OE Standard Object Reference

class ObjectBase

ObjectBase is the base class of all OE the built-in classes. The following methods are available to any derived class and are therefore available anywhere in a program.

Operators

This class has instance methods to support the following operators.
== === <> != > >= < <= <=>

These operators have minimal functionality. They are included so that any two objects can be compared without throwing a PropertyNotFound exception. The **<=>** operator is usually overridden for any class (such as Integer, String, etc.) that have meaningful comparisons.

Instance Methods

`String` **`className`**;

This method returns the name of the class of the receiver object.

`String` **`superClass`**;

This method returns a Null value for instance objects.

`String` **`classTree`**;

This method returns a string value containing the class hierarchy back to Object.

`Object` **`onInit`**[(arg list)];

This is the default onInit method that will be called by the make method when the receiver does not have one of its own. It performs no function other than preventing a property not found exception from occurring.

`Integer` **`count`**;

This method always returns an integer value of 1 for the instance object. User objects can override this method if there are object states which should indicate a value different than one. An array, for example, will return a count value indicating the number of array elements.

`Boolean` **`isTrue`**;

This method always returns a True value for the instance object. User objects can override this method if there are object states which should indicate a False condition.

`Boolean` **`isFalse`**;

This is the opposite of the isTrue method.

`Boolean` **`isEmpty`**;

This method always returns a False value for the instance object. User objects can override this method if there are object states which should indicate a True condition.

`Boolean` **`isNull`**;

This method always returns a False value for the instance object. User objects can override this method if there are object states which should indicate a True condition.

`Boolean` **`isInstanceOf`**(String name);

This method returns a True value if the receiver is an instance of the class identified by the 'name' parameter.

`Boolean` **`isKindOf`**(String name);

This method returns a True value if the 'name' parameter identifies the class or any superclass of the receiver.

`Boolean` **`isClassDef`**(String name);

This method returns a False value.

`Boolean` **`isNaN`**[(obj)];

This method returns a `True` value if object specified (obj) is a `NaN` value. If obj is not specified, then the receiver is tested for having a NaN value.

`Boolean` **`isUndefined`**[(obj)];

This method returns a `True` value if object specified (obj) is an `Undefined` value. If obj is not specified, then the receiver is tested for having an Undefined value. Synonyms are **`isUndef`** and **`isU`**.

`Boolean` **`isDefined`**[(obj)];

This works in the opposite way from the `isUndefined` method. It returns a `False` value if object specified (obj) is an `Undefined` value. If obj is not specified, then the receiver is tested for having an Undefined value. Synonyms are **`isDef`** and **`isD`**.

`Boolean` **`isInfinity`**[(obj)];

This method returns a `True` value if object specified (obj) is a Positive or a Negative infinity value. If obj is not specified, then the receiver is tested for having a Positive or a Negative infinity value.

Integer result=object1 <=> object2

This method compares the receiver object with the value specified. It returns 0 if the objects are equal, it returns -1 if the receiver is less than the value object and it returns 1 if the receiver is greater than the value object.

`String` **`toString`**;
`String` **`toS`**;

This method returns a string representation of the receiver object. For user defined objects this information will be minimal. You may override this method in user defined objects for a more accurate representation.

`Integer` **`toInteger`**;
`Integer` **`toI`**;

This method returns an integer representation of the receiver object, if applicable. For user defined objects this information will result in an error. You may override this method in user defined objects for a correct value.

`Number` **`toNumber`**;
`Number` **`toN`**;

This method returns a real number representation of the receiver object, if applicable. For user defined objects this information will result in an error. You may override this method in user defined objects for a correct value.

`Number` **`toArray`**;
`Number` **`toA`**;

This method returns a one element array containing only the receiver object.

Class Methods

`Integer` **`alert`**(String message);

This method displays an alert dialog box with the specified message.

`Object` **`make`**[(arg list)];

If the receiver is a class object, this method creates a new instance of the object . After the object is created, it calls the onInit method of the new object using the arguments passed to it.

If the receiver is an instance, an InvalidDataType exception is thrown.

`String` **`className`**;

This method returns the name of the class of the class object. For class objects, this will always be “ClassDef”.

`String` **`objectName`**;

This method returns the name of the class object.

`String` **`superClass`**;

This method returns the name of the super class. For the Object class, it returns Null.

`String` **`classTree`**;

This method returns a string value containing the class hierarchy back to Object.

`Boolean` **`isTrue`**;

This method always returns a True value for the class object. User defined class objects can override this method if there are object states which should indicate a False condition.

`Boolean` **`isFalse`**;

This is the opposite of the isTrue method.

`Boolean` **`isEmpty`**;

This method always returns a False value for the class object. User defined class objects can override this method if there are object states which should indicate a True condition.

`Boolean` **`isNull`**;

This method always returns a False value for the class object. User objects can override this method if there are object states which should indicate a True condition.

`Boolean` **`isClassDef`**;

This method returns a True value.

`Boolean` **`hasSuperClass`**(String name or classDef);

This method returns a True value, if the name is in the class tree or if the class object specified is in the class tree.

`Boolean` **`isClassOf`**(String name or classDef);

This method returns a True value, if the name is in the class tree or if the name matches the receiver class.

`Integer` **`write(String msg)`**;

This method writes the 'Msg' string value to the standard output stream returning the number of characters.

`Integer` **`writeln(String msg)`**;

This method is the same as the write method except that it starts a new line after writing.

Null **`import(String fileName)`**;

The import method loads another program module. This can be a source file (.oes), an evm object file (.evmo) or an evm extension file (.dll). If a complete file path is provided, it will use that to locate the file to import. If only a file name is provided (no path), then it searches the search path for the file.

Struct **`versionInfo(String fileName)`**;

The import method returns a structure with information about the current version of the evm. The returned structure has the following integer properties: versionMajor, versionMinor, buildNumber and the expiration date (for evaluation versions). The expiration date is also a structure containing integer year, month and day properties.

Null **`exitProgram[(Integer exitValue)]`**;

This method causes the program to terminate. The default exist value is 0. If zero is specified or used by default, it will be a normal program exit. The program will ‘unwind’ and all the active **`try`** exception blocks with **`anyway`** clauses will be run. If the exitValue is negative, it specifies an abort condition. The **`anyway`** clauses will not be run and program will terminate with an exitProgram exception. If the exitValue is positive, then the program will terminate with an exitProgram exception, but all the **`anyway`** clauses will be run.

Boolean **`utf16Mode[(Boolean mode)]`**;

This method gets and/or sets the utf16 mode. The initial value is False. This means that the string values are examined as if they were 16-bit unicode values only. It will not account for values outside of the basic multilingual plane (Bmp). If your program needs to use string values with characters outside this range (a rare occurrence), then this should be set to True. Performance for string values is faster with this mode set to False.

class Boolean

The Boolean class represents a True or False value. It is returned by many methods.

Operators

The Boolean class has instance methods to support the following operators.
== === <> != > >= < <= <=>

Instance Methods

`Boolean` **`isTrue`**;

This method returns a True value if the boolean is True, otherwise it returns a False value.

`Boolean` **`isFalse`**;

This is the opposite of the isTrue method.

`String` **`toString`**;
`String` **`toS`**;

This method returns a string representation of the boolean value. It will either be 'True' or 'False'.

`Integer` **`toInteger`**;
`Integer` **`toI`**;

This method returns a 1 if the boolean is True and 0 if the boolean is False.

`Number` **`toNumber`**;
`Number` **`toN`**;

This method returns a 1.0 if the boolean is True and 0.0 if the boolean is False.

`Number` **`toArray`**;
`Number` **`toA`**;

This method returns a one element array containing only the receiver object.

class Integer

The Integer class represents a 64 bit integer value, allowing integer values in the range of-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

Operators

The Integer class has instance methods to support the following operators.
+ - * / % ** == === <> != > >= < <= <=> & | ~ ^ << >>

Instance Methods

Integer result=intValue1 <=> intValue2

This method compares the receiver object with the value specified. It returns 0 if the objects are equal, it returns -1 if the receiver integer is less than the value object and it returns 1 if the receiver integer is greater than the value object.

