in other words, matches every line) placed last in the program.

BeforeLineMatch Event

TRex Reference

Declaration

```
property BeforeLineMatch: TNotifyEvent;
```

Description

Occurs after the input line has been tokenized but before any OnMatch events have occurred.

Awk equivalent

An action clause with a blank pattern (in other words, matches every line) placed first in the program.

OnBOF Event

TRex Reference

Declaration

```
property OnBOF: TNotifyEvent;
```

Description

Occurs after a file has been opened, but before any processing of the first line of the file.

Awk equivalent

No standard awk equivalent, except when only one file is being processed: then it is equivalent to the BEGIN pattern (however, NR==1 can be used to achieve almost the same effect). Tawk equivalent is the BEGINFILE pattern.

OnEOF Event

TRex Reference

Declaration

```
property OnEOF: TNotifyEvent;
```

Description

Occurs as the last event in the processing of a file before it is closed.

Awk equivalent

No standard awk equivalent, except when only one file is being processed: then it is equivalent to the END pattern. Tawk equivalent is the ENDFILE pattern.

OnMatch Event

TRex Reference See also

Declaration

```
property OnMatch: TMatchEvent;
TMatchEvent = procedure (Sender: TObject; const Expression: word; const Token:
                string; const InputLine: pchar; const Offset, Length: word) of
                object;
```

Description

This event occurs when an input line contains one or more strings that match one of the patterns in `MatchPattern`.

The pattern that was matched is indicated by `Expression`. The string that matched the pattern is returned in `Token`. Sometimes, knowing the string that matched is not enough. For these cases, the last three parameters provide the entire input line, and the offset and length of the match. That is,

```
StrLCopy(buffer,InputLine[Offset],Length)
```

will copy into `buffer` the same value that is returned as `Token`.

OnMatch may fire several times for the same line, notifying your program of matches on different patterns in `MatchPattern`. Matching is done in ascending sequence of the regular expressions in `MatchPattern`.

OnMatch will fire at most once for a given pattern on a given line. That is, even if an input line contains several strings that match `MatchPattern[0]`, OnMatch will fire only once for `MatchPattern[0]`, returning the leftmost longest match in `Token`.

Awk equivalent

OnMatch is the event that connects a pattern to an action. In awk the connection is made by placing a pattern and an action on the same line of the program.

See also

OnToken event.

OnMultipleMatch Event

Declaration

```
property OnMultipleMatch: TMultipleHitEvent;
TMultipleHitEvent = procedure (Sender: TObject; const MatchSet: TMatchSet) of
                object;
TMatchSet = set of 0..255;
```

Description

If you wish to take action based on the simultaneous match of several patterns, this can be achieved by setting flags in OnMatch. OnMultipleMatch provides a simpler way to do this, returning a set giving all of the patterns in MatchPattern that matched the input line.

Occurs after the last OnMatch event and before the AfterLineMatch event.

Awk equivalent

Patterns of the form `$0 ~ /r1/ && $0 ~ /r2/ && $0 ~ /r3/`

OnToken Event

TRex Reference

Declaration

```
property OnToken: TTokenEvent;
TTokenEvent = procedure (Sender: TObject; const Token: string) of object;
```

Description

Occurs every time a token is identified in the input stream. Occurs after the BeforeLineMatch event, and occurs once for each token in InputLine.

You may not take an action that causes the input line to be recomputed in an OnToken event: that is, you may not assign a value to InputLine, TokenCount, or Token. Attempting to do so raises an EIllicitTrexOp exception.

New elements are added to the token array before the corresponding OnToken event. So, for example, if the input line is

```
A B C D
```

the third OnToken event will return the token 'C'. At that point, TokenCount will be 3, Token[2] will have the value 'C', and Token[3] will not yet contain the value 'D'.

Awk equivalent

```
for (i=1;i <= NF;i++) Handler($i)
```

Note

The OnToken event offers token-by-token processing. The OnMatch event offers line-by-line processing. The two modes of processing are alternatives. In the current implementation all OnToken events occur before the first OnMatch event, but this relative sequence is not guaranteed in future versions.

