VBossAPI v1.0 rev 2.00 Reference Manual

Copyright © 1995, Greg Truesdell

CIS ID **: 74131,2175**
Internet **: 74131.2175@compuserve.com**

VBossAPI.DLL is a Visual Basic language extension module providing capabilities useful when building script and language compilers and interpreters. This library is in use by companies and individuals in Canada, Europe and The United States.

This DLL is designed for, and requires, Visual Basic for Windows.

Help File Updated: 95.04.23

Contents

Reference

Getting Started

With Release 2.00, you need to handle a few extra steps to insure proper operation of VBossAPI. This section describes the three special calls required to initialize and exit the library.

The Very FIRST Step

To begin, you must call CreateScrObject(). This function initializes the library to support the current instance of the program you are running. It returns an integer handle that you must use when exiting your program.

```
'
' First Step
'

Global Script%

...
...

Form_Load ()

   Script% = CreateScrObject()

   If Script% = -1 Then

      Print "Sorry, No enough memory to continue"
      End

   End If

End Sub
```

Registering the Library with your Registration Key

When you register the library you receive a personal registration key. The next step would be to use the RegisterVBossAPI function to disable the shareware panel(s).

```
'
' Register the Library
'

If Not RegisterVBossAPI( Name$, Key$ ) Then

   Print "Sorry, Incorrect Registration Information"

End If

'
' The program continues anyway ....
'
```

Final Step - Usually in the main form's UnLoad event.

Finally, you must destroy the script object created in Step 1. You must use the integer handle returned above (Script%) as the argument to the DestroyScrObject procedure:

```
'
' All Done
'
Form_UnLoad ()

   DestroyScrObject Script%

End Sub
```

Constants

These constants can be found in VBossAPI.BAS. Check that file for lastest additions as well.

- **AddKeyword() and related functions:**

```
'
'   AddKeyword() Return Codes
'
Global Const AKW_NO_MORE_ROOM = -1       'no more keyword space
Global Const AKW_INVALID_CHAR = -2       'invalid character in keyword
Global Const AKW_DUPLICATE_KEYWORD = -3 'duplicate keyword
Global Const AKW_KEYWORD_TOO_LONG = -4  'keyword too long
Global Const AKW_INVALID_TOKEN = -5     'invalid token (keycode) value
                                        '- negative numbers not allowed
Global Const AKW_TYPE_MISMATCH = -6     'AddVariable() type mismatch
Global Const AKW_OVERFLOW = -7          'AddVariable() overflow
'
'   AddKeyword() Limits
'
Global Const AKW_MAX_KEYWORD_LEN = 16   'maximum keyword length
Global Const AKW_MAX_KEYWORDS = 256     'maximum number of keywords
```

- **AddVariable() and related functions:**

```
'
' Variable Type Constants
'
Global Const VTNONE = 0
Global Const VTSTRING = 1
Global Const VTINTEGER = 2
Global Const VTFLOAT = 3
Global Const VTPROCEDURE = 4    ' defined to help implement procedures by
name
Global Const VTFUNCTION = 5     ' defined to help implement functions by
name
Global Const VTLABEL = 6        ' defined to help implement labels
```

- **General Constants used with VBossAPI.DLL**

```
Global Const OSS_MAX_WORD_LEN = 255 ' maximum word length
```

- **NextToken() and related functions:**

```
'
'   NextToken Return Codes (in <token>)
'
'
Global Const NT_MAX_OPERATORS = 32      ' reserved operator tokens
'   Note: Positive numbers >= NT_MAX_OPERATORS are valid tokens

Global Const NT_PAST_EOL = -1           ' end of the line or string
Global Const NT_NO_KEYWORDS = -2        ' no keywords in keyword DB
Global Const NT_TOKEN_NOTFOUND = -3     ' next word can not be tokenized
Global Const NT_NO_FREE_MEMORY = -4     ' no heap available for buffer
Global Const NT_VARIABLE_FOUND = -5     ' keyword found was a variable
Global Const NT_LABEL_FOUND = -6        ' keyword parsed was a label name
Global Const NT_FUNCTION = -7           ' keyword parsed was a function
name
Global Const NT_PROCEDURE = -8          ' keyword parsed was a procedure
name

Global Const NT_USER_ERROR = -99        ' added as a convience, not used
internally

'
'  Operator Constants - Tokens returned by NextToken() for operators
'
'  + - * / " ' ; : [ ] { } ( ) ! @ # $ % ^ & = < > ,
'
Global Const NT_PLUS = 1
Global Const NT_MINUS = 2
Global Const NT_TIMES = 3
Global Const NT_DIVIDE = 4
Global Const NT_DBL_QUOTE = 5
Global Const NT_SNG_QUOTE = 6
Global Const NT_SEMICOLON = 7
Global Const NT_COLON = 8
Global Const NT_LEFTBRACKET = 9
Global Const NT_RIGHTBRACKET = 10
Global Const NT_LEFTBRACE = 11
Global Const NT_RIGHTBRACE = 12
Global Const NT_LEFTPAREN = 13
Global Const NT_RIGHTPAREN = 14
Global Const NT_EXCLAMATION = 15
Global Const NT_AT = 16
Global Const NT_POUND = 17
Global Const NT_DOLLAR = 18
Global Const NT_PERCENT = 19
Global Const NT_CARET = 20
Global Const NT_AMPERSAND = 21
Global Const NT_EQUAL = 22
Global Const NT_LESSTHAN = 23
Global Const NT_GREATERTHAN = 24
Global Const NT_COMMA = 25
```

- EvalErrorString() Returned Error codes:

```
'
'  EvalExpression Error Codes (returned via EvalErrorString())
'
Global Const EXPR_SYNTAX_ERROR = 1
Global Const EXPR_PARAMETER_MISSING = 2
Global Const EXPR_PARAMETER_COUNT_ERROR = 3
Global Const EXPR_INVALID_PARAMETER = 4
Global Const EXPR_OVERFLOW = 5
Global Const EXPR_COMMA_MISSING = 6
Global Const EXPR_MISSING_RPAREN = 7
Global Const EXPR_TYPE_MISMATCH = 8
Global Const EXPR_INVALID_IDENTIFIER = 9
Global Const EXPR_PARAMETERS_NOT_ALLOWED = 10
Global Const EXPR_EXPECTED_FACTOR = 11
Global Const EXPR_EXPECTED_TERM = 12
Global Const EXPR_EXPECTED_EXPRESSION = 13
Global Const EXPR_ZERO_DIVIDE = 14
Global Const EXPR_OUT_OF_MEMORY = 15
Global Const EXPR_GARBAGE_FOLLOWS = 16
Global Const EXPR_VARIABLE_EQUATE_ERROR = 17
Global Const EXPR_INVALID_FUNCTION = 18
```

Copyright

Your use of VBossAPI.DLL indicates your acceptance of the following terms and conditions:

History of Changes

The following summarizes changes to VBossAPI.DLL in each SHAREWARE release.