Null **times**

This is an iterator method which will loop for the number of times in the value of the receiver. The value of the loop starts at 0. For example, the following lines loop for five iterations, writing out the values 0 through 4.

```
5.times do(i)
   writeln("Loop-"+i.toString);
end;
```

Null **upto**(Integer lastValue[,Integer incr]);

This is an iterator method which will loop starting at the number of times in the value of the receiver, incrementing the loop variable by incr (default value is 1) for each subsequent iteration, and ending when the loop variable exceeds the lastValue. For example, the following lines loop for five iterations, writing out the values 0 through 5.

```
0.upto(5) do(i)
   writeln("Loop-"+i.toString);
end;
```

The following lines will iterate 3 times. The loop variable (i) will have the values 0, 2, and 4.

```
0.upto(5,2) do(i)
   writeln("upto Loop by 2-"+i.toString);
end
```

Null **downto**(Integer lastValue[,Integer decr]);

This is an iterator method which will loop starting at the number of times in the value of the receiver, decrementing the loop variable by decr (default value is 1) for each subsequent iteration, and ending when the loop variable is less than the lastValue. For example, the following lines loop for five iterations, writing out the values 5 through 0.

```
5.downto(0) do(i)
   writeln("Loop-"+i.toString);
end;
```


The following lines will iterate 3 times. The loop variable (i) will have the values 5, 3, and 1.

```
5.downto(0,2) do(i)
   writeln("upto Loop by 2-"+i.toString);
end
```

`String` **`chr`**;

This method returns a one character string using the integer value as a character code.

```
var charStr=65.chr;
```

This example uses 65 as the character code and creates a one character string with the value of 'A'.

`Integer` **`inc`**;

This method returns an integer one higher than the receiver integer.

```
var intValue=65.inc;
```

The intVaue will be 66.

`Integer` **`dec`**;

This method returns an integer one lower than the receiver integer.

```
var intValue=65.dec;
```

The intVaue will be 64.

`Number` **`sqrt`**;

This method returns a number value(floating point) that is the square root of the receiver integer.

```
var nValue=64.sqrt;
```

The nValue will be 8.0.

`Boolean` **`isTrue`**;

This method returns a False value if the integer is equal to 0, otherwise it returns a True value.

`Boolean` **`isFalse`**;

This is the opposite of the isTrue method.

`String` **`toString`**;
`String` **`toS`**;

This method returns a string representation of the integer value.

`Integer` **`toInteger`**;
`Integer` **`toI`**;

This method returns the integer value.

`Number` **`toNumber`**;
`Number` **`toN`**;

This method returns a real number representation of the receiver object.

`Number` **`toArray`**;
`Number` **`toA`**;

This method returns a one element array containing only the receiver object.

class Number

The Number class represents a 64 bit floating point value, allowing values in the range of 1.7E +/- 308 (15 digits).

Operators

The Number class has instance methods to support the following operators.
+ - * / % ** == === <> != > >= < <= <=>

Instance Methods

Integer result=numValue1 <=> numValue2

This method compares the receiver object with the value specified. It returns 0 if the objects are equal, it returns -1 if the receiver number is less than the value object and it returns 1 if the receiver number is greater than the value object.

`Number` **`sqrt`**;

This method returns a number value(floating point) that is the square root of the receiver.

```
var nValue=64.0.sqrt;
```

The nValue will be 8.0.

`Boolean` **`isTrue`**;

This method returns a False value if the number is equal to 0.0, otherwise it returns a True value.

`Boolean` **`isFalse`**;

This is the opposite of the isTrue method.

`String` **`toString`**;
`String` **`toS`**;

This method returns a string representation of the number value.

`Integer` **`toInteger`**;
`Integer` **`toI`**;

This method returns an integer value, truncating the number value.

`Number` **`toNumber`**;
`Number` **`toN`**;

This method returns a floating point number.

`Number` **`toArray`**;
`Number` **`toA`**;

This method returns a one element array containing only the receiver object.

class String

The String class represents a set of Unicode (UTF-16) characters.

Note: The strUtils module (See "strUtils" on page 12.) contains more string methods. Also the Pcre methods allow search and replace using perl compatible regular expressions.

Operators

The Number class has instance methods to support the following operators.
+ * == === <> != > >= < <= <=> **[]**

Instance Methods

Integer result=strValue1 <=> strValue2

This method compares the receiver object with the value specified. It returns 0 if the objects are equal, it returns -1 if the receiver number is less than the value object and it returns 1 if the receiver number is greater than the value object.

`Boolean` **`isTrue`**;

This method returns a False value if the string has no characters, otherwise it returns a True value.

`Boolean` **`isFalse`**;

This is the opposite of the isTrue method.

`String` **`toString`**;
`String` **`toS`**;

This method returns the string.

`Integer` **`toInteger`**[(Integer base)];
`Integer` **`toI`**[(Integer base)];

This method returns an integer value, if the string contains characters within the range of the base. The optional base parameter has a default value of 10.

`Number` **`toNumber`**;
`Number` **`toN`**;

This method returns a floating point number from the value in the string.

`Number` **`toArray`**;
`Number` **`toA`**;

This method returns an array containing one element for each character in the string.

`String` String1 **`+`** String2;

The + operator concatenates the second string onto the end of the first string creating a new string.

`String` String1 **`*`** intValue;

The * operator copies the first string intValue times creating a new string.

`Integer` **`count`**;
`Integer` **`length`**;

This method returns an integer value containing the number of characters in the string.

`String` **`copy`**;

This method returns a copy of the string.

`Boolean` **`isAlpha`**;

This method returns a True value if the string has all alpha characters, otherwise it returns a False value.

`Boolean` **`isUpper`**;

This method returns a True value if the string has all uppercase alpha characters, otherwise it returns a False value.

`Boolean` **`isLower`**;

This method returns a True value if the string has all lowercase alpha characters, otherwise it returns a False value.

`Boolean` **`isDigits`**;

This method returns a True value if the string has all digit (0-9) characters, otherwise it returns a False value.

`Boolean` **`isXDigits`**;

This method returns a True value if the string has all hex digits (0-9,A-F,a-f) characters, otherwise it returns False.

`Boolean` **`isSpace`**;

This method returns a True value if the string has all space (HT,VT,FF,CR and space) characters, otherwise it returns a False value.

`Boolean` **`isPunct`**;

This method returns a True value if the string has all punctuation and special characters, otherwise it returns a False value.

`Boolean` **`isAlphaNum`**;

This method returns a True value if the string has all alpha or digit characters, otherwise it returns a False value.

`Boolean` **`isPrint`**;

This method returns a True value if the string has all printable characters, otherwise it returns a False value.

`Boolean` **`isGraph`**;

This method returns a True value if the string has all punctuation, alpha of or digit characters, otherwise it returns a False value.

`Boolean` **`isControl`**;

This method returns a True value if the string has all control characters, otherwise it returns a False value.

`Boolean` **`isAscii`**;

This method returns a True value if the string has all ascii characters, otherwise it returns a False value.

`Integer` **`charCodeAt`**[(Integer charPos)];

This method returns an integer value containing the character code of the character at the charPos position specified. The default charPos position is zero.

`String` **`insert`**(Integer charPosBefore,strValue[,strValue2, ...]);

This method inserts.each of the strValue items into the string before the specified charPos. A strValue item can be a string or it can be an integer character code. If the charPos value is negative, it is relative to the end of the string. The result is a new string. The original string is unchanged.

`String` **`remove`**(Integer startCharPos[,endCharPos]);

This method removes the characters starting with the startCharPos and ending with the endCharPos position. If the endCharPos is not specified, then it will remove characters from the startCharPos to the end of the string. If the charPos value is negative, it is relative to the end of the string.The result is a new string. The original string is unchanged.

`Integer` **`index`**(String findStr[,String opts]);

This method finds the substring within the string. The optional opts string allows you to choose one of two options to modify the action of the search. By default the search will begin at the character position 0 and will be case sensitive. If you use the 'I' value in the opts it will be a case insensitve search. If you use the 'R' option, it will start the search from the last character position and search from the right.

`String` **`subStr`**(Integer startCharPos[,endCharPos]);

This method selects the characters starting with the startCharPos and ending with the endCharPos position and returns a new string with just the characters in the range. It the endCharPos is not specified, then the range will be from the startCharPos to the end of the string. If the charPos value is negative, it is relative to the end of the string.

`String` **`subStrL`**(Integer startCharPos[,length]);

This method selects the characters starting with the startCharPos going for ‘length’ characters. It returns a new string with just the characters in the range. It the length is not specified, then the range will be from the startCharPos to the end of the string. If the charPos value is negative, it is relative to the end of the string.

`String` **`reverse`**;

This method returns a copy of the string with the character order reversed.The result is a new string. The original string is unchanged.

Null **`each`**

This is an iterator method which will loop for each character in the receiver string. This example will write each character in the string to the console

```
"My String".each do(cc)
   writeln("Char-"+cc);
end;
```

`String` **`onInit`**(String stringValue);

This is method is called by the String.**`make`** method when you create a string.

Note: You should not call this method directly. If you subclass the String class with one of your own and you need to do initialization for the subclass, you should have your own **`onInit`** method and call the **`super`** with the appropriate parameters to make sure the string is initialized.

Class Methods

`String String.`**`make`**(String stringValue);

This method creates a new string instance. After the string is created, it calls the **`onInit`** method of the new string using the arguments passed to it.