Association of Shareware Professionals (ASP)

This program is produced by a member of the Association of Shareware Professionals (ASP). ASP wants to make sure that the shareware principle works for you. If you are unable to resolve a shareware-related problem with an ASP member by contacting the member directly, ASP may be able to help. The ASP Ombudsman can help you resolve a dispute or problem with an ASP member, but does not provide technical support for members' products. Please write to the ASP Ombudsman at 545 Grover Road, Muskegon, MI USA 49442-9427, Fax 616-788-2765, or send a CompuServe message via CompuServe Mail to ASP Ombudsman 70007,3536 (Internet: 70007.3536@compuserve.com).

How to Contact Us

You can contact us by mail, by fax, or by email. We don't offer telephone support because the timezones don't fit (our office hours are 10:30 PM to 06:30 AM PST).

Defect Reports

If you report problems with T-Rex, we will be glad to try and resolve them. If you have suggestions about how the components could be improved to be more useful to you, they will be welcomed.

Please submit a support request form to report the defect you have encountered, or to describe the enhancement you need.

When you buy a developer licence, you get guaranteed support for 90 days. The guarantee means that if T-Rex does not work in your application, we will do our best to fix it so that it does work.If we can't fix the defect we will refund your licence fee.

We will support you even if you have not yet licensed T-Rex, but the support will be limited to getting the components installed and working to allow you to evaluate them, and if your problem requires program changes, they may be accorded a low priority.

Support by Mail

Paul Keating
Prodigy Computing
PO Box 2194
Cramerview
South Africa 2060

Please be patient if you use the mail. Airmail delivery from the US takes 3–5 weeks. Surface mail takes up to 4 months.

Support by Fax

Paul Keating
Prodigy Computing
+27-11-888-2370 or +27-11-792-9512.

Support by Email

Email Paul Keating:

CompuServe **73770,660**

Internet **keating@acm.org**

Licensing T-Rex

See also

T-Rex is distributed as shareware. It is not in the "public domain". It is not free software. However, you do have the opportunity to try it for 90 days before you pay for it, subject to the terms of the evaluation licence.

You may not distribute programs of your own that incorporate `T_Rex.dcu` without a developer licence. A developer licence costs $17-50 per developer workstation (or $37-50 if you want the source code). This is a one-off fee: there are no royalties to pay.

Generous site licence discounts are available. Discounts commence at 5 copies. Contact one of the support addresses for further information.

You may distribute copies of the T-Rex package to other developers if you wish: please read the terms of the distribution licence first.

How to Obtain a Developer Licence

Benefits of Obtaining a Developer Licence

Developer Licence Terms

See Also

How to Obtain a Developer Licence
Benefits of Obtaining a Developer Licence
Developer Licence Terms
Evaluation Licence Terms
Distribution Licence Terms
Source Code Licence Terms
Warranty
Copyright

How to Obtain a Developer Licence

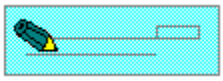
See also

The licence form is in this helpfile, and Help will print it for you, or copy it to the Windows clipboard. Just select your preferred method of payment and follow the prompts.

If you can pay by credit card, you can obtain a licence by email or by fax or by mail.

We accept NetCash for email registrations. NetCash is quick and it's fun!

If you have a CompuServe account, the quickest and easiest way to buy a licence is to use CompuServe's online shareware registration service. This way, the whole process takes place online, you don't have to bother with a form, and the licence fee is simply added to your regular monthly CompuServe bill.



If you need to send a cheque, then you will need to mail your licence fee to us.

Some companies make it difficult to send payment with order. Corporate users who have this problem may issue a purchase order.

Benefits of Obtaining a Developer Licence

See also

1. *Deployability.* Without a developer licence, you can't deploy applications containing `T_Rex.dcu` to end-user sites because the programs won't work. When we receive your licence form we will send you a licence number that will permit you to deploy unlimited copies of your programs.

2. *Support.* Developer licensees get guaranteed support for 90 days. If T-Rex does not work on your system we will do our best to fix it so that it does work. We can't promise to make it work in all cases. If we can't fix the defect we will refund your fee.

3. *Nag-free usage.* After your evaluation licence expires, the program will nag you to license it.

Developer Licence Terms

See also

In return for the developer licence fee, Prodigy Computing grants you a non-exclusive licence to use `T_Rex.dcu` indefinitely without further payment and to distribute programs that contain it freely and without restraint.

1. You may install `T_Rex.dcu` on one Delphi developer's workstation: that is, any computer where there is also installed a licensed copy of Borland International's Delphi development environment.