Version 1.0 rev 2.01

- Squashed VTSTRING variable declaration bug that damaged the keyword database when the 65th variable was created. Now properly supports 128 VTSTRING variables.

Version 1.0 rev 2.00

- The library now supports multiple instances (separate programs accessing the library concurrently.)
- Enhanced the demonstration program to demonstrate a two-pass method of execution: Pass One: Locate and define Labels. Pass Two: Normal execution.
- Documentation Update

Version 1.0 rev 1.72

- Increased DLL code execution speed by optimizing iterative blocks.
- Documentation Update

Version 1.0 rev 1.71

- Modified temprary string creation logic.
- Several minor efficiency modifications.

Version 1.0 rev 1.62

- Fixed a parsing bug encountered when the first few characters to be parsed were operators. This effected NextToken() and PeekNextToken().
- Added ParseUntil() function.
- NextToken() and PeekNextToken() now recognize Labels, Procedures and Functions.
- Added TestNumExpr() function.
- Removed resource reference to BWCC.DLL
  My thanks to Richard Miller on CompuServe for catching this one.
- Sample application SweetPEA updated to include IF..THEN GOTO and improved comments in the sample source code.

Version 1.0 rev. 1.61b

- Maintenance Release
- COMMA character added to the reserved token list.

Version 1.0 rev 1.61

- Added VARIABLE EXPRESSION EVALUATOR module.

Version 1.0 rev 1.52

- First SHAREWARE Release.

Introduction to VBossAPI.DLL

The VBoss (Visual Basic Optimized Script Support) API is a Windows DLL designed to help the Visual Basic application programmer define a script language.   There are, however, a number of related functions provided for completeness and utility.

Getting Started

The library is divided into three major functional groups:

- Word Related Functions
- Token Related Functions
- Support and Utility Functions

Word Related Functions

- These functions operate on a string of characters, and are designed to help the programmer parse words and characters within the string. Functions are provided for parsing, counting, locating and collecting words.

Token Related Functions

- These functions also operate on a string of characters, and are designed to support token (representitive) values, keyword lists, variable lists and operators.

Support and Utility Functions

- These functions are not strictly required in a Script Support DLL, but have been made available since the DLL either uses them, or the probability is great that a VB program would need them.

Word Related Functions

Word related functions are used to parse strings of words, locate words and count words in strings. Note that Word related functions are one-based. This makes it easier to use with Visual Basic functions like Mid$() etc.   Token related functions, however, use zero-based character indexes.

GetWordAt()
LocateWord()
ParseStr()
ParseUntil()
WordCount()

GetWordAt()

A word related function that returns the idx-th word using a programmer supplied set of delimiters.

Visual Basic Declaration:

```
Declare Function GetWordAt Lib "VBossAPI.DLL" (ByVal idx As Integer, ByVal
st As String, ByVal delims As String) As String
```

Parameters:

- idx as Integer
  Word index. If idx = 3 and st = "Hello there, world and delims = " ," then GetWordAt() would return "world".
- st as String
  String of words to parse for the idx-th word.
- delims as String
  String of delimiters used to delimit words in the st.

Returns:

- String
  The idx-th word in st is returned. If idx is greater than the number of words in the string, then a null string is returned.

LocateWord()
WordCount()

Example:

```
' This example will print "golly"
Print GetWordAt(2, "Good golly, Miss Molly!", " ,!")
```

LocateWord()

A word related function used to locate the character position of the idx-th word using a programmer defined set of delimiters.

Visual Basic Declaration:

```
Declare Function LocateWord Lib "VBossAPI.DLL" (ByVal idx As Integer,
ByVal st As String, ByVal delims As String) As Integer
```

Parameters:

- idx as Integer
  Word index. If idx = 3 and st = "Hello there, world" and delims = " ," then LocateWord() would return 14.
- st as String
  String of words to parse.
- delims as String
  String of delimiters used to delimit words in the st.

Returns:

- Integer
  The character index of the first character of the idx-th word in st.

Comments:

Word related functions, like this one, return one (1) based indexes.   This makes it easier to use Mid$() etc in Visual Basic using the index returned. Token based functions, however, are zero-based.

GetWordAt()
WordCount()

Example:

```
' This example will print 6.
Print LocateWord(2, "Good golly, Miss Molly!", " ,!")
```

WordCount()

A word related function used to count the number of words in a string based on a programmer provided set of delimiters.

Visual Basic Declaration:

```
Declare Function WordCount Lib "VBossAPI.DLL" (ByVal st As String, ByVal
delims As String) As Integer
```

Parameters:

- st as String
  String of words to parse.
- delims as String
  String of delimiters used to delimit words in the st.

Returns:

- Integer
  Returns the number of words delimited by <delims>.

Example:

```
' This example will print 6.
Print WordCount("Paradox exists only in belief systems.", " .")
```

ParseStr()

A word related function used to return the first or next word in a string based on a set of programmer provided delimiters.

Visual Basic Declaration:

```
Declare Function ParseStr Lib "VBossAPI.DLL" (start As Integer, ByVal st
As String, ByVal delims As String) As String
```

Parameters:

- start as Integer Variable
  Character index of first character in string to start parsing. If start = 0 or 1 then ParseStr() will begin at the first character.
- st as String
  String of words to parse.
- delims as String
  String of delimiters used to delimit words in the st.