Note: This method should not be overridden. If you have a subclass that needs to do initialization, you should override the **`onInit`** instance method.

Note: You should not need to use this method, since strings are usually created using quotes or returned from other methods.

strUtils

strUtils is a set of extra string instance methods that are not included in the built-in string methods. To use these string methods, you must import the strUtils file as follows:

`import`('strUtils');

Instance Methods

`Integer String.`cmpSubStr(pSubStr,pStartPos,pSubStrPos,pCase);

This method compares a sub string to part of the receiver string. This method compares the substring specifed by pSubStr (starting at position pSubStrPos) to the receiver string starting at position pStartPos. This returns 0 if there is a match. It returns -1 of the receiver string is less than the substring and 1 otherwise.

`Integer String.`findChar(pChar,pStartPos=0,pCase=true);

This method finds char first occurrence of the character (pChar) in the string starting at position pStartPos and returning the position.

`Boolean String.`isPrefix(pPrefixStr,pCase=true);

This method returns true if the pPrefixStr string is a prefix of the receiver string.

`Boolean String.`isSuffix(pSuffixStr,pCase=true);

This method returns true if the pSuffixStr string is a suffix of the receiver string.

String String.leftTrim(pCharsToTrim);

This method returns a string with all the characters in the pCharsToTrim string removed from the left part of the string. If no parameters are specified, then all the control and space characters are removed from the beginning of the string..

String String.rightTrim(pCharsToTrim);

This method returns a string with all the characters in the pCharsToTrim string removed from the right part of the string. If no parameters are specified, then all the control and space characters are removed from the end of the string.

String String.trim(pCharsToTrim);

This method returns a string with all the characters in the pCharsToTrim string removed from the left and right part of the string. If no parameters are specified, then all the control and space characters are removed from the beginning and end of the string.

Examples:

The following illustrates some of the trim methods.

```
var vTestTrimStr='       88881234567898888';
writeln("TestStr="+vTestTrimStr.toS+'  Size='+vTestTrimStr.count.toS);
var vTrimStr=vTestTrimStr.leftTrim;
writeln("LTrim Spaces1="+vTrimStr.toS+'  Size='+vTrimStr.count.toS);
var vTrimStr=vTestTrimStr.leftTrim;
writeln("LTrim Spaces2="+vTrimStr.toS+'  Size='+vTrimStr.count.toS);
var vTrimStr=vTrimStr.leftTrim('8');
writeln("LTrim 8="+vTrimStr.toS+'  Size='+vTrimStr.count.toS);

var vTestTrimStr="88881234567898888   \t ";
writeln("TestStr="+vTestTrimStr.toS+'  Size='+vTestTrimStr.count.toS);
var vTrimStr=vTestTrimStr.rightTrim;
writeln("RTrim Spaces1="+vTrimStr.toS+'  Size='+vTrimStr.count.toS);
var vTrimStr=vTrimStr.rightTrim;
writeln("RTrim Spaces2="+vTrimStr.toS+'  Size='+vTrimStr.count.toS);
var vTrimStr=vTrimStr.rightTrim('8');
writeln("RTrim 8="+vTrimStr.toS+'  Size='+vTrimStr.count.toS);

var vTestTrimStr="    88881234567898888   \t ";
writeln("TestStr="+vTestTrimStr.toS+'  Size='+vTestTrimStr.count.toS);
var vTrimStr=vTestTrimStr.trim;
writeln("Trim Spaces1="+vTrimStr.toS+'  Size='+vTrimStr.count.toS);
var vTrimStr=vTrimStr.trim;
writeln("Trim Spaces2="+vTrimStr.toS+'  Size='+vTrimStr.count.toS);
var vTrimStr=vTrimStr.trim('8');
writeln("Trim 8="+vTrimStr.toS+'  Size='+vTrimStr.count.toS);
```

This results in the following console output.

```
TestStr=       88881234567898888  Size=24
LTrim Spaces1=88881234567898888  Size=17
LTrim Spaces2=88881234567898888  Size=17
LTrim 8=1234567898888  Size=13
TestStr=88881234567898888      Size=22
RTrim Spaces1=88881234567898888  Size=17
RTrim Spaces2=88881234567898888  Size=17
RTrim 8=8888123456789  Size=13
TestStr=    88881234567898888      Size=26
Trim Spaces1=88881234567898888  Size=17
Trim Spaces2=88881234567898888  Size=17
Trim 8=123456789  Size=9
```

String String.replaceChars(pFindChar,pReplaceChar,pCase=true);

This method returns a string with all occurrences of the pFindChar character with the pReplaceChar character.

String String.replaceSubStr(pStrToInsert,pStartIdx,pEndIdx);

This method returns a string with a new substring inserted into the place identified by the start and end indexes.

String String.replaceSubStrL(pStrToInsert,pStartIdx,pLen);

This method returns a string with a new substring inserted into the place identified by the start index and the length.

String String.leftJustify(pWidth,pFillChar=' ');

This method returns a string with the characters justified to the left of a string pWidth characters in length and the rest of the width padded with the fill character (pFillchar).

String String.rightJustify(pWidth,pFillChar=' ');

This method returns a string with the characters justified to the right of a string pWidth characters in length and the beginning of the width is padded with the fill character (pFillchar).

Examples:

The following illustrates some of the justify methods.

```
var vTestJustStr='12345';
writeln("TestJustStr="+vTestJustStr.toS+'  Size='+vTestJustStr.count.toS);
var vJustStr=vTestJustStr.leftJustify(30);
writeln("LJustStr ="+vJustStr.toS+'  Size='+vJustStr.count.toS);
var vJustStr=vTestJustStr.leftJustify(30,'0');
writeln("LJustStr0="+vJustStr.toS+'  Size='+vJustStr.count.toS);
var vJustStr=vTestJustStr.rightJustify(30);
writeln("RJustStr ="+vJustStr.toS+'  Size='+vJustStr.count.toS);
var vJustStr=vTestJustStr.rightJustify(30,'0');
writeln("RJustStr0="+vJustStr.toS+'  Size='+vJustStr.count.toS);
```


This results in the following console output.

```
TestJustStr=12345  Size=5
LJustStr =12345                                Size=30
LJustStr0=12345000000000000000000000000000  Size=30
RJustStr =                              12345  Size=30
RJustStr0=00000000000000000000000000012345  Size=30
```

String String.extractFileExtension;

This method assumes that the string contains a file (and optionally a path) name and returns the file extension part of the string.

String String.extractPathname;

This method assumes that the string contains a file (and optionally a path) name and returns the path name part of the string.

String String.extractFilename;

This method assumes that the string contains a file (and optionally a path) name and returns the file name part of the string.

String String.replaceFileExtension(pExt);

This method assumes that the string contains a file (and optionally a path) name and returns a string with the file extension changed tot he one specified (pExt).

Examples:

The following illustrates some of the file methods.

```
var vPathName='c:\path1\path2\filename.ext';

var vRExt=vPathName.extractFileExtension;
writeln("File Ext="+vRExt.toS);
var vPath=vPathName.extractPathname;
writeln("File path="+vPath.toS);
var vFile=vPathName.extractFilename;
writeln("File Name="+vFile.toS);
var vRExt=vPathName.replaceFileExtension;
writeln("ReplFile1 Ext="+vRExt.toS);
var vRExt=vPathName.replaceFileExtension('qaz');
writeln("ReplFile2 Ext="+vRExt.toS);
```

This results in the following console output.

```
File Ext=ext
File path=c:\path1\path2
File Name=filename.ext
ReplFile1 Ext=c:\path1\path2\filename
ReplFile2 Ext=c:\path1\path2\filename.qaz
```

class Array

The Array class represents a set of objects, numbered starting at 0. An array can hold any type of object. It can also be set to only hold one kind of object by setting the array data type. Array members can be set or retrieved using the index operator ([]) before an array value. You can also create an array using the bracket without an array value.

Note: The standard library (stdlib.evma) contains more instance Array methods. To use these, a script will need to import **arrayUtils**. For more information, see the Standard Library Reference.

Instance Methods

`Integer` **count**;
`Integer` **length**;

This method returns an integer value containing the number of objects in the array.

`Boolean` **isEmpty**;

This method returns a True value if the array has no objects, otherwise it returns a False value.

`classObject` **dataType**(classObject);
arr.**dataType**=classObject;

This method sets the object type of the array to the class object specified. After this is set, then only objects of the type set will be allowed in the array. Any other type will be rejected. This will not remove any objects previously inserted. To remove the type, set this value to Null.

`Array` **copy**;

This method returns a copy of the array. A new array is created and each member object is copied to it.

`Array` **clear**;

This method removes all the objects from this array.

`lastObject` **push**(object1[,object2, ...]);

This method adds one or more objects to the end of the array. If the data type is set and any of the objects are not of this type, then they will not be added and a TypeError exception will be generated. The last object added is returned.