2. You may write and compile your own application programs using `T_Rex.dcu`.

3. You may use, reproduce, give away or sell any program you write using `T_Rex.dcu`, without additional licence or fees.

4. All copies of the programs you create must bear a valid copyright notice and must be accompanied by a small file that contains your name and licence number in machine-readable form.

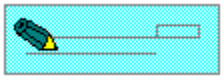
5. The terms of Prodigy Computing's warranty to you are as set out in this helpfile. You agree to those terms. Prodigy Computing provides no warranty to anyone else: you are solely responsible to anyone receiving your programs for support, service, upgrades, or technical or other assistance.

6. You will indemnify and hold Prodigy Computing harmless from and against any claims arising out of the use, reproduction or distribution of your programs.

7. The file `T_Rex.dcu` is and remains the property of Prodigy Computing.

8. This licence is to be construed according to the laws of South Africa. You consent to the jurisdiction of the Magistrate's Court, Randburg, Gauteng in any legal action that is brought by or against you in terms of this licence.

Licensing T-Rex by Mail



On the licence form you will see a button like this:  Press that button to send the form to your printer. Fill it in and mail it to the address shown on the form.

You can pay by credit card (we accept MasterCard and Visa) or by cheque. We can accept cheques in US, Canadian and South African currency. Our agents will process cheques in other currencies.
Please don't draw a cheque in a currency not your own. It costs us more than their face value to cash them.
Sending cash through the mail is risky, but if you accept the risk, we'll happily accept the banknotes.
If you pay by credit card, we ask for your address, because our bank requires us to mail your copy of the completed credit card voucher to you. This address will be used for no other purpose.
Mail is slow. Surface mail (one way) from the US to Southern Africa takes anything up to three months. Airmail takes 2-4 weeks. Surface mail (one way) from Europe takes 3-5 weeks. Airmail takes 8 days.
We will fax your licence number to you (if you provide a fax number) or email it (if you provide an email address), and after some weeks you will receive a credit card voucher in the mail. If you do not provide a fax number or an email address, we will send you your licence number and credit card voucher by airmail.

Another currency

If your bank account is in a currency other than US or Canadian dollars or South African rands, this is unlikely to be a problem, but please contact us before sending payment for pricing and payment instructions.

Licensing T-Rex by Fax

On the licence form you will see a button like this:  Press this button to send the form to your printer. Fill it in and fax it to the number shown on the form.

When you pay by fax you can choose one of two methods. You can pay by credit card (we accept MasterCard and Visa), or send us a company purchase order.
If you use your credit card, we ask for your address because our bank requires us to mail your copy of the completed credit card voucher to you. This address will be used for no other purpose.
We will fax or email your licence number back to you, and after some weeks you will receive a credit card voucher in the mail.

Licensing T-Rex by Email

On the licence form you will see a button like this:  Press that button to copy the form to the clipboard. Use a text editor or the message editor of your communications program to fill it in. Then email it to us: email addresses are on the form.

When you pay by email you can choose one of two methods. You can pay by credit card (we accept MasterCard and Visa), or send us a NetCash voucher.
We will email your licence number back to you.
If you use your credit card, we ask for your address because our bank requires us to mail your copy of the completed credit card voucher to you. This address will be used for no other purpose.
If you are concerned about sending your credit card number over the Internet (and you should be), then either use the split-number technique or encrypt your message using PGP.

Split-Number Technique

Credit card numbers are easy to pick out of a mail message automatically because they have a very fixed structure: 1111-2222-3333-4444 05/96. Packet-sniffers can steal your number by scanning messages and looking for this pattern. If you send two messages with the first half of your number in the first message, and the second half of your number in the second message, your message will be proof against pattern-recognizers.

If you're unconvinced, spend three minutes thinking about how you would write a program that would not be fazed by this technique.

Encrypting Your Message using PGP

If you have a copy of PGP, you can use it to encrypt your message. Included in the T-Rex package is a file called `public.key`, which contains Prodigy Computing's PGP public key. To add this key to your keyring, use the command `pgp -ka public.key`. To encrypt your registration form, save it as a text file (for example `regform.txt`), then use the command `pgp -ea regform.txt Prodigy`. Import or paste the resulting file into an email message.