Returns:

- Integer (start)
  After parsing the string, start will contain either the index to the next word in the string or -1 to indicate that no more words are available (EOL).
- String
  Returns the word located starting at character <start>, delimited by <delims> in string <st> or null string if no more words (EOL).

Example:

```
' This example will print:
'
'  My
'  mother
'  the
'  car

st = "My mother, the car."
start = 0
Word$ = ParseStr(start, st, " ,.")

While start > -1

   Print Word$
   Word$ = ParseStr(start, st, " ,.")

Wend
```

ParseUntil()

A word related function used to return every character from the current character until a delimiter character is found.

Visual Basic Declaration:

```
Declare Function ParseUntil Lib "VBossAPI.DLL" (start As Integer, ByVal st
As String, ByVal cset As String) As String
```

Parameters:

- start as Integer Variable
  Character index of first character in string to start parsing. If start = 0 then ParseStr() will begin at the first character.
- st as String
  String of words to parse.
- cset as String
  String of delimiters used to delimit words in the st.

Returns:

- Integer (start)

  After parsing the string, start will contain either the index to the next word in the string.   Unlike ParseStr(), this function does not return -1 if at end of the string. Instead, the character index returned is invalid.
- String

  Returns the string copied from the current location (start) until a character in cset was located.

ParseStr()

Token Related Functions

Token related functions are used to define keywords and their tokens, variables and their types, access the token and variable lists and implement syntax parsing functions. Unlike Word related functions, token related functions use zero-based character indexes.

Token and Keyword Functions:

Definition Functions:

AddKeyword()
GetKeyword()
GetKeywordToken()
GetTokenKeyword()

List Functions:

KeywordCount()
LoadKeywords()
SaveKeywords()
ZapKeywords

Parsing Functions:

EvalErrorString()
EvalExpression()
NextToken()
PeekNextToken()
TestNumExpr()

Support Functions:

DefTokenDelims()
NT_CodeString()
NT_Operators()

Variable Functions:

AddVariable()
GetVariable()
SetVariable()
VariableCount()
ZapVariables

AddKeyword()

A token related function used to add keywords and tokens to the keyword list.

Visual Basic Declaration:

```
Declare Function AddKeyword Lib "VBossAPI.DLL" (ByVal kw As String, ByVal
kc As Integer) As Integer
```

Parameters:

- kw as String
  Keyword string. The length of this string can be no greater then AKW_MAX_KEYWORD_LEN.

  This string MUST be unique. You cannot define a token that allready exists; that includes operators (which are predefined). All tokens are converted to uppercase before storage.
- kc as Integer
  Integer token value (keyword code).

  There are some limitations on what values you may use as tokens. No token value can be negative, and MUST be greater or equal to MAX_OPERATORS. Operator codes reserve the first set of tokens and negative values are used internally and represent error return codes..

Returns:

- Integer
  If successful, a positive number >= MAX_OPERATORS is returned (this number represents the slot in the Keyword List where this token is stored.) Otherwise a negative error code is returned. See Constants() for error codes.

Constants()
Limitations

Example:

```
rc = AddKeyword("begin", 100)
rc = AddKeyword("end", 101)
rc = AddKeyword("print", 102)
rc = AddKeyword("input", 103)

Print ""
Print "Keyword Count is " & KeywordCount()
Print "Keywords:"

For ii = 1 To KeywordCount()

   Print GetKeyword(ii)

Next
```

Limitations

The current implementation of VBossAPI.DLL imposes the following limitations:

Internal Definitions:

- 256 Keyword/Token entries.
- 128 Variables of type VTSTRING.
- Numeric variables limited only by available local memory.
- Arrays are not implemented.

This is due to the use of internal tables for Keyword/Tokens and Variables. This greatly enhances the execution of the parsing engine. It also allows the library to access larger amounts of memory as well as simplifying the handling of resources. VBossAPI will complain if more than one program attempts to use it.

- String variables are limited to 255 characters.
- VTFLOAT type variables are internally limited to eight decimal places.

GetKeyword()

A token related function that returns the keyword string for the keyword located as the idx-th entry in the list.

Visual Basic Declaration:

```
Declare Function GetKeyword Lib "VBossAPI.DLL" (ByVal idx As Integer) As
String
```

Parameters:

- idx as Integer

  The index into the keyword list. The first keyword in the list is 1 (one). The last keyword in the list is KeywordCount().   If KeywordCount() = 0 then the list is empty.

Returns:

- String
  The text Keyword if the value of idx is valid, otherwise a null string is returned.

Example:

```
' list all stored keywords and their token values
if KeywordCount() > 0 then

   for idx% = 1 to KeywordCount()

      kw$ = GetKeyword(idx%)
      Print kw$ & " = " & GetKeywordToken(kw$)

   next idx%

endif
```

GetKeywordToken()

A token related function used to return the integer token value of the character string previously added to the keyword list with AddKeyword().

Visual Basic Declaration:

```
Declare Function GetKeywordToken Lib "VBossAPI.DLL" (ByVal kw As String)
As Integer
```

Parameters:

- kw as String
  The keyword to locate.

Returns:

- Integer
  If successful, the token value assigned to this keyword, otherwise a negative value (-1) is returned.

Comments:

GetKeywordToken() is provided to allow the programmer to access the Keyword list stored internally by VBossAPI. It can also be used to test for the existence of a keyword. If the token returned is -1, then the keyword does not exist. Use AddKeyword() to add a keyword to the list.

AddKeyword()
GetTokenKeyword()

Example:

```
' dynamic keyword definition example

if GetKeywordToken("BEGIN") = -1 then

   rc% = AddKeyword("BEGIN",101)

endif
```

GetTokenKeyword()

A token related function used to return the text keyword referred to by the integer token value previously stored with AddKeyword().

Visual Basic Declaration:

```
Declare Function GetTokenKeyword Lib "VBossAPI.DLL" (ByVal token As
Integer) As String
```

Parameters:

- token as Integer
  The token previously assigned by AddKeyword() to a keyword.

Returns:

- String
  The keyword assigned to this token.