`object` **pop**;

This method removes the object at the end of the array and returns the value. An OutOfRange exception is generated if the array is empty.

`object` **shift**;

This method removes the object at the front of the array and returns the value. An OutOfRange exception is generated if the array is empty.

`lastObject` **`unshift`**(object1[,object2, ...]);

This method adds one or more objects to the front of the array. If the data type is set and any of the objects are not of this type, then they will not be added and a TypeError exception will be generated. The last object added is returned.

`Integer` **`insert`**(Integer indexBefore,object1[,object2, ...]);

This method inserts.each of the object items into the array before the specified index. If the indexBefore value is negative, it is relative to the end of the array. The number of objects inserted is returned. If the data type is set and any of the objects are not of this type, then they will not be inserted and a TypeError exception will be generated.

`Integer` **`remove`**(Integer startIndex[,endIndex]);

This method removes the objects starting with the startIndex and ending with the endIndex position. If the endIndex is not specified, then it will remove the objects from the startIndex to the end of the array. If the index value is negative, it is relative to the end of the array. The number of objects removed is returned.

`Array` **`subSet`**(Integer startIndex[,endIndex]);

This method selects the objects starting with the startIndex and ending with the endIndex position and returns a new array with just the objects in the range. It the endIndex is not specified, then the range will be from the startIndex to the end of the array. If the index value is negative, it is relative to the end of the array.

`Array` **`subSetL`**(Integer startIndex[,length]);

This method selects the objects starting with the startIndex and going on for 'length' members and returns a new array with just the objects in the range. It the length is not specified, then the range will be from the startIndex to the end of the array. If the index value is negative, it is relative to the end of the array.

`Array` **`reverse`**;

This method returns a copy of the array with the object order reversed.

Null **`each`**

This is an iterator method which will loop for each object in the array.

```
arrayVar.each do(obj)
   writeln("Char-"+obj.className);
end;
```

This example will write the object type for each object in the array to the console.

`Array` **`onInit`**([classDef][,object1[,object2, ...]);
`Array` **`onInit`**(Integer num);

This is method is called by the Array.**`make`** method when you create an array or when an array is created using the bracket operators.

Note: You should not call this method directly. If you subclass the Array class with one of your own and you need to do initialization for the subclass, you should have your own **`onInit`** method and call the **`super`** with the appropriate parameters to make sure the array is initialized.

Class Methods

`Array Array.`**`make`**([classDef][,object1[,object2, ...]);
`Array Array.`**`make`**(Integer num);

This method creates a new array instance. After the object is created, it calls the **`onInit`** method of the new object using the arguments passed to it. If this method is called with no parameters, then an empty array will be created. If it is called with one integer parameter, then it will create an array with that number of members. If it is called with one more objects as parameters, then it will created an array with those parameters. If the first parameter is a classDef, then that classDef will be used as the type of the array. All the subsequent objects will be inserted into the array as long as they match the type.

Note: This method should not be overridden. If you have a subclass that needs to do initialization, you should override the **`onInit`** instance method.

class InputTextFile

The InputTextFile class represents a text file open for reading.

Instance Methods

`Integer` **`open`**(String fileName);

This method opens the text file and prepares it for reading. It returns 0 if the file opens without error.

`Null` **`close`**;

This method closes the file.

`Integer` **`length`**;

This method returns an integer value containing the length of the file in bytes.

`Integer` **`rewind`**;

This method resets the file pointer back to the beginning of the file.

`String` **`encoding`**[(String encoding)];

This method returns a string value with the encoding type of the file. When the file is opened, the first few bytes are examined. If there is a byte order mark, it is used to determine the encoding type. If not, the bytes are tested and an

encoding type is chosen. If you know the encoding type you can set it by including the `encoding` with this method after the file has been opened. The possible values are as follows:
`'Ansi' 'UTF-8' 'UTF-16' 'Utf-16LE' Utf-16BE'`

`String` **`readLine`**;

The method reads the next line from the text file and returns it in a string. The text line will be converted to UTF-16, if the file is Ansi or UTf-8.

`String` **`readChar`**;

The method reads the next character from the text file and returns it in a string. The character will be converted to UTF-16, if the file is Ansi or Utf8.

`Boolean` **`isEOF`**;

This method returns True if the last line has been read, otherwise it returns False.

class OutputTextFile

The OuputTextFile class represents a text file open for writing.

Instance Methods

`Integer` **`open`**(String fileName[,String mode]);

This method opens the text file and prepares it for writing. It returns 0 if the file opens without error. If the optional mode parameter is 'a' then the text will be appended to the existing file, otherwise an existing file will be cleared.

`Null` **`close`**;

This method closes the file.

`String` **`encoding`**[(String encoding)];

This method returns a string value with the encoding type of the file. The default value is 'Ansi'. If there are characters outside the range of ansi, they will be replaced by question marks('?'). If you wish to have a different encoding type you can set it by including the `encoding` with this method after the file has been opened. The possible values are as follows:
`'Ansi' 'UTF-8' 'UTF-16' 'Utf-16LE' Utf-16BE'`

For MS Windows, the UTF-16 and UTF-16LE encodings are equivalent. The UTF-16BE encoding has reversed byte order and is used in some Unix systems.

`Boolean` **`includeBom`**[(Boolean value)];

This method gets or sets the Boolean value indicating whether wish to have a byte order mark in the output file. A byte order mark at the beginning of the file identifies the encoding type.

`Integer` **`writeLine`**[(String outString)];

This method will write the specified string to the file converting to the specified encoding, if necessary. An end of line character will be written at the end. If no string is specified, then only the end of line character is written. The number of characters written is returned.

`Integer` **`write`**[(String outString)];

This method will write the specified string to the file converting to the specified encoding, if necessary. If no string is specified, then nothing happens. The number of characters written is returned.

class Object

The Object class is the base class of all user created objects. This class is created (along with a an instance of it) when a virtual machine (Evm) is created. When you use the import function.

class ETextInputFile

The ETextInputFile class represents a text file. It is a class that provides the basic functionality for a reading text characters from a disk file. You can specify the encoding of the text file (Ansi, Unicode, etc) or let the class determine it be examining the bytes in the file.

This class is not built-in, so it has be imported, as follows:

```
import('bsio\ETextInputStream');
```

Instance Methods

`Integer` **`open`**(String fileName);

This method opens the specified text file and prepares it for reading. It returns 0 if the file opens without error.

`Null` **`close`**;

This method closes the file.

`Integer` **`rewind`**;

This method resets the byte stream back to the beginning.

`Boolean` **`isEOF`**; (or **`isEOS`**)

This method returns **`true`** if it is at the end of the file and no more text characters are available, otherwise it returns **`false`**;

`Integer` **`streamSize`**;

This method returns the size of the file in bytes.

`Boolean` **`isPresent`**;

This method returns **`true`** if the file is present (the open method succeeded), otherwise it returns **`false`**;

`Integer` **`encoding`**[(String or Integer `pEncoding`)];

This method returns an integer value with the encoding type of the file. When the file is opened, the first few bytes are examined. If there is a byte order mark, it is used to determine the encoding type. If not, the bytes are tested and an encoding type is chosen. If you know the encoding type you can set it by including the `pEncoding` with this method after the file has been opened. This value can be an integer value (as specified in the class properties) or you can use one of the following string values:
`'Ansi' 'UTF-8' 'dbcs' 'UTF-16' 'Utf-16LE' Utf-16BE'`

If the encoding value is 'dbcs' you may also need to specify a code page.

`Integer` **`codePage`**[(String or Integer `pCodePage`)];

This method returns an integer value with the code page of the text file. When the encoding is chosen, the code page is also chosen. If you know the code page you can set it by including the `pCodePage` with this method after the file has been opened. This is usually only necessary when the encoding type is 'dbcs'. This value can be an integer value (as specified in the class properties) or you can use one of the following string values:
`'acp' 'eom' 'UTF-8' 'UTF-16' 'Utf-16LE' Utf-16BE' 'shiftjis' 'gb2312' 'hangul' 'big5' 'johab'`

`String` **`getNextChar`**;

This method returns the next available text character.

`String` **`peekNextChar`**;

This method returns the next available text character, but does not move the the character pointer to the next character.

`String` **`getNextLine`**;

This method returns the next text line.