PGP is short for "Pretty Good Privacy", a public-key cryptosystem using the Rivest-Shamir-Adleman algorithm. PGP is a trademark of Philip Zimmermann and Phil's Pretty Good Software, 3021 Eleventh Street, Boulder, Colorado 80304 USA. PGP is © copyright Philip R. Zimmermann, 1990-1993.

Please don't ask Prodigy Computing or Philip Zimmermann to send you a copy of PGP. Both will refuse, for legal reasons (see below). However, you can probably obtain PGP from your nearest bulletin board, or via anonymous ftp from `nic.funet.fi`, `ghost.dsi.unimi.it` or `src.doc.ic.ac.uk`.

Notice: If you as a private citizen use or distribute PGP within the US, you may be infringing US Patent 4405829, held by Public Key Partners. If you import PGP into the US from another country, you may be infringing US laws regarding the import of munitions. If you import PGP into your own country from the US, you (and the source you import it from) are almost certainly infringing US laws regarding the export of munitions.

Licensing T-Rex Online

CompuServe offers a way to license shareware online. When you do this, the licence fee is added to your CompuServe charges and CompuServe will bill you in the normal way.

To license the program, first find its description in the CompuServe's Shareware database.

To begin, GO SWREG and select "Register Shareware" from the menu. CompuServe calls the process of buying a licence "registration". A list of search criteria will be displayed to you. CompuServe has assigned the object-only T-Rex package the Registration ID 10907. The with-source option has the Registration ID 10908. Enter one of these IDs under option #1 on the search criteria menu. You will then navigate directly to a description of T-Rex. This is the easiest and fastest method of licensing a program.

If you have any questions or concerns about the service, send a message to the Shareware Administrator by selecting the "Provide Feedback" option at the main SWREG menu.

We will send you your licence number (and the program source, if applicable) by CompuServe email.

Purchase Orders

Corporate users who find it administratively difficult to send payment with order may issue a purchase order.

Because of the administration involved, we only accept corporate purchase orders for four copies ($70) or more. We will invoice your company in US dollars (or whatever currency you prefer) and issue the licence numbers on receipt of payment.

Generous site licence discounts are available. Discounts commence at 5 copies. Contact one of the support addresses for further information.

Mail your purchase order to: Prodigy Computing, PO Box 2194 Cramerview, South Africa 2060.

NetCash

NetCash is a new form of online currency that may be used by anyone who has an electronic mail address.

NetCash coupons are essentially a string of numbers. You pay cash, or provide services, to obtain them. You trade them simply by including them in an e-mail message. Like real cash, there are no transaction fees when NetCash changes hands. For this reason, NetCash is ideal for paying small amounts of money over the Internet.

The NetBank, operated by Software Agents Inc, manages the exchange of NetCash between users. The NetBank validates NetCash coupons and makes change.

You can buy NetCash from the NetBank by mail, fax or email. For full details, send an email message to netbank-info@agents.com. Here is a quick summary of how it works:

This is a NetCash coupon that represents a five dollar bill:

```
NetCash US$ 5.00 E123456H789012W
```

To pay for your licence, you simply send one or more NetCash coupons to us in the licence form. After you give the coupon to us, it is no longer yours. You may only spend a NetCash coupon once.

When we receive your coupon, we ask the NetBank if it's valid, by sending the following email message:

```
NetCash US$ 5.00 E123456H789012W /Accept
```

The NetBank will check that the NetCash is valid. If it is, the NetBank will mark the coupon as used up, and issue us a fresh NetCash coupon with a new number. We will keep the coupon in our "cash drawer" and ultimately deposit it into our NetBank account. When we have enough to make it worthwhile, we will turn the coupons into cash.

You pay 2% commission to buy NetCash coupons from the NetBank.

Exchange Rate Fluctuations

The US dollar licence fee will be converted to South African rands at the ruling exchange rate on the day your order is processed, and that amount will be billed to your credit card. Because exchange rates fluctuate, your credit card company may use a slightly different rate when it converts the charge back into your own currency. So the amount actually charged to your card may be slightly lower or slightly higher.

South African users will be billed directly in rands. The rand licence fee is R72-50 (including 14% VAT), or R156-00 if the program source is included.

See Also

Evaluation Licence Terms
Distribution Licence Terms
Source Code Licence Terms
Developer Licence Terms
Warranty
Copyright

Evaluation Licence Terms

See also

These terms form a contract between us. Even though you have not signed the contract, using the product for an exended period indicates your assent.