  If unsuccessful, a null string is returned.

Comments:

This function is provided the allow the programmer to decode a token. This may be used to provide debugging capabilities during development. It can also be used to test whether a given token has already been assigned.

AddKeyword()
GetKeywordToken()

Example:

```
' Locate first free token value
' In this simplistic example, the do while..loop
' would continue until a free token was found.
ii = 1
do while Len(GetTokenKeyword(ii)) > 0

   ii = ii + 1

loop
```

KeywordCount()

A token related function which returns the number of keywords currently stored in the keyword list.

Visual Basic Declaration:

```
Declare Function KeywordCount% Lib "VBossAPI.DLL" ()
```

Parameters:

- None

Returns:

- Integer
  The number of keywords in the keyword list.   Does NOT include the internally defined operators.

AddKeyword()

LoadKeywords()

A token related function used to load a list of keywords and tokens previously saved with the SaveKeywords() function.

Visual Basic Declaration:

```
Declare Function LoadKeywords Lib "VBossAPI.DLL" (ByVal filename As
String) As Integer
```

Parameters:

- filename as String
  Name of the file to load the keyword list from.

Returns:

- Integer
  Returns 0 if successful, -1 otherwise.

Comments:

Provided to help implement alternate language keywords for the same set of tokens.

AddKeyword()
SaveKeywords()

SaveKeywords()

A token related function used to save the current contents of the keyword list.

Visual Basic Declaration:

```
Declare Function SaveKeywords Lib "VBossAPI.DLL" (ByVal filename As
String) As Integer
```

Parameters:

- filename as String
  Name of the file to save the keyword list.

Returns:

- Integer
  Returns 0 if successful, -1 if not.

Comments:

Saves the entire keyword list structure to a file.   Included primarily to help implement alternate language keywords while retaining the same tokens.

AddKeyword()
LoadKeywords()

ZapKeywords and ZapVariables

Token related functions used to completely erase the contents of the keyword list or variable list.

Visual Basic Declarations:

```
Declare Sub ZapKeywords Lib "VBossAPI.DLL" ()
Declare Sub ZapVariables Lib "VBossAPI.DLL" ()
```

Parameters:

- None

Returns:

- Nothing

Comments:

Erases ALL Keywords or Variables.

EvalErrorString()

An expression evaluation function used to return the error code and descriptive text for the last evaluation error in EvalExpression()

Visual Basic Declaration:

```
Declare Function EvalErrorString Lib "VBossAPI.DLL" (errcode As Integer)
As String
```

Parameters:

- errcode as Integer (variable)

  EvalErrorString() returns the numeric code for the last evaluation error. This parameter must be a variable.

Returns:

- Integer (errcode)
  See errcode above.

Constants()

- String

  The single-line text description of the last error found while evaluating an expression with EvalExpression()

EvalExpression()
TestNumExpr()

Example:

```
'
' An example use of EvalErrorString
'
answer$ = EvalExpression("Test = ABC", rc%)

If Not rc% Then

   Print EvalErrorString( rc% ) & " [Error#" & rc% & "]"

End If
```

EvalExpression()

A token related function used to evaluate infix notation numeric expressions.

Visual Basic Declaration:

```
Declare Function EvalExpression Lib "VBossAPI.DLL" (ByVal ExprStr As
String, rc As Integer) As String
```

Parameters:

- ExprStr as String
- rc as Integer (variable)

Returns:

- rc as Integer
  True if successful, otherwise false.

- String
  The string representation of the results of the calculation.

  EvalExpression will return a null string if the calculation was unsuccessful, otherwise the string returned can be used as a parameter to the Val() function if it's numeric value is required.

Comments:

The expression evaluator is a very important part of any set of language tools. The evaluator is linked closely with the variable definition table created and maintained by VBossAPI. In fact, the evaluator is capable of defining and returning variables and their values. It works much like Visual Basic does. This version of the evaluator is designed for integer and real variables only.

You may predefine variables (and in fact should, to ensure that string variables are properly allocated) using the AddVariable() function, or let the evaluator define the variable. The evaluator defaults it self-defined variables as reals. It will convert integers and reals on-the-fly to insure that calculations are successful. The example given below should give you some idea of how you can use the evaluator.

EvalExpression() comes with built-in functions available (in an up-comming version, you will be able to define the actions of your own functions):

PI, ABS, ARCTAN, COS, EXP, LN, SQR and SQRT are available. Each, of course, return a value of type VTFLOAT (real).

Constants()
EvalErrorString()
TestNumExpr()

Example:

```
'
' This example demonstrates how EvalExpression can evaluate expressions
' including variables and functions.
'
' In the following example, the variables Radius and Area are created
' automatically.  They are accessible by either EvalExpression or
' GetVariable()
'
Dim rc As Integer
Dim VarType As Integer

If EvalExpression( "Radius = 1.24", rc ) <> "" Then

   If EvalExpression( "Area = Pi * Sqr(Radius)", rc ) <> "" Then

      Print "The area of a circle with a radius of ";

      ' return the value of Radius using EvalExpression

      Print EvalExpression("Radius", rc);

      ' return the value of Area using GetVariable()

      Print " is equal to " & GetVariable("Area", VarType)

   End If

End If
```

TestNumExpr()

Token related function used to evaluate the truth of a numerical expression.

Visual Basic Declaration:

```
Declare Function TestNumExpr Lib "VBOSSAPI.DLL" (ByVal LExpr As String,
ByVal Op As String, ByVal RExpr As String, Success As Integer) As Integer
```

Parameters:

- LExpr as String
  A valid numeric expression. May include numeric variables, functions and constants.
- Op as String
  A test operator. The only tests supported are =, <, >, <=, >=, <>. If your script language requires !=, !, # , == etc, then you must translate the operator before calling this function.
- RExpr as String
  A valid numeric expression. May include numeric variables, functions and constants.

Returns:

- Success as Integer (True/False)
- Integer (True/False)

  True if the expression is true.

Comments:

This function is provided to simplify the implementation of control constructs such as if ... then, while ... wend, do ... until etc.