Class Properties

The following values indicate the encoding type of the input text file

```
ETextInputFile.TCS_Ansi;     // Standard Ms Windows Character set
ETextInputFile.TCS_UTF8;     // Unicode
ETextInputFile.TCS_DBCS;     // Double Byte Character Set
ETextInputFile.TCS_UTF16LE;  // Ms Windows,x86,MacOs on x86
ETextInputFile.TCS_UTF16BE;  // Sparc,Solaris,MacOs on PowerPc or 680xxx
```

The following values indicate the code page of the input text file. You only need to specify the code page value when the encoding type is `TCS_DBCS`.

```
ETextInputFile.CP_DBCS_ShiftJIS; // Japanese
ETextInputFile.CP_DBCS_GB2312; // Simplefied Chinese
ETextInputFile.CP_DBCS_Hangul; // Korean
ETextInputFile.CP_DBCS_BIG5; // Chinese(Taiwan,Hong Kong, Sar, PRC
ETextInputFile.CP_DBCS_Johab; // Korean
ETextInputFile.CP_UTF8;
ETextInputFile.CP_ACP;
ETextInputFile.CP_UTF16LE;
ETextInputFile.CP_UTF16BE;
ETextInputFile.CP_OEMCP;
```

Example:

The following opens and reads each line in a text file (indicated by the vFileName value) and writes the each line to the console.

```
var vTextInputFile=bsio.ETextInputFile.make
var vRetCode=vTextInputFile.open(vFileName);
if vRetCode==0
   var vDone=false;
   while !vDone
      var vCurrLine = vTextInputFile.getNextLine;
      if vCurrLine.count<1 and vTextInputFile.isEOF
         vDone=true;
      else
         writeln("Line="+vCurrLine);
      end;
   end;
   vTextInputFile.close;
end
```

If the text file is UTF-8 and there is no byte order mark, you can set the encoding by placing the following line after the open method as follows:

```
vTextInputFile.encoding('utf-8');
```

If the text file is Chinese text, you can set the encoding by placing the following lines after the open method as follows:

```
vTextInputFile.encoding('dbcs');
vTextInputFile.codePage('big5');
```

class ETextOutputFile

The ETextOutputFile class represents a text file open for writing.

This class is not built-in, so it has be imported, as follows:

```
import('bsio\ETextOutputStream');
```

Instance Methods

`Integer` **`open`**(String fileName[,appendFlag=false]);

This method opens the text file and prepares it for writing. If the appendFlag is false, it creates a new file overwriting any previous contents. If the appendFlag is true, then it wll open the file for output, keeping the current contents and adding any data to the end of the file. It returns 0 if the file opens without error.

`Null` **`close`**;

This method closes the file.

`Boolean` **`isPresent`**;

This method returns **`true`** if the file is present (the open method succeeded), otherwise it returns **`false`**;

`String` **`encoding`**[(String `pEncoding`)];

This method returns an integer value with the encoding type of the file. The default value is 'Ansi'. If there are characters outside the range of ansi, they will be replaced by question marks('?'). If you wish to have a different encoding type you can set it by including the `pEncoding` with this method after the file has been opened, but before any characters are written.This value can be an integer value (as specified in the class properties) or you can use one of the following string values:
`'Ansi' 'UTF-8' 'UTF-16' 'Utf-16LE' Utf-16BE' 'dbcs'`

If you specify 'dbcs' for the encoding, you should specify a code page value to indicate which language to use

For MS Windows, the UTF-16 and UTF-16LE encodings are equivalent. The UTF-16BE encoding has reversed byte order and is used in some Unix systems.

`Integer` **`codePage`**[(String or Integer `pCodePage`)];

This method returns an integer value with the code page of the text file. When the encoding is chosen, the code page is also chosen.You can set it by including the `pCodePage` with this method after the file has been opened. This is usually only necessary when the encoding type is 'dbcs'. This value can be an integer value (as specified in the class properties) or you can use one of the following string values:
`'acp' 'eom' 'UTF-8' 'UTF-16' 'Utf-16LE' Utf-16BE' 'shiftjis' 'gb2312' 'hangul' 'big5' 'johab'`

`null` **`putBOM`**;

This method writes a byte order mark to the output file. A byte order mark at the beginning of the file identifies the encoding type. This is only available for unicode types (Utf-8, Utf16). This should be called after the output text is opened and after the encoding and codePage have been set.

> **Note:** This should not be called if you open the file with the appendFlag set to true.

`null` **`putString`**[(String outString)];

This method will write the specified string to the file converting to the specified encoding, if necessary. An end of line character will be written at the end. If no string is specified, nothing happens.

`null` **putLine**[(String outString)];

This method will write the specified string to the file converting to the specified encoding, if necessary. An end of line character will be written at the end. If no string is specified, then only the end of line character is written.

`null` **putEOL**;

This method will write the end of line character.

Class Properties

The following values indicate the encoding type of the output text file

```
ETextOutputFile.TCS_Ansi;     // Standard Ms Windows Character set
ETextOutputFile.TCS_UTF8;     // Unicode
ETextOutputFile.TCS_DBCS;     // Double Byte Character Set
ETextOutputFile.TCS_UTF16LE;  // Ms Windows,x86,MacOs on x86
ETextOutputFile.TCS_UTF16BE;  // Sparc,Solaris,MacOs on PowerPc or 680xxx
```

The following values indicate the code page of the output text file. You only need to specify the code page value when the encoding type is `TCS_DBCS`.

```
ETextOutputFile.CP_DBCS_ShiftJIS; // Japanese
ETextOutputFile.CP_DBCS_GB2312; // Simplefied Chinese
ETextOutputFile.CP_DBCS_Hangul; // Korean
ETextOutputFile.CP_DBCS_BIG5; // Chinese(Taiwan,Hong Kong, Sar, PRC
ETextOutputFile.CP_DBCS_Johab; // Korean
ETextOutputFile.CP_UTF8;
ETextOutputFile.CP_ACP;
ETextOutputFile.CP_UTF16LE;
ETextOutputFile.CP_UTF16BE;
ETextOutputFile.CP_OEMCP;
```


Example:

The following opens a file (indicated by the vFileName value) for output and writes each line in an array to it .

```
var vFileName='c:\Test\MyOutputFile.txt';
var vList=Array("Double, double, toil And trouble,",
                "Fire burn And cauldron bubble.",
                "By the pricking of my thumbs,",
                "Something wicked this way comes.");

var vFile=bsio.ETextOutputFile.make;
vFile.encoding='utf8';
var vStat=vFile.open(vFileName);
vFile.putBOM;
if vStat==0
   var vCount=vList.count;
   vList.each do(vLine)
      vFile.putLine(vLine);
   end;
   vFile.close;
else
   writeln("Open fail-"+vStat.toS);
end
```

If you wish to write the file with a double byte Japanese text encoding, replace the encoding line with the following:

```
vFile.encoding='dbcs';
vFile.codePage='shiftjis';
```

class ETempFiles

The ETempFiles class allows you to create one or more temporary files. There are times when you would like to temporarily save data that is too big to fit into memory. When this happens, programmers will usually write the data to a temporary disk file, then read it back in later. This happens often enough that Ms Windows has a special folder to put these temporary files (C:\Temp). This class helps you create unique temporary files names.

You can optionally specify a folder name (usually a company name or some unique string) and a SubFolder name (usually a Project/Product code). These are not necessary, but might be useful if you keep the around to check the results manually.

This class is not built-in, so it has be imported, as follows:

```
import('ETempFiles');
```

Instance Methods

String folder[(String folderName)];

This method gets and optionally sets the majorFolder name.

String subFolder[(String subFolderName)];

This method gets and optionally sets the subFolder name.

String getTempFileName[(String prefix[,fileExtension])];

This method returns the name of a newly created temporary file. If you specify a prefix string (up to 3 characters), it will put this string at the beginning of the temporary file name. If you specify an extension, it will use this as the extension of the temporary file.

The following example creates a ETempFiles instance object, then creates two temp files. You should delete these files when you are finished with them.

```
var vTempFileMaker=ETempFiles.make('MyCompany','MyProject');
var vTemp1=vTempFileMaker.getTempFileName;
...
var vTemp2=vTempFileMaker.getTempFileName('XXX','txt');
```

class Methods

ETempFiles.make[(folderName[,subFolderName])];

This creates an instance of a ETempFiles object.

class ETempFileMgr

The ETempFileMgr class is derived from ETempFiles. Since it is a subclass, it has all the properties and methods of ETempFiles. Its extra functionality is that it keeps track of all the temp file names created and it adds an additional method (deleteAllTempFiles) that will delete all the temp files that were created with this object. This means that you do not have to keep track of any temp files created. You just have to call the deleteAllTempFiles method when you are finished.

This class is not built-in, so it has be imported, as follows:


```
import('ETempFiles');
```

Instance Methods

String getTempFileName[(String prefix[,fileExtension])];

This method returns the name of a newly created temporary file. If you specify a prefix string (up to 3 characters), it will put this string at the beginning of the temporary file name. If you specify an extension, it will use this as the extension of the temporary file. The name of the file will be saved in a list.

String deleteAllTempFiles;

This method deletes all the temporary files created with this object up to this point.