1. You may install `T_Rex.dcu` on a Delphi developer's workstation: that is, any computer where there is also installed a licensed copy of Borland International's Delphi development environment.

2. You may include `T_Rex.dcu` in as many programs as required to evaluate the product. You may continue to use the unit `T_Rex.dcu` without payment for a trial period of 90 days.

3. You may not distribute to end-users any program that uses `T_Rex.dcu`.

4. At the end of the 90-day trial period you must either delete the program from your computer system or obtain a developer licence to use the program.

5. To legally distribute a program of your own that incorporates `T_Rex.dcu`, you must likewise obtain a developer licence to use the program.

6. You agree to the terms of the warranty.

These restrictions do not apply to the dynamic link library `regexp.dll`. You may use that file without payment and without restraint, provided you reproduce the copyright notice.

Shareware

Shareware means "try-before-you-buy" software.

Shareware is copyrighted software which is distributed by authors through bulletin boards, on-line services, disk vendors, and copies passed among friends. It is commercial software that you are allowed to try before you pay for it.

You do not have to pay for the software until you have had an opportunity to try it out for a reasonable period. You use the software on your own system, in your own work environment, for a fixed period, like 90 days. If you decide not to continue using it, you throw it away and forget all about it. You only pay for it if you continue to use it.

Shareware is a distribution method, not a type of software. There is good and bad shareware, just as there is good and bad retail software. If retail software turns out to be unsatisfactory, you might have trouble getting your money back. Wth shareware, you know if it's good or bad before you pay for it.

Source Code Licence Terms

See also

You may choose to license a copy of the source code along with your developer licence for `T_Rex.dcu`. If you do, your use of the source code is subject to the following conditions:

1. You may write and compile your own application programs using the source code.

2. You may use, reproduce, give away or sell any program you write using the source code, in executable form only, without additional licence or fees; provided that the programs that you distribute may not be merely a subset of `T_Rex.dcu`.

3. All copies of the programs you create must bear a valid copyright notice.

4. The terms of Prodigy Computing's warranty to you are as set out in this helpfile. You agree to those terms. Prodigy Computing provides no warranty to anyone else: you are solely responsible to anyone receiving your programs for support, service, upgrades, or technical or other assistance.

5. You will indemnify and hold Prodigy Computing harmless from and against any claims arising out of the use, reproduction or distribution of your programs.

6. The source code is and remains the property of Prodigy Computing. Regardless of any modifications that you make, you may not distribute the source code. You are not, of course, restricted from distributing source code that is entirely your own.

7. This licence is to be construed according to the laws of South Africa. You consent to the jurisdiction of the Magistrate's Court, Randburg, Gauteng in any legal action that is brought by or against you in terms of this licence.

These restrictions do not apply to the dynamic link library `regexp.dll`. You may use that file without payment and without restraint, provided you reproduce the copyright notice.

The source code will be delivered to you by email after we receive your licence form. To keep costs (and licence fees) low, we don't mail out disks.

Distribution Licence Terms

See also

The following may distribute the T-Rex package including `T_Rex.dcu` and all of its supporting materials completely unaltered, without further permission:

- ◊ private individuals passing copies to friends without charge;
- ◊ bulletin board systems;
- ◊ bulletin board file distribution networks;
- ◊ disk vendors who are ASP Vendor Members; and
- ◊ disk vendors who are not ASP Vendor Members but who disclose to their customers prior to purchase in a visible fashion that the product is shareware, the nature of shareware, and that separate payment to the copyright owner is required if the product is used beyond the 90-day trial period.

For other channels of distribution or to distribute in modified form, you must consult the data record in the file VENDINFO.DIZ, which is included in the package, and which is hereby incorporated by reference. Any distribution satisfying all the distribution requirements expressed in that data record is hereby authorized. Distribution that does not conform to the requirements of this licence nor to the requirements expressed in the attached data record requires explicit written permission from the copyright owner in every case.

This distribution licence does not permit you to incorporate `T_Rex.dcu` into an application program for distribution for end-users. To distribute the product in that form you must have a developer licence.

These restrictions do not apply to the dynamic link library `regexp.dll`. You may use that file without payment and without restraint, provided you reproduce the copyright notice.