EvalErrorString()
EvalExpression()

Example:

```
'
' This example demonstrates the use of TestNumExpr()
'
Dim Success As Integer
Dim Result As Integer
Result = TestNumExpr("1+2*3", "=", "7", Success )

If Not Success then

   Print "Error in Expression"

ElseIf Result then

   Print "Expression is TRUE"

Else

   Print "Expression is FALSE"

End If
```

NextToken() and PeekNextToken()

Token related functions used to begin or continue parsing defined tokens.

NextToken() and PeekNextToken() differ only in that PeekNextToken() does not update the character index variable (start).   PeekNextToken() allows you to peek at what the next token is without updating the character index.

Visual Basic Declaration:

```
Declare Function NextToken Lib "VBossAPI.DLL" (start As Integer, ByVal st
As String, token As Integer) As String
```

```
Declare Function PeekNextToken Lib "VBossAPI.DLL" (ByVal start As Integer,
ByVal st As String, token As Integer) As String
```

Parameters:

- start as Integer Variable
  Character index of first character in string to start parsing. If start = 0 or 1 then NextToken() will begin at the first character of the string.

  This parameter is a variable. NextToken() will return the updated value of start, indicating the character location of the next word to parse.
- st as String
  String of characters to parse for the next token.
- token as Integer Variable
  The token value, if any, of the currently parsed word.

  If an error occured during parsing, the error code will be returned in <token>.

Returns:

- start as Integer Variable
  Updated to the next character start position in the string.
- token as Integer Variable
  Contains the token for the current word, or an error code.
- String
  The word parsed. If the token value is positive, then this is a keyword.   If the token value is negative, then either the word is not a keyword, or an error has occured.

Comments:

NextToken() and PeekNextToken() are the core functions of VBossAPI. Once you have defined the working parameters for your language, you then use these functions to parse lines of script text.   By placing NextToken() in your main processing loop, you can use Select..Case..End Select statements to implement the language.

Constants()

Example:

```
' Simple parsing example
'
' This example is intentionally kept simple, and
' should be concidered pseudo-code. A guide to how
' you might implement the main loop of a script
' interpreter.
'
' In the example calls for DIM, PRINT and INPUT you
' will notice the use of ii% as a parameter. This is
' because, in all likelyhood, the implementation
' of these keywords will require the parsing index
' value (ii%) to continue parsing st$ for
' parameters they require.
'
' Assume the variable st$ contains the following text:
'
' Dim A$
' Begin
'    A = "Hello"
'    Input A
'    Print A
' End
'
rc% = AddKeyword("BEGIN", 100)
rc% = AddKeyword("END",999)
rc% = AddKeyword("INPUT", 201)
rc% = AddKeyword("PRINT", 202)
rc% = AddKeyword("DIM",203)

ii% = 0
running = True
token = 0

Do While running

   keyword$ = NextToken(ii%, st$, token)

   Select Case token

      Case 100
        DoBegin() ' your call to implement BEGIN

      Case 999, NT_PAST_EOL
        DoEnd()  ' your call to implement END
        running = False

      Case 201
        DoInput(ii%) ' your call to implement INPUT

      Case 202
        DoPrint(ii%) ' your call to implement PRINT

      Case 203
        ' your call to implement variable
        ' creation (see AddVariable())
        If Not DimVariable(ii%) then
```

```
            running = False
        End If

      Case NT_VARIABLE_FOUND
        ' your variable equating code
        ' (see SetVariable())
        DoSetVariable(ii,Keyword)

      Case NT_TOKEN_NOTFOUND
        DoSyntaxError() ' your syntax error code
        running = False

      Case Else
        DoOtherError() ' your other error code
        running = False

   End Select

Loop
```

DefTokenDelims()

A token related support function that returns a string containing the default set of token delimiters.

Visual Basic Declaration:

```
Declare Function DefTokenDelims Lib "VBassAPI.DLL" () As String
```

Parameters:

- None

Returns:

- String

  Returns a string containing the default token parsing delimiters.   This string can be used in other functions that require a delimiter string, if desired.

Example:

```
' Sample using Word function WordCount()
Print WordCount("Here, in this box, is the answer.", DefTokenDelims()+".")
```

NT_CodeString()

A token related function used to return a context-class statement based on the value of the integer token provided.

Visual Basic Declaration:

```
Declare Function NT_CodeString Lib "VBossAPI.DLL" (ByVal token As Integer)
As String
```

Parameters:

- token as Integer
  Normally the token value returned by NextToken() or PeekNextToken().

Returns:

- String
  A context statement describing the class of token represented by the value of token.

Comments:

This function is provided for the programmer for debugging purposes.   It returns a short statement that describes the class of the token based on the NT_* result codes.

NextToken()

NT_Operators()

A token related function used to return the string of pre-defined operators.

Visual Basic Declaration:

```
Declare Function NT_Operators Lib "VBossAPI.DLL" () As String
```

Parameters:

- None

Returns

- String
  Containing the string of operators defined by VBossAPI

Comments:

The first character of the string contains the first operator.   This means that the token code for the first operator is equal to 1.

DefTokenDelims()

Example:

```
' Determining the token for a given operator.
' This example returns a token value of 3.

token% = Instr( NT_Operators(), "*" )
```

AddVariable()

A token related function used to add variable declarations to the variable list.

Visual Basic Declaration:

```
Declare Function AddVariable Lib "VBossAPI.DLL" (ByVal vname As String,
ByVal vtype As Integer, ByVal vdata As String) As Integer
```

Parameters:

- vname as String
  Name of the variable to add. Can not be longer than AKW_MAX_KEYWORD_LEN.   Must be unique.
- vtype as Integer
  The variable type.
- vdata as String
  The variable data.

  All data is copied to VBossAPI as a string. The value of vdata is determined by the value of vtype.   Type checking is done internally to verifiy the type. For example: If vtype = VTINTEGER, then vdata could contain "125", but would be a type mismatch is it contained "one hundred and twenty five."

Returns:

- Integer
  Returns the enumerated variable type if successful. Otherwise returns an error code.