The following example creates a ETempFileMgr instance object, then creates two temp files. Later, it calls the deleteAllTempFiles method to delete these two files.

```
var vTempFileMgr=ETempFileMgr.make;
var vTemp1=vTempFileMgr.getTempFileName;
...
var vTemp2=vTempFileMgr.getTempFileName('Qaz','txt');
...
...
vTempFileMgr.deleteAllTempFiles;
```

class Methods

ETempFileMgr.make[(folderName[,subFolderName])];

This creates and returns an instance of a ETempFileMgr object.

class utl.SysFiles

The SysFiles class provides system information. Since there is only one operating system, you do not have to create an instance of this class. All the methods are class methods.

There are 5 methods that return operating system id values (version number, etc.).

Windows creates a special folder for each user. This is used to store files specific to that user. Applications use this to store configuration and preference files. The actual folder name depends on the operating system version and the user name. To make it easy to read and write these Configuration/Preference files the SysFiles class has two methods to return full file names (getUserAppDataFileName and getUserProfileFileName). All you have to do is to supply relative path, usually in the following form:

```
\CompanyName\ProductName\ProductVersion\configFile.txt
```

The getUserAppDataFileName will return the full path. For example,

```
var vRelFileName='\Company\Product\V1\config.txt';
var vMyConfigFile=SysFiles.getUserAppDataFileName(vRelFileName);

For Windows XP or lower, vMyConfigFile will be:
c:\Documents and Settings\<username>\Application Data\Company\Product\V1\config.txt
For Windows Vista or greater, vMyConfigFile will be:
c:\Users\<username>\AppData\Roaming\Company\Product\V1\config.txt
```

The getUserProfileFileName does something similar with the Profile folder.

There are also two methods that do the same with the Global folder.

This class is not built-in, so it has be imported, as follows:

```
import('utl\SysFiles');
```

class Methods

Integer utl.SysFiles**.majorVersion;**

This returns the operating system major version number.

Integer utl.SysFiles**.minorVersion;**

This returns the operating system minor version number.

Note: The major and minor versions identify the operating system (6.2=Windows 8, 6.1=Windows 7, 6.0=Windows Vista, 5.1=Windows XP and 5.0=Windows 2000).

Integer utl.SysFiles**.buildNumber;**

This returns the operating system build number.

Integer utl.SysFiles**.platformId;**

This returns the operating system platformId number.

String utl.SysFiles**.versionString;**

This returns the operating system version string (e.g. Service pack 2).

String utl.SysFiles.**getUserAppDataFileName**(pFileStr);

This returns the file name string if the file is located in the User Application data folder.

String utl.SysFiles.**getUserProfileFileName**(pFileStr);

This returns the file name string if the file is located in the User Profile folder

String utl.SysFiles.**getGlobalAppDataFileName**(pFileStr);

This returns the file name string if the file is located in the Global Application data folder

String utl.SysFiles.**getGlobalProfileFileName**(pFileStr);

This returns the file name string if the file is located in the Global Profile folder

Examples:

The following illustrates some of the sys files methods.

```
import('sysFiles');
var vFullFile=utl.SysFiles.getUserAppDataFileName('\ElmSoft\MyProject\MyConfig.cfg');
writeln(vFullFile.toS);
var vFullFile=utl.SysFiles.getGlobalAppDataFileName('\ElmSoft\MyProject\MyConfig.cfg');
writeln(vFullFile.toS);
var vFullFile=utl.SysFiles.getUserProfileFileName('\ElmSoft\MyProject\MyConfig.cfg');
writeln(vFullFile.toS);
var vFullFile=utl.SysFiles.getGlobalProfileFileName('\ElmSoft\MyProject\MyConfig.cfg');
writeln(vFullFile.toS);

var vMajorVersion=utl.SysFiles.majorVersion;
var vMinorVersion=utl.SysFiles.minorVersion;
var vBuild=utl.SysFiles.buildNumber;
var vPlatformId=utl.SysFiles.platformId;
var vVersionString=utl.SysFiles.versionString;

writeln("OsVersion("+vMajorVersion.toS+","+
         vMinorVersion.toS+
         ") Build="+vBuild.toS+
         "  PlatformId="+vPlatformId.toS+
         "  version="+vVersionString.toS);
```

This results in the following console output.

```
C:\Documents and Settings\<user>\Application Data\ElmSoft\MyProject\MyConfig.cfg
C:\Documents and Settings\All Users\Application Data\ElmSoft\MyProject\MyConfig.cfg
C:\Documents and Settings\<user>\ElmSoft\MyProject\MyConfig.cfg
C:\Documents and Settings\All Users\ElmSoft\MyProject\MyConfig.cfg
OsVersion(5,1) Build=2600  PlatformId=2  version=Service Pack 3
```

class Pcre

The Pcre class provides the ability to search strings using perl compatible regular expressions.

This class is an external class and has to be imported, as follows:

```
import('Pcre');
```

class Methods

`String` **`subStr`**(pRegExStr,pSearchStr,pOptions=0,pOffset=0);

This method finds a sub string corresponding to the regular expression (pRegExStr) in the source string (pSearchStr) using the options (pOptions) starting at the specified offset (pOffset). This returns the substring itself, or null if not found. Both the pOptions and pOffset parameters are optional.

`Pcre.PcreRange getRange`(pRegExStr,pSearchStr,pOptions=0,pOffset=0);

This method finds a sub string corresponding to the regular expression (pRegExStr) in the source string (pSearchStr) using the options (pOptions) starting at the specified offset (pOffset). This returns a Pcre.PcreRange object which contains a start position and a length value. It returns null if not found. Both the pOptions and pOffset parameters are optional.

`Pcre.PcreRange equate`(pRegExStr,pSearchStr,pOptions=0,pOffset=0);

This method returns true if the regular expression (pRegExStr) completely matches the search string (pSearchStr) using the options (pOptions) starting at the specified offset (pOffset).. It returns false if not a complete match. Both the pOptions and pOffset parameters are optional.

`Pcre.PcreMatchList search`(pRegExStr,pSearchStr,pOptions=0,pOffset=0);

This method finds a sub string corresponding to the regular expression (pRegExStr) in the source string (pSearchStr) using the options (pOptions) starting at the specified offset (pOffset). This returns a Pcre.PcreMatchList object which contains a list of matches (if any). The Pcre.PcreMatchList object is a kind of Array of Pcre.PcreMatch objects. A Pcre.PcreMatch object contains the match itself (a Pcre.Pcre.Range object) and optionally a set of subpattern objects (Pcre.PcreRange objects), one for each sub pattern matched. Both the pOptions and pOffset parameters are optional.

It returns null if not found.

`Pcre.PcreCompPattern makeCompiledRegEx`(pRegExStr,pOptions=0);

The makeCompiledRegEx method compiles a regular expression and returns the compiled regular expression object (Pcre.PcreCompPattern). This object can be used to search strings. If you want to use the same regular expression many times, it is more efficient to compile it first, then use it instead of compiling each time.The pOptions parameter is optional.

`String replace`(pRegExStr,pSearchStr,pOptions=0,pOffset=0,pReplace);

This method calls the **`search`** method using the first four parameters and if there are one or more matches, it replaces the matched text with the pReplace value. Both the pOptions and pOffset parameters are optional. Use empty parameters if necessary.

It returns the original search string if no matches are found.

class Properties

`Integer cmLastError`;

This property contains the error code of the last method called.

`String cmLastErrorMsg`;

This property contains the error message of the last method called.

Regular Expressions

A regular expression can be plain or compiled. A plain regular expression is a simple string. A compiled regular expression is an optimized version of it. A regular expression must be compiled to perform searches. If you specify a string as the regular expression, the Pcre object will compile it for you. if you are only going to use the regular expression once, then this is no problem. However, if you are using the same regular expression many times, it is more efficient to compile it once, rather than for each search.

The syntax of Regular Expressions is a topic that is covered by books and is to extensive to be covered here. For details of the syntax of regular expressions check online at www.pcre.org or perl documentation.

Pcre options

Prce options is a string value which provides more information about how a regular expression run is evaluated. The following values are allowed:

Table 1: Pcre Options

Option Character	Description
i	Ignore Case **(When Compiling)**
m	Treat source string as multiple lines **(When Compiling)**
s	Treat source string as a single line **(When Compiling)**
x	Use extended regular expressions **(When Compiling)**
A	Force pattern anchoring
E	$ not to match newline at end **(Evaluate Only)**
f	Force matching to be before newline
C	Compile automatic callouts **(When Compiling)**
U	Invert greediness of quantifiers **(When Compiling)**
Z	Include subpatterns **(Evaluate Only)**
g	Global (find all occurrences **(Evaluate Only)**

Table 1: Pcre Options

Option Character	Description
G	Global (find all occurrences) start at last offset **(Evaluate Only)**
B	This specifies that the first character is not to be treated as the beginning of the line **(Evaluate Only)**
Q	This specifies that the last character is not to be treated as the end of the line **(Evaluate Only)**
<cr>,<lf>,<crlf>, <anycrld>,<any>	This specifies the line termination character(s).
<JS>	This specifies JavaScript compatibility.

class Pcre.PcreCompPattern

The Pcre.PcreCompPattern class represents a compile regular expression. This object is returned from the Pcre.makeCompiledRegEx method. The instance methods here correspond to the class methods of the Pcre object. In fact, the Pcre object's class methods compile the specified regular expression and then it calls these methods to perform the search.