Acknowledgements and Copyright

PRODIGY

About Awk

T-Rex was inspired by the awk language. We don't consider a programmer's toolkit to be complete without an awk interpreter. You can learn about awk in Aho, Kernighan and Weinberger: *The awk progamming language*, Reading MA: Addison-Wesley, 1988. The implementation of awk we use is Tawk version 4.0 by Thompson Automation, 5616 SW Jefferson, Portland OR 97221.

About the Regular Expression Compiler Engine

The regular expression compiler is supplied in a dynamic link library called `regexp.dll`. This compiler was originally written by Henry Spencer for the University of Toronto. The code was modified by Borland International in 1992 to compile with Borland C++ 3.1 and for use in a DLL. It was further modified by Vincent Risi in 1996 to accept conventional escape sequences.

No Warranty

T-Rex is made available *voetstoots* (a Roman-Dutch legal condition that excludes, among other things, all supplier's warranties of any kind, express or implied, against defects latent or patent). You assume all responsibility for the adverse consequences of any defects in T-Rex, including any adverse consequences of including it in your own products. If T-Rex does not work, or if it works differently from the way you expected or intended or were led to believe, your sole remedy is to stop using it, uninstall it from your Delphi environment, and remove all references to it from your own program code. You may in certain cases claim a refund of any licence fee you have paid.

TRexList Object

Methods

A TRexList object is a descendant of TStrings for storing regular expressions.

The regular expressions that you specify live in the Strings property. When you add a regular expression, using Add, AddStrings or Insert, the object compiles the string representation into a finite state recognizer for it, and the recognizer is stored in the corresponding Objects property of TRexList.

Using TRexList: Example

First create an object of type TRexList, like this:

```
var MyRexList: TRexList;
...
MyRexList := TRexList.Create;
```

Next, add one or more regular expressions to the list:

```
MyRexList.Add('^[0-9]+$');
```

This tells MyRexList that it is looking for strings that consist only of numbers.

Finally test the data you are working with against MyRexList:

```
if MyRexList.MatchFirst (somestring, e, s, l) then
                writeln (somestring, 'matched on r.e. ',e, 'beginning at byte
                ',s,' for ',l,' bytes');
```

or

```
if MyRexList.MatchAll (somestring, matchset) then
                writeln (somestring, 'matched');
```

Methods

MatchFirst

MatchAll

MatchFirstP

MatchAllP

MatchFirst and MatchFirstP Methods

Example

Declaration

```
function  MatchFirst (const InputString: string; var index, start, length:
                word): boolean;
function  MatchFirstP(const InputString: PChar;  var index, start, length:
                word): boolean;
```

Description

These functions provide the analogue of the OnMatch event for objects of type TRexList that you define yourself.

Call `MatchFirst`, passing it a string. If it returns `True`, then the string you passed it matched one of the regular expressions in stored in the Strings property. Which regular expression matched is returned by `index`, with zero meaning the regular expression stored in `Strings[0]`; and the location of the match in `InputString` is returned by `start` and `length`.

`MatchFirstP` is provided to avoid pointless conversion between strings and character arrays. Internally, `MatchFirst` calls `MatchFirstP`.

MatchAll and MatchAllP Methods

Example

Declaration

```
function  MatchAll   (const InputString: string; var MatchSet: TMatchSet):
                boolean;
function  MatchAllP  (const InputString: PChar;  var MatchSet: TMatchSet):
                boolean;
```

Description

These functions provide the analogue of the OnMultipleMatch event for objects of type TRexList that you define yourself.

Call `MatchAll`, passing it a string. If it returns `True`, then the string you passed it matched one or more of the regular expressions in stored in the Strings property. Which regular expression matched is returned by `MatchSet`, which is a `set of 0..255`. `MatchSet` will contain one element for every regular expression matched, with `[0]` meaning the regular expression stored in `Strings[0]`.

`MatchAllP` is provided to avoid pointless conversion between strings and character arrays. Internally, `MatchAll` calls `MatchAllP`.

TRegExp Object

Methods Example

A TRegExp object contains a finite state recognizer for a regular expression that you specify. Creating the object compiles the recognizer from the regular expression. Destroying the object releases the memory occupied by the recognizer.

Using TRegExp

Suppose you want to match some data against a regular expression supplied by the user, independently of what happens under the control of the TRex component.