Constants()
Limitations
Variable Data Types

Example:

```
' Variable handling example.

rc% = AddVariable( "Balance", VTFLOAT, "324.94" )
if rc% > -1 then

   amt = Val(GetVariable("Balance", rc%)) + 10.32
   rc% = SetVariable("Balance", Str(amt))

   if rc% > -1 then

      Print GetVariable("Balance", rc%)

   else

      Print "SetVariable failed with error code " & rc%

   endif

else

   Print "AddVariable failed with error code " & rc%

endif
```

Variable Types

VBossAPI Variable Table Data Types:

Undefined	VTNONE
String	VTSTRING
Integer	VTINTEGER
Float	VTFLOAT
Procedure	VTPROCEDURE
Function	VTFUNCTION
Label	VTLABEL

Constants

GetVariable()

A token related function used to return the string representation of the value currently stored for the variable keyword. The variable type is also returned. The variable must have been stored with the AddVariable() function.

Visual Basic Declaration:

```
Declare Function GetVariable Lib "VBossAPI.DLL" (ByVal vname As String,
vtype As Integer) As String
```

Parameters:

- vname as String
  The name of the variable to be retrieved.
- vtype as Integer Variable
  This parameter should be provided as an Integer variable, not a constant. Its original contents are ignored and overwritten.

Returns:

- vtype as Integer Variable
  The variable type or error code is returned in this variable.

If successful:	Contains the variable type of the Variable vname.
If unsuccessful:	Contains the returned error code.
- String
  Returns the variables contents as a string.
  If unsuccessful returns a null string.

Variable Data Types
AddVariable()
GetKeyword()
SetVariable()

Example:

```
' Variable handling example.

rc% = AddVariable( "Balance", VTFLOAT, "324.94" )
if rc% > -1 then

   amt = Val(GetVariable("Balance", rc%)) + 10.32
   rc% = SetVariable("Balance", Str(amt))

   if rc% > -1 then

      Print GetVariable("Balance", rc%)

   else

      Print "SetVariable failed with error code " & rc%

   endif

else

   Print "AddVariable failed with error code " & rc%

endif
```

SetVariable()

A token related function used to modify the variable contents of a previously defined variable.

Visual Basic Declaration:

```
Declare Function SetVariable Lib "VBossAPI.DLL" (ByVal vname As String,
ByVal vdata As String) As Integer
```

Parameters:

- vname as String
  The text name of the variable to set. Limited in length to AKW_MAX_KEYWORD_LEN.
- vdata as String
  The string representation of the data to store with this variable. Limited in length to OSS_MAX_WORD_LEN.

Returns:

- Integer
  Success returns a positive number representing the data type of the variable, otherwise a negative error code is returned. (See AddVariable() for error codes.)

AddVariable()
GetVariable()

VariableCount()

A token related function that returns the number of variables in the variable database.

Visual Basic Declaration:

```
Declare Function VariableCount Lib "VBossAPI.DLL" () As Integer
```

Parameters:

- None

Returns

- Integer
  Number of declared variables in the variable database.

Support and Utility Functions

Utility functions provided to assist the programmer with implementation issues involving filenames, strings and .DLL accessing.

Filename Related Functions:

DirOnly()
ExtOnly()
FullPath()
NameOnly()
ReplacePath()

String Related Functions:

PackSpaces()

DLL Access Related Functions:

LPGetVBStr()
VBStrGetLP()

DirOnly()

A general purpose function that returns only the path (directory) part only for the qualified filename given.

Visual Basic Declaration:

```
Declare Function DirOnly Lib "VBossAPI.DLL" (ByVal fn As String) As String
```

Parameters:

- fn as String
  The filename.

Returns:

- String
  The Directory part of the filename. (Includes the trailing "\")

ExtOnly()
FullPath()
NameOnly()
ReplacePath()

Example:

```
'
' This example prints "C:\DATA\"
'
Print DirOnly("C:\DATA\BOOK.ONE")
```

ExtOnly()

A general purpose function that returns only the extension part of a qualified filename.

Visual Basic Declaration:

```
Declare Function ExtOnly Lib "VBossAPI.DLL" (ByVal fn As String) As String
```

Parameters:

- fn as String
  Valid partial or full filename.

Returns:

- String
  Only the extension (with preceeding ".") for the filename.

DirOnly()
FullPath()
NameOnly()
ReplacePath()

Example:

```
'
' prints ".DAT"
'
Print ExtOnly("D:\Editor\Config.Dat")
'
' prints ""
'
Print ExtOnly("D:\Editor\DataFile")
```

FullPath()

A general purpose function used to expand a filename into a completely qualified filename.

Visual Basic Declaration:

```
Declare Function FullPath Lib "VBossAPI.DLL" (ByVal fn As String) As
String
```

Parameters:

- fn as String
  Valid partial or full filename.

Returns:

- String
  Fully qualified path for this file.   Uses the current drive and directory if necessary.

ExtOnly()
DirOnly()
NameOnly()
ReplacePath()

Example:

```
'
' If the current directory is C:\ACCOUNTS
' then the following code would set fp$ = "C:\ACCOUNTS\AUDIT.TXT

fp$ = FullPath("audit.txt")
```

NameOnly()

A general purpose function used to return only the name part of a qualified filename.

Visual Basic Declaration:

```
Declare Function NameOnly Lib "VBossAPI.DLL" (ByVal fn As String) As
String
```

Parameters:

- fn as String
  Valid partial or full filename.

Returns:

- String
  Only the file name part of the filename.

DirOnly()
ExtOnly()
FullPath()
ReplacePath()

Example:

```
'
' This code will print "README"
'
Print NameOnly("C:\Windows\ReadMe.Txt")
```

ReplacePath()

A general purpose function used to replace the path part of a filename with a new path. The string containing the new path may be a completely qualified filename.

Visual Basic Declaration:

```
Declare Function ReplacePath Lib "VBossAPI.DLL" (ByVal fn As String, ByVal
np As String) As String
```

Parameters:

- fn as String
  Source filename.
- np as String
  New path filename. May be a completely qualified filename. Only the (valid) path part of the filename will be used.

Returns:

- String
  The newly created filename with the path replaced by the path part of <np>.

