SRP_String TokenizeCode

Parses a string containing BASIC+ into an array of discrete BASIC+ tokens.

Syntax

```
Tokens = SRP_String("TokenizeCode", String, WhitespaceOptions, IncludeComments, ref State)
```

Parameters

Parameter	Description
String	A string of BASIC+ code. **(REQUIRED)**
WhitespaceOptions	Whitespace tokenizing options. (*OPTIONAL, Default="None")*
IncludeComments	TRUE to include comments, FALSE to omit them. (*OPTIONAL, Default=0)*
State	The parser state.

Returns

An @FM delimited array of strings, each one a BASIC+ token.

Remarks

This service takes a string of BASIC+ code and returns an @FM delimited array of tokens. A token is a discrete element of code: a string, an operator, whitespace, comment, identifier, etc. This was written to support the SRP Precompiler, which parses BASIC+ code in order to provide language enhancements. This routine does all the work of breaking up the code into tokens for easier analysis. To use this service, pass your BASIC+ code to the String parameter. The remaining parameters provide some options for controlling the output.

WhitespaceOptions: This parameter supports one of three values: "None", "AllWhitespace", or "LineBreaksOnly". The default option is "None", in which case, whitespace tokens are never included in the final array. "AllWhitespace" has the exact opposite effect: all whitespace tokens are included in the output. "LineBreaksOnly" will omit whitespace composed of spaces and tabs but include carriage return and/or line feeds. This last option could be useful for detecting line breaks in code while still ignoring all other whitespace.

IncludeComments: This true/false parameter determines whether or not comment tokens are included in the output. The default is FALSE.

State. *This parameter was added in SRP Utilities 2.1.* It's a numerical value indicating the parser state. This is useful if you are parsing code line by line and want to preserve the state of the code from one line to the next. The best way to use this parameter is to create a variable called State and set it to "". Then pass it into this service, which will update your variable after each call. Pass it as is, and you'll be passing the previous state into the parser.

Example

```
// Sample code
Code = "A = B + C; // This is a simple expression"

// Output will be: "A":@FM:"=":@FM:"B":@FM:"+":@FM:"C":@FM:";"
Tokens = SRP_String("TokenizeCode")

// Output will be: "A":@FM:" ":@FM:"=":@FM:" ":@FM:"B":@FM:" ":@FM:"+"::@FM:" "@FM:"C":@FM:";":@FM:" ":@FM:
\0D0A\
Tokens = SRP_String("TokenizeCode", "AllWhitespace")

// Output will be: "A":@FM:"=":@FM:"B":@FM:"+":@FM:"C":@FM:";":@FM:\0D0A\
Tokens = SRP_String("TokenizeCode", "LineBreaksOnly")

// Output will be: "A":@FM:"=":@FM:"B":@FM:"+":@FM:"C":@FM:";":@FM:"// This is a simple expression"
Tokens = SRP_String("TokenizeCode", "None", 1)
```

See Also

DetokenizeCode