First, create an object of type TRegExp:

```
var MyRegExp: TRegExp;
...
MyRegExp := TRegExp.Create(UserRE);
```

Now match the data against this object, like this:

```
if MyRexList.Match (somestring, s, l) then
                writeln (somestring, ' matched user r.e. ', UserRE, '
                beginning at byte ',s,' for ',l,' bytes');
```

Methods

Create
Match
MatchP
MatchImmediate
MatchImmediateP
Subst
SubstP
SubstImmediate
SubstImmediateP

Create Method

Example

Declaration

```
constructor Create(expstring: string);
```

Description

Calling Create creates the TRegExp object, and compiles the regular expression that is specified in `expstring`.

Match and MatchP Methods

Example

Declaration

```
function    Match(MatchString: string; var MatchStart, MatchLength: word):
                boolean;
function    MatchP(MatchString: PChar; var MatchStart, MatchLength: word):
                boolean;
```

Description

These functions return `True` if `MatchString` matches the regular expression that was supplied to `TRegExp.Create`.

Awk equivalent

`match()` or `~`

Subst and SubstP methods

Declaration

```
function Subst(var Target: string; ReplaceString: string; MaxRep: Word): word;
function SubstP(Target, ReplaceString: PChar; MaxSize, MaxRep: word): word;
```

Description

These functions examine `Target` for the regular expression that was supplied to `TRegExp.Create`. If one is found, it is replaced with the string specified in `ReplaceString`.

You place a limit on the number of times the replacement operates by passing it in `MaxRep`. The usual values for `MaxRep` are `1`, meaning replace the leftmost longest occurence, and `maxint`, meaning replace all nonoverlapping occurrences, starting with the leftmost longest.

Leftmost longest nonoverlapping means the following: 1. The effect of substituting the string `aa` for all occurrences of the regular expression `a` in the target `banana` is `baanaanaa`. 2. The effect of substituting the string `x` for all occurrences of the regular expression `ana?` in the target `banana` is `bxna`. 3. The effect of substituting the string `x` for all occurrences of the regular expression `a?na` in the target `banana` is `bxx`.

The replacement operation can cause `Target` to grow longer. `MaxSize` specifies the size of the buffer pointed to by `Target`. If a replacement operation would cause the string to exceed `MaxSize`, then `Subst` returns early. `Subst` has no `MaxSize` parameter. The implied maximum is 255 or the length of the string passed to `Subst`, whichever is smaller.

These functions return the number of substitutions actually performed. If no matches are found, it will be zero. If `Subst` returns early because continuing would cause `Target` to grow beyond its buffer, then the value returned will be less than `MaxRep`. The value returned will never be greater than `MaxRep`.

Awk equivalent

`gsub()` or `sub()`. The `&` symbol in the replacement string is not supported.

MatchImmediate and MatchImmediateP functions

Declaration

```
function MatchImmediate  (var RegularExp: string; MatchString: string; var
                MatchStart, MatchLength: word): boolean;
function MatchImmediateP (var RegularExp: string; MatchString: PChar; var
                MatchStart, MatchLength: word): boolean;
```

Implementation

```
re := TRegExp.Create(RegularExp);
MatchImmediate := re.Match(MatchString, MatchStart, MatchLength);
re.Free;
```

Description

Provides a simple single-line call to Match and MatchP when there is no benefit in compiling `RegularExp` beforehand because it is or may be different on each call.

Awk equivalent

`match()` or `~` with the first parameter a variable or an expression (not `/..../`).

SubstImmediate and SubstImmediateP functions

Declaration

```
function function SubstImmediate  (var RegularExp: string; var Target: string;
                ReplaceString: string; MaxRep: word): word;
function SubstImmediateP (var RegularExp: string; Target, ReplaceString:
                PChar; MaxSize, MaxRep: word): word;
```

Implementation

```
re := TRegExp.Create(RegularExp);
SubstImmediate := re.Subst(Target,ReplaceString,MaxRep);
re.Free;
```

Description

Provides a simple single-line call to Subst and SubstP when there is no benefit in compiling `RegularExp` beforehand because it is or may be different on each call.

Awk equivalent

`sub()` or `gsub()` with the first parameter a variable or an expression (not `/..../`). The `&` symbol in the replacement string is not supported.