DirOnly()
ExtOnly()
FullPath()
NameOnly()

Example:

```
'
' This code will print F:\DATA\SMITH.TXT
'
path1$ = "C:\ACCOUNT\SMITH.TXT
path2$ = "F:\DATA\SOURCE.DAT"

Print ReplacePath(path1$, path2$)

'
' This example will print "C:\MESSAGE.EXE"
'
Print ReplacePath("F:\BACKUP\MESSAGE.EXE","C:\")
```

PackSpaces()

A general purpose function provided to pack multiple spaces and tabs within a string. Leading and trailing spaces are preserved, but packed to a single space.

Visual Basic Declaration:

```
Declare Function PackSpaces Lib "VBossAPI.DLL" (ByVal st As String) As
String
```

Parameters:

- st as String
  Contains the string to be packed.

Returns:

- String
  All multiple spaces and tabs are compressed into a single space. This compresses the string to the minimum size required for simple parsing.

Example:

```
' This example will print "This is a test."
Dim st As String

   st = "This      is  a    test."
   Print PackSpaces(st)
```

LPGetVBStr()

A general purpose function provided to return a Visual Basic String from a zero-terminated string.

Visual Basic Declaration:

```
Declare Function LPGetVBStr Lib "VBossAPI" (ByVal pStr As Long) As String
```

Parameters:

- pStr as Long
  Pointer to a zero terminated string (lpsz).

Returns:

- String
  Returns a Visual Basic String created from the string pointer.

Comments:

This function is provided to allow VB programmers to collect a string from a pointer returned by a Windows .DLL library call.

VBStrGetLP()

VBStrGetLP()

A general purpose function provided to return a pointer to the zero-terminated string within a Visual Basic String.

Visual Basic Declaration:

```
Declare Function VBStrGetLP Lib "VBossAPI" (ByVal pStr As String) As Long
```

Parameters:

- pStr as String
  Visual Basic String   passed by value.

Returns:

- Long
  Returns a pointer to the lpsz portion of the VB String.

Comments:

Some .DLL library functions require a pointer to a zero terminated string.   This function allows the VB programmer to pass the address of a declared and sized string as a parameter. Strictly speaking, however, this function is not often required, since the VB ByVal modifier will pass the address of the string.   It is included, none the less, for completeness as it may be useful in some circumstances.

LPGetVBStr()

Example:

```
' Passing a string to a .DLL library call.
' This example uses a Windows .DLL function call to
' convert a string to ANSI uppercase.
'
' THIS IS A LOWERCASE STRING will be printed.

Dim szBuffer As String * 128

   szBuffer = "this is a lowercase string."
   AnsiUpperBuff(VBStrGetLP(szBuffer), 128)

   Print szBuffer
```

Registration

To register VBossAPI.DLL, you must obtain a registration key from the author.   The registration key is then used by your program to register the DLL.   This will disable the shareware registration dialog that appears whenever the DLL is loaded or used by your program. You will also receive the latest version of the library.   This key will work on all subsequent bug-fix and minor revision releases until a new version is released.

Obtaining a Registration Key

To obtain a registration key you must send the registration amount to:

Greg Truesdell
Suite 308
633 North Road
Coquitlam, BC
CANADA
V3J 1P3

Registration Fee Options:

- CompuServe SWREG ID# 4362: US$19.95

  The registration key will be sent to you via CompuServe E-Mail within 24 hours of receipt.   You will also receive a ZIP archive containing the distribution files.
- Mail: US$23.95

  The registration key will be sent to you via return mail.   You will also recieve a 3½ disk containing the distribution files. The package will be mailed to you within 24 hours after receiving your payment.   With this option you MUST send a MONEY ORDER made out to GREG TRUESDELL.

Registration Form:

Note: All registration information is held in the strictest of confidence.

```
--------------------------------------------------------------------------
-------
            VBossAPI Function Library (DLL) v1.0 Registration Form
--------------------------------------------------------------------------
-------

  Mail this registration form to:

     Greg Truesdell
     Suite 308
     633 North Road
     Coquitlam, BC
     CANADA
     V3J 1P3

  or E-Mail to:

     CIS User ID :74131,2175
     Internet    :74131.2175@compuserve.com

--------------------------------------------------------------------------
-------

Registered User Name:   [
]
                 { will be used as the UserID for RegisterVBossAPI() }

Company Name:           [
]

Address:                [
]
                        [
]
                        [
]

City:                   [                              ]
State/Prov.:            [                              ]
Zip/Postal Code:        [            ]

Tel:                    [(   )   -    ]

E-Mail or CIS User ID:  [                           ]

Fee Option:  [] E-Mail (US$19.95) [] Regular Mail (US$23.95)

--------------------------------------------------------------------------
-------
  REMEMBER: PAYMENT MUST BE MADE BY MONEY ORDER made payable to GREG
TRUESDELL.
            Checks will not be accepted unless you are a resident of
            BRITISH COLUMBIA, CANADA.
--------------------------------------------------------------------------
-------
```

Using the Registration Key

RegisterVBossAPI()

You should include the RegisterVBossAPI() call before ANY other VBossAPI.DLL function.

RegisterVBossAPI()

This function is used to register the shareware version of the library.   If successful it will disable any and all shareware related nag screens etc.

Visual Basic Declaration:

```
Declare Function RegisterVBossAPI Lib "VBossAPI.DLL" (ByVal UserID As
String, ByVal RegID As String) As Integer
```

Parameters:

- UserID as String

  This is the Registered User ID exactly as you provided it in the registration form. Case is significant!

- RegID as String

  This is the Registration ID (key) sent to you after you sent your payment and registered the library.

Results:

- Integer

  Returns 0 if successful, otherwise -1.

Registration

Comments:

Once you have purchased the Registration Key you can use this function to inhibit the shareware nag dialog(s).

