Gentee overview

Introduction

The Gentee programming language can be classified as a procedure-oriented language with some features typical of object-oriented programming. It has no complicated constructions and is easy to use, but at the same time it is a powerful tool for solving all kinds of tasks. The syntax of the language is based on the syntax of the **C** programming language and it has a lot in common with other C-like languages **C++, Java, C#**. Gentee has the same numeric types **int, uint, byte, ubyte, long, double, float, ...** and can perform the same operations with them **+, ==, <, >, -, /, +=, ++, --, /=,...** as in other similar programming languages. When you write programs, you can use all basic constructions that you come across in other languages. For instance, such as **while, if, for, with, foreach, switch, include** .

The compilation unit in Gentee is a declaration command. Below you can see the sample declarations of global variables and macros.

```
global
{
   uint   i my = 0xFF
   str    name = "Alexey"
   arrstr colour = %{"red", "green", "blue" }
}

define
{
   PATH = $"c:\temp\docs"
   FLAG = 0x0001
}
```

While it is enough to specify the name and parameters (in case of a function) in order to access a variable or call a function, you have to add '**$**' to the left in order to substitute a macro. The values of macros are substituted during compilation.

```
i = my | $FLAG
```

Types

Besides basic numeric and built-in types **buf, str, collection**, it is possible to declare your own types using the **type** command.

```
type mytype_a
{
   uint id
   str  name
}
```

Variables of any type do not require additional initialization after their declaration, you can access them at once. Type fields are accessed with the help of the '**.**' operation. Types can be inherited (the same as in object-oriented languages) and the polymorphism of operations is provided for. If there is no method or function for some variable of a certain type, similar methods for its parent types will be searched for. It is possible to define and use operations used for numeric types (**=, +=, ==, !=**) and the **foreach** loop for any types.

```
type mytype_b<inherit = mytype_a> : double d

operator mytype_b =( mytype_b left, mytype_a right )
{
   left.id = right.id
   left.name = right.name
   return left
}
```

Functions

Gentee has three kinds of commands for determining the executable code: **func, method, operator**. The program is executed starting from the function that has the **main** attribute.

func - A regular function responsible for performing operations specified in it.

```
func hello< main >
{
   print("Hello, World!")
   getch()
}
```

method - A function linked to a certain type. Calling a method is similar to taking a field of a type and is performed with '**.**' with the name and parameters of the method following it.

```
method uint str.islastchar( uint ch )
{
   return this[ *this - 1 ] == ch
}

func myfunc
{
   str my = "String"
   print( my.islastchar( 'g' ))
}
```

operator - This command allows you to define assignment, comparison, arithmetical and other operators and use them later for any types.

```
operator str +=( str left, uint i )
{
   return left += str( i )
}

func myfunc : print( "Value = " += 100 )
```

Gentee is a strongly-typed language. It imposes certain limitations on programming, but it considerably reduces the possibility of mistakes on the other hand. Several functions and methods with the same names can exist, but they must have at least one different parameter or a different number of parameters.

Strings

Gentee has wide capabilities regarding working with strings. Strings are defined with the help of double quotation marks and have the control character '****'. If a string begins with '**$**', it will not take the control character into account. Besides inserting special characters, the control character allows you to insert data from files, calculate and insert expressions inside a string and also insert macros.

```
print( "Name = \( name += " gentee" ) Path = \$PATH\n")
```

It is often necessary to output some large amounts of text and part of this text is to be generated dynamically. It is convenient to use **text functions** in this case. They can output data to the string you specified while calling them or to the console.

```
text mytext( uint x )
Some text
x = \( x )
x * x = \( x*x )
\{ uint i
   fornum i, 5
   {
      @"x * \( i ) = \( x * i )\l"
   }
}
Some text
\!
```

Importing functions and using Gentee in other applications

From the very beginning Gentee has been developed in such a way that it would be possible to import functions from DLL (or similar modules in other operating systems) on the one hand and that it would be possible to use the Gentee compiler from programs written in other programming languages on the other hand.

If you need to import functions from a DLL, just specify the name of the DLL file and declare the imported functions.

```
import "kernel32.dll" {
   uint CloseHandle( uint )
        ExitProcess( uint )
   uint GetModuleFileNameA( uint, uint, uint ) ->
GetModuleFileName
}
```

If you want to compile files in the Gentee language and execute them from your application, just take the file gentee.dll and call the necessary interface functions. You can use the module gentee.dll free of charge, but you must comply with the license agreement.

Conclusion

Here are a few words about how the compiler works. The source code of the compiler in the **C** programming language is publicly available since Gentee is an open source project. The compilation rate is very high. As a result of compiling a program, you get a byte code that can be saved to a file or executed at once. It is possible to run the saved byte code without the second compilation or use it as a library module in other programs. Note that there is a set of ready libraries available and it is being constantly updated, which helps to create programs of any complexity. Besides, it is possible to create executable (exe) files.

We have described only the main things typical of the Gentee programming language. You can always find additional information on this site and discuss any questions with the developers and other users of Gentee.

How Gentee was created

Alexey Krivonogov

The idea of creating my own programming language occurred to me at the end of the 1990's. I was working on installation software at the time, and I realized that I needed a simple scripting language that would make programming easier and more comfortable. I started experimenting by creating simple languages, and by 2002 I felt that my work had yielded some real results. My brother joined me in this work, and we created the test version of the language soon after. It wasn't really the prototype of Gentee, but it gave us an idea of what our future language ought to be like, and in the process we gained invaluable experience.

In 2003 I stopped working on other projects and seriously got down to developing Gentee. Neither my brother nor I could devote all our time to Gentee, so development took more than a year. Our most difficult task was deciding on the syntax and features of the language. We developed Gentee as a procedure-based programming language. We refused to use objects and classes in their usual sense (although it should be mentioned that the language has both type inheritance and polymorphism now). We based the language on C-like syntax because this has stood the test of time, and has achieved an iconic status. We wanted to make a compact and fast compiler. And we wanted to make it possible to use Gentee from other applications via a small DLL file, so we were careful not to overload the language.

We can't say that everything went smoothly. Some problems took us several days to solve. Some of our solutions didn't work, and we had to do some things all over again. We even had to disallow some other features. And so Gentee became the language we wanted it to be - a personal and subjective language for sure!

The first public version of the compiler was published on the Internet on **November 1, 2004**. This date can be considered the birthday of the language. After that we regularly released versions with new features - we even released a version for Linux. In June 2006 we made Gentee an **open source project**. Although the compiler had been free from the very beginning, we decided not to publish the source code we had at that time, but instead to rewrite everything from scratch. The language had already become stable by then, but we wanted to complete some things and rewrite some others. It took more than a year for us to rewrite the compiler because of interruptions by other jobs, but we were determined and enthusiastic in our commitment to Gentee. You could say that Gentee was re-born in August / September, 2007.

Now that the source code of the compiler and libraries is open, we look forward with excitement to seeing how Gentee develops and improves through the efforts and expertise of its users.

Language Syntax Reference

You have opened the manual on the syntax and semantics of the **Gentee** programming language. All syntactic language constructions are described here. Programming opportunities offered by the language are also described here.

This manual is not a textbook on programming. It contains the description of the official version of the language from the developers of the