Prodigy Computing **Print this form**

T-Rex Developer Licence

Mail: Cheques

I want a developer licence for my copy of T-Rex. Please license the program in the name of

_____

and send me a licence number.

I enclose a cheque in favour of Prodigy Computing (Pty) Ltd for the amount shown below.

My bank account is in	Currency	Standard Fee	Fee Including Source Code
US dollars	USD	$17-50	$37-50
Canadian dollars	CAD	$23-75	$51-00
SA rands	ZAR	R72-50 inc VAT	R156-00 inc VAT

Another currency...

Email address or fax number or mailing address *(for notification of licence number)*:

_____

_____

_____

_____

NB: If you want the source code, please supply an email address.

Mail to:
Prodigy Computing
PO Box 2194
Cramerview 2060
South Africa

$Revision:: 1.25 $

Prodigy Computing **Print this form** 

T-Rex Developer Licence

Mail: Credit Cards

I want a developer licence for my copy of T-Rex. Please license the program in the name of

_____

and send me a licence number.

[ ] Charge the $17-50 licence fee to my [ ] MasterCard [ ] Visa account.

[ ] I want the source code.

Charge the $37-50 licence fee to my [ ] MasterCard [ ] Visa account.

I understand that exchange rate fluctuations may slightly affect the amount billed.

Number: _____

Expiry date: _____ / _____ *(must be supplied)*

Name on card *(if different from name above)*:

_____

Postal Address *(for return of credit card voucher: our bank insists that we mail it to you)*

_____

_____

_____

_____

Signature

_____

Email address or fax number *(for notification of licence number: if omitted you will be notified by mail)*:

_____

NB: If you want the source code, please supply an email address.

Mail to:

Prodigy Computing
PO Box 2194
Cramerview 2060
South Africa

$Revision:: 1.25 $

Prodigy Computing **Print this form** 

T-Rex Developer Licence Form

Copy to Clipboard 

Fax/Email

I want a developer licence for my copy of T-Rex. Please license the program in the name of

_____

and send me a licence number.

[ ] I am paying the normal $17-50 licence fee.

[ ] I want the source code and so I am paying the $37-50 licence fee.

[ ] Here is a NetCash coupon for the licence fee:_____

[ ] Charge the licence fee to my [ ] MasterCard [ ] Visa account. I understand that exchange rate fluctuations may slightly affect the amount billed.

Note: If you are concerned about sending your credit card number over the Internet, then use either the split-number technique or encrypt your message using PGP.

Card number: _____

Expiry date: _____ / _____ *(must be supplied)*

Name on card *(if different from name above)*:

_____

Postal Address *(for return of credit card voucher: our bank insists that we mail it to you)*

_____

_____

_____

_____

Signature *(for fax orders):*

_____

Email address or fax number *(for notification of licence number)*:

_____

NB: If you want the source code, please supply an email address.

Fax to:
Prodigy Computing, +27-11-888-2370 or +27-11-792-9512. The + means your international call prefix, such as 011 in the US or 00 in Europe.

Email to:
CompuServe: 73770,660
Internet: keating@acm.org

$Revision:: 1.25 $

Prodigy Computing **Print this form** 

T-Rex Support Request

Copy to Clipboard 

Name and email address for response:

This is

[ ] a defect report.

[ ] an enhancement request.

For defects: Do you think the problem is

[ ] a permanent logic error?

[ ] data-related?

[ ] timing-related?

[ ] platform-related?

What version of T-Rex are you using? (The revision number is returned by the string function `T_Rex.Revision`. It is also in the timestamp of `T_Rex.dcu` and inside `vendinfo.diz`.)

Please describe in as much detail as you would like to get if this support request were coming to you about your own products:

Fax to:
Prodigy Computing, +27-11-888-2370 or +27-11-792-9512. The + means your international call prefix, such as 011 in the US or 00 in Europe.

Email to:
CompuServe: 73770,660
Internet: keating@acm.org

$Revision:: 1.25 $

Licence Number

The licence number we send you is to be included, along with your name, in a file called `t_rex.ini` that identifies you as a licensed developer. This file is to be placed in the Windows directory on each end-user's workstation.

We think this is a simple, straightforward scheme, but some developers dislike it. If you're one of them, we recommend that you take the source code option, which has the licence number traps removed, since you could easily remove them yourself.