Example:

```
'
' assuming your Registration User ID was "J. Smith" and
' the registration key sent to you was "123456789"
'
If RegisterVBossAPI( "J. Smith", "123456789" ) = -1 Then

   MsgBox "Invalid Registration Key for J. Smith"

End If
```

Glossary

A

AKW_DUPLICATE_KEYWORD

AKW_INVALID_CHAR

AKW_INVALID_TOKEN

AKW_KEYWORD_TOO_LONG

AKW_MAX_KEYWORD_LEN

AKW_MAX_KEYWORDS

AKW_NO_MORE_ROOM

AKW_OVERFLOW

AKW_TYPE_MISMATCH

API

D

Default token parsing delimiters

Delimiters

Index

#
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z

AKW_DUPLICATE_KEYWORD

AKW_DUPLICATE_KEYWORD = -3

An attempt was made to add a duplicate keyword or variable.

AKW_INVALID_CHAR

AKW_INVALID_CHAR = -2

An invalid character was passed in a keyword or variable name.

AKW_INVALID_TOKEN

AKW_INVALID_TOKEN = -5

An invalid token value was passed in AddKeyword(). Tokens are only legal as integers from MAX_OPERATORS to 32768.

AKW_KEYWORD_TOO_LONG

AKW_KEYWORD_TOO_LONG = -4

The keyword or variable name passed was greater in length than AKW_MAX_KEYWORD_LEN.

AKW_MAX_KEYWORD_LEN

AKW_MAX_KEYWORD_LEN = 16

AKW_MAX_KEYWORDS

AKW_MAX_KEYWORDS = 256

The maximum number of keywords that the keyword list can hold.

AKW_NO_MORE_ROOM

AKW_NO_MORE_ROOM = -1

Not enough memory to allocate another keyword or variable record.

AKW_OVERFLOW

AKW_OVERFLOW = -7

String length greater than 255 characters or an attempt to set a variable to a value larger than the defined data type.   This error occurs in calls to AddVariable().

AKW_TYPE_MISMATCH

AKW_TYPE_MISMATCH = -6

The data passed in AddVariable() or SetVariable() is incompatable with the defined data type.

API
Application Programming Interface

Default token parsing delimiters

Internally set by VBossAPI for token parsing routines.   It is a character set containing all control characters [chr(1) to chr(31)], the space and the comma.

Delimiters

A string of one or more characters used to delimit words in a string of characters.   For example: "This is a String" contains four words delimited by spaces.

DLL
Dynamic Link Library

Keyword Lists

An internally maintained database of Keywords including the token representing the keyword. This list is used when parsing a string using the NextToken() and PeekNextToken() functions.

Keyword

The string of characters (word) you wish to assign a token value to.   VBossAPI restricts the length of keywords to MAX_KEYWORD_LEN characters.

lpsz
Long Pointer to String Zero-terminated.

MAX_OPERATORS.

MAX_OPERATORS = 32

NT_FUNCTION

NT_FUNCTION = -7

Token is a variable of type VTFUNCTION

NT_LABEL_FOUND

NT_LABEL_FOUND = -6

Token is a variable of type VTLABEL

NT_MAX_OPERATORS

NT_MAX_OPERATORS = 32

Internally, VBossAPI reserves 32 characters for operators.
User-defined tokens are >= NT_MAX_OPERATORS with a maximum value of 32768.

NT_NO_FREE_MEMORY

NT_NO_FREE_MEMORY = -4

Not enough free memory to allocate the parsing buffer.
NT_CodeString returns "NO MEMORY" for this code.

NT_NO_KEYWORDS

NT_NO_KEYWORDS = -2

There are no keywords in the keyword list.
NT_CodeString returns "NO KEYWORDS" for this code.

NT_PAST_EOL

NT_PAST_EOL = -1

Parsed past the end of line or file. NextToken() and PeekNextToken() return a null string if EOL is encountered.

NT_CodeString returns "PAST EOF" for this code.

NT_PROCEDURE

NT_PROCEDURE = -8

Token is a variable of type VTPROCEDURE.

NT_TOKEN_NOTFOUND

NT_TOKEN_NOTFOUND = -3

Word parsed was not a keyword. Not a tokenized word.
NT_CodeString returns "NOT A TOKEN" for this code.

NT_VARIABLE_FOUND

NT_VARIABLE_FOUND = -5

Keyword found was a Variable name. No token value.
NT_CodeString returns "VARIABLE" for this code.

Null String

An empty string containing no characters (length = 0).

Operators

An operator is a special character used to represent a function. Normally operators are used to define arithmetic and string evaluation functions. (ie: MyVal = 2 * (Amount))
The operators in the above example are =, *, (and).

Internally, the operators are set to +-*/'";:[]{}()!@#$%^&=<>

OSS_MAX_WORD_LEN

OSS_MAX_WORD_LEN = 255

Maximum number of characters allowed in a text fragment.

OSS_MAX_WORD_LEN.

OSS_MAX_WORD_LEN = 255

Maximum length of string data stored in VBossAPI variables.   Some functions will truncate strings.

Parsing

Parse (pars,parz) : To dissect (a sentance) according to the grammatical functions of its parts.

Qualified Filename

A filename containing all components required to uniquely identifiy a file.

```
[DRIVE:][PATH\][NAME][.EXT]
```

Token

Defined as a coded representation of a given set of characters (or Keyword). By referencing an integer value for a command, it is possible to create a script execution module independent of the language or spelling of a Keyword.

Variable Lists

A list of variable declaration Keywords, their data type and current value.

VTFLOAT

VTFLOAT = 3

A floating point number.

VTFUNCTION

VTFUNCTION = 5

This variable type is provided for completeness.   Internally to VBossAPI it is stored as a String (VTSTRING).

VTINTEGER

VTINTEGER = 2

A two byte integer having values of -32767 to +32768.

VTLABEL

VTLABEL = 6

This variable type is provided for completeness.   Internally to VBossAPI it is stored as a String (VTSTRING).

VTNONE

VTNONE = 0

Variable is un-assigned.

VTPROCEDURE

VTPROCEDURE = 4

This variable type is provided for completeness.   Internally to VBossAPI it is stored as a String (VTSTRING).

VTSTRING

VTSTRING = 1

A string variable.
Strings are limited to 255 characters in length.

Word

Defined as a set of contigious characters delimited by a non-inclusive set of characters. In other words, a block of characters separated by one or more differing characters.