This class is an external class and has to be imported, as follows:

```
import('Pcre');
```

Instance Methods

`String` **`subStr`**(pSearchStr,pOptions=0,pOffset=0);

This method finds a sub string corresponding to this regular expression in the source string (pSearchStr) using the options (pOptions) starting at the specified offset (pOffset). This returns the substring itself, or null if not found. Both the pOptions and pOffset parameters are optional.

`Pcre.PcreRange getRange`(pSearchStr,pOptions=0,pOffset=0);

This method finds a sub string corresponding to this regular expression in the source string (pSearchStr) using the options (pOptions) starting at the specified offset (pOffset). This returns a Pcre.PcreRange object which contains a start position and a length value. It returns null if not found. Both the pOptions and pOffset parameters are optional.

`Pcre.PcreRange equate`(pSearchStr,pOptions=0,pOffset=0);

This method returns true if this regular expression completely matches the search string (pSearchStr) using the options (pOptions) starting at the specified offset (pOffset).. It returns false if not a complete match. Both the pOptions and pOffset parameters are optional.

`Pcre.PcreMatchList search`(pSearchStr,pOptions=0,pOffset=0);

This method finds a sub string corresponding tothis regular expression in the source string (pSearchStr) using the options (pOptions) starting at the specified offset (pOffset). This returns a Pcre.PcreMatchList object which contains a list of matches (if any). The Pcre.PcreMatchList object is a kind of Array of Pcre.PcreMatch objects. A

Pcre.PcreMatch object contains the match itself (a Pcre.Pcre.Range object) and optionally a set of subpattern objects (Pcre.PcreRange objects), one for each sub pattern matched. Both the pOptions and pOffset parameters are optional.

It returns null if not found.

String replace(pSearchStr,pOptions=0,pOffset=0,pReplace);

This method calls the **search** method using the first three parameters and if there are one or more matches, it replaces the matched text with the pReplace value. Both the pOptions and pOffset parameters are optional.

It returns the original search string if no matches are found.

class Pcre.PcreRange

The Pcre.PcreRange class represents a range of text from a search string created after doing a Pcre regular expression search or getRange.

This class is an external class and has to be imported, as follows:

import('Pcre');

Instance Methods

Integer **start**;

This method returns the start position of the range.

Integer **len**;

This method returns the length of the range.

String **getString**(pSearchStr);

This method returns the substring from the match, if the pSearchStr value is the same string as in the search that created this PcreRange object.

String replace(pOrigString,pReplaceStr);

This method returns an updated string after replacing the text in the range with the pReplaceStr string.

class Pcre.PcreMatchList

The Pcre.PcreMatchList class is returned when the **search** method is successful. This is an array of Pcre.PcreMatch objects, one object for each match. Since this class is derived from Array, you can get the number of matches using the **count** property.

This class is an external class and has to be imported, as follows:

```
import('Pcre');
```

Instance Methods

String replace(pOrigString,pReplace);

This method returns the updated string after replacing the text from all the matches with the pReplace value.See page 34 for more information and replace strings.

class Pcre.PcreMatch

The Pcre.PcreMatch class is created as the result of successful **search**, as a member of the Pcre.PcreMatchList array. This array will have one or more Pcre.PcreMatch objects. Each Pcre.PcreMatch objects has two data items, a Pcre.PcreRange object representing the entire match range and an array (optional) of subpattern ranges (also Pcre.PcreRange objects). This will be include if subpatterns exist in the regular expression and the Z option was specified.

This class is an external class and has to be imported, as follows:

```
import('Pcre');
```

Instance Methods

Pcre.PrecRange match;

This method returns the Pcre.PcreRange object representing the entire match range.

Array subPatterns;

This method returns the array of subpattern (Pcre.PcreRange objects) ranges.

Integer getSubPatternCount;

This method returns the number of subpatterns in the match.

Pcre.PrecRange getSubPattern(pIdx);

This method returns the Pcre.PcreRange object representing the subpattern indicated by the pIdx integer value.

String replace(pOrigString,pReplace);

This method returns the updated string after replacing the text from all the matches with the pReplace value.

Note: The replace value can be a simple string value or it can contain match or subpattern codes. If the text includes \1 it will substitute the text from the first matching subpattern. \2 will use the 2nd subpattern and so on. \0 will substitute the entire match range.

class utl.EReadZip

The EReadZip class represents a standard zip file used to reading.

Note: This zip class uses the deflate method (the standard LZ77-based algorithm) to compress files. It can also uncompress files of this type. However, if a zip file is created with another program using a more sophisticated compression method, then class might not be able to uncompress the files.

This class is not built-in, so it has be imported, as follows:

```
import('utl\EZipFile');
```

Instance Methods

Integer openZip(String pFileName);

This method opens the zip file identified by pFileName.

Integer closeZip;

This method closes the zip file.

Array loadDir;

This method returns the directory of the zip file. This is an Array of EReadZipDirItem objects (see page 37), each object represents an entry in the zip directory.

EReadZipDirItem findDirItem(pFileName);

This method returns the EReadZipDirItem object (see page 37), indicated by the pFileName parameter.

Integer extract[(pFileList[,pOutDir[,pOptions[,pPassword]]])];

This method extracts the files from the pFileList parameter into the specified folder (pOutDir), using the options (if any) specified in the pOptions parameter and the optional password (pPassword). If no password is specified, it is assumed that no password was used when the zip file was created. The pOptions (if specified) is a **String** value containing one or more option names separated by semicolons. The possible options are 'nopath' indicating to extract without using the path name, 'noovr' indicating not to overwrite existing files and 'keepdt' which means to keep the original date-time. If no output folder is specified, the files will be extracted to the same folder as the zip file. The pFileList can be a string **Array** containing the names of the files to extract. It can also be a **String** value specifying one file to extract or it can be omitted which means to extract all the files in the zip file.

The following opens a zip file, then looks for the file mimetype. If it is found it writes the directory information to the console.

```
var vReadZip=utl.EReadZip.make;
var vZipFileName='C:\MyFolder\MyZipFile.zip';

var vRcc=vReadZip.openZip(vZipFileName);
if vRcc==0
   var vItem1=vReadZip.findDirItem('mimetype');
   if vItem1
      writeln("Directory entry FOUND-%s  Size(%d)".formatString(vItem1.fileName,
         vItem1.compSize));
   else
      writeln("mimetype NOT FOUND");
   end;
end;
```

class Methods

utl.EReadZip.make;

This creates an instance of a EReadZip object.

These class methods can be called without creating an instance of the object.

Array getFileList(pZipFileName);

This method returns the directory of the specified (pZipFileName) zip file. This is an Array of EReadZipDirItem objects (see page 37), each object represents an entry in the zip directory.

EReadZipDirItem getFileInfo(pZipFileName,pFileName);

This method returns the EReadZipDirItem object (see page 37),from the specified (pZipFileName) zip file indicated by the pFileName parameter.

Integer extract(pZipFileName[,pFileList[,pOutDir[,pOptions[,pPassword]]]]);

This method extracts the files (from the specified (pZipFileName) zip file) from the pFileList parameter into the specified folder (pOutDir), using the options (if any) specified in the pOptions parameter and the optional password (pPassword). If no password is specified, it is assumed that no password was used when the zip file was created. The pOptions (if specified) is a **String** value containing one or more option names separated by semicolons. The possible options are 'nopath' indicating to extract without using the path name, 'noovr' indicating not to overwrite existing files and 'keepdt' which means to keep the original date-time. If no output folder is specified, the files will be extracted to the same folder as the zip file. The pFileList can be a string **Array** containing the names of the files to extract. It can also be a **String** value specifying one file to extract or it can be omitted which means to extract all the files in the zip file.

Example:

This example extracts all the files in the MyZipfile to the MyOutFolder folder

```
var vZipFileName='C:\MyFolder\MyZipFile.zip';
var vOutDir='C:\MyOutFolder';
var vRcc=utl.EReadZip.extract(vZipFileName,null,vOutDir);
if vRcc>=0
   writeln(vRcc.toS+" files extracted");
else
   writeln('extract error-'+vRcc.toS);
end;
```

class utl.EReadZipDirItem

The EReadZipDirItem class represents an entry is a zip file directory.

This class is not built-in, so it has be imported, as follows:

```
import('utl\EZipFile');
```

Instance Methods

String fileName;

This method returns the filename.

String compMethod;

This method returns the compression method used when the file was added to the zip file.

String dateTime;

This method returns the file date time string.

Integer compSize;

This method returns the compressed size of the file.

Integer uncompSize;

This method returns the uncompressed size of the file.

Integer crc32;

This method returns the crc32 code of the file.

class utl.EWriteZip

The EWriteZip class represents a standard zip file used to writing.

> **Note:** This zip class uses the deflate method (the standard LZ77-based algorithm) to compress files. It can also uncompress files of this type. However, if a zip file is created with another program using a more sophisticated compression method, then class might not be able to uncompress the files.

This class is not built-in, so it has be imported, as follows:

```
import('utl\EZipFile');
```

Instance Methods

Integer openZip(String pFileName[,pAppend=false]);

This method opens the zip file identified by pFileName for output. If the file is not present, it is created. If the file is present and the pAppend is set to true, any files added will be put at the end of the file, otherwise, the contents of the zip file will be erased.

Integer closeZip;

This method closes the zip file.

Struct addFiles[(pFileList[,pNewFileNames[,pOptions[,pCompLevel[,pPassword]]]])];

This method adds the files in the `pFileList` parameter using the corresponding names (`pNewFileNames`, if specified), using the options (if any) specified in the `pOptions` parameter, compressed to the `pCompLevel` level and the optional password (`pPassword`). If no password is specified, then no password will be used. The `pOptions` (if specified) is a **String** value containing one or more option names separated by semicolons. The possible options are 'nopath' indicating that only the file name is stored in the directory, 'newfile' indicating that a new zip file should be created and 'utf8' which means the directory filename will be stored in utf-8 format. The `pCompLevel` indicates the level of compression (0-9). The higher the number the better compression, but it will run slower. Unless you have special requirements it is better to use the default value. The pFileList can be a string **Array** containing the names of the files to add. It can also be a **String** value specifying one file to add. If the `pNewFileNames` parameter is specified, it represents the names to be used in the zip (if different than the names of the files on disk). These should correspond the the `pFileList` parameter.

This method returns a Struct object with the following members.

```
fileCount     The number of files added
failedCount   The number of files that failed (for some reason) to be added
dirCount      The number of directories added
missingCount  The number of files that did not exist
errFlag       The error flag
```


The following creates a zip file then writes a set of files to it.

```
var vWriteZip=utl.EWriteZip.make;
var vZipFileName='C:\MyFolder\MyOutZipFile.zip';
var vList=['c:\MyFiles\File1.txt','c:\MyFiles\File2.txt','c:\MyFiles\File3.txt',
       'c:\MyFiles\File4.txt'];

var vRcc=vWriteZip.openZip(vZipFileName);
if vRcc==0
   var vRetStruct=vWriteZip.addFiles(vList;
   if vRetStruct
      writeln("Results\n  added files="+vRetStruct.fileCount.toS);
   else
      writeln("Files NOT Added");
   end;
end;
```

Struct addDir(pFolderName[,pWildCard[,pOptions[,pCompLevel[,pPassword]]]]);

This method is similar to the addFiles method, except that you specify a folder name (and optionally a wildcard string) instead of a list of files and you cannot change the names of the files.

This method returns the same Struct object as the addFiles method.

class Methods

utl.EWriteZip.make;

This creates an instance of a EWriteZip object.

These class methods can be called without creating an instance of the object.

Struct addFiles(pZipFileName[,pNewFileNames[,pOptions[,pCompLevel[,pPassword]]]]));

This method works the same way that the corresponding instance method, except that you have to specify the zip file name and you do not make an instance of the object.

Struct addDir(pZipFileName,pFolderName[,pWildCard[,pOptions[,pCompLevel[,pPassword]]]]);

This method works the same way that the corresponding instance method, except that you have to specify the zip file name and you do not make an instance of the object.

Example:

This example adds all the text files in the MyInFolder to the MyOutZipFile.

```
var vZipFileName='C:\MyFolder\MyOutZipFile.zip';
var vInDir='C:\MyInFolder';
var vStruct=utl.EWriteZip.addDir(vZipFileName,vInDir,'*.txt');
if vStruct
   writeln("Results\n  added files="+vStruct.fileCount.toS);
else
   writeln('add error');
end;
```

class utl.ELookup

The ELookup class represents a look up table.

This class is not built-in, so it has be imported, as follows:

```
import('utl\ELookup');
```

Instance Methods

Object list[(pArray)];

This method gets and/or sets the array of arrays used in the search. This is usually specified when the object is created.

Object find(pKeyValue,pKeyCol=0,pRetCol=-1);

This method finds the entry specified by pKeyValue in column pKeyCol. If pKeyValue is found, then it returns the value in the pRetCol column. If pRetCol is not specified, then it returns the entire array value. If it is not found, then **null** is returned.

Integer findIndex(pKeyValue,pKkeyCol=0);

This method finds the entry specified by pKeyValue in column pKeyCol. If it is found, the index of the found entry is returned, otherwise -1 is returned.

Object findString(pKeyValue,pKeyCol=0,pRetCol=-1);

This method finds the string entry specified by pKeyValue (case sensitive) in column pKeyCol. If pKeyValue is found, then it returns the value in the pRetCol column. If pRetCol is not specified, then it returns the entire array value. If it is not found, then **null** is returned.

Object findStringI(pKeyValue,pKeyCol=0,pRetCol=-1);

This method finds the string entry specified by pKeyValue (case insensitive) in column pKeyCol. If pKeyValue is found, then it returns the value in the pRetCol column. If pRetCol is not specified, then it returns the entire array value. If it is not found, then **null** is returned.

Example

The following creates a look up table. This is an abbreviated list of Ms Windows IO errors. When Windows returns an error code for some file function, you can use a table like this to lookup the corresponding text message. vErrorCodeList is an Array and each member of the Array is also an Array (with 2 members each).

```
     var vErrorCodeList=[
      [0,'Success'],
      [1,"Operation not permitted"],
      [2,"File or folder does not exist"],
      [3,"No such process"],
      [4,"Interrupted function"],
      [5,"I/O error"],
      [6,"No such device or address"],
      [7,"Arg List Too Long"],
      [8,"Exec Format Error"]
   ];

   var vErrorCodeLookup=utl.ELookup.make(vErrorCodeList); // Create the lookup table
...
...
...
       // This looks up the errorcode (pCode) in column 0 of the table
       // and returns the value in column 1.
   var vMsg=vErrorCodeLookup.find(pCode,0,1);

       // If you wish you could also do the opposite. If you have the error string,
       // you can lookup the error code as follows. You would have to switch the
       // column numbers so you search column 1 and return column 0.
   var vCode==vErrorCodeLookup.find('No such process',1,0);
```

Example

The following creates a look up table as a mini-database. This is list of characters from Lord of Rings, along with their character type, home location and the all important social security number. The vCharacterList is an Array of Arrays (each having 4 members).

```
     var vCharacterList=[
       //  SSN          Name             Type     Home
      ['123-45-9876','Frodo Baggins','Hobbit','Shire'],
      ['123-45-9812','Sam Gamgee','Hobbit','Shire'],
      ['876-55-1234','Aragorn','Man','Rivendell'],
      ['567-11-5643','Boromir','Man','Gondor'],
      ['000-00-0001','Gandalf','Wizard','????']
      ];
      var vCharacterLookup=utl.ELookup.make(vCharacterList); // Create the lookup table
...
...
...
      // This search looks up by Ssn and returns the corresponding character name.
      var vName=vCharacterLookup.find('567-11-5643',0,1);

...
...
      // This search looks up by name (case insensitive) and returns the entire entry.
      var vCharacterInfo=vCharacterLookup.findStringI('aragorn',1);
      if vCharacterInfo
         writeln('SSN='+vCharacterInfo[0]+'  Char Type='+vCharacterInfo[2]);
      end
```

class Methods

utl.ELookup.make[(pArray)];

This creates an instance of a ELookup object. The pArray parameter is an Array of Arrays, each of which should have the same number of members.

class EventObject

The EventObject class is the base class of all event objects. It is an abstract class, so it cannot be instantiated be itself. Other classes can be derived from this and they can be instantiated.

This class is an external class and has to be imported, as follows:


```
import('EventObject');
```

Instance Methods

`String` **`eventName`**[(String name)];

This method gets and optionally sets the event name for the event.

`Object` **`eventSource`**`[`(Object src)];

This method gets and optionally sets the event source object for the event. The event source is the object that generated (or is the subject of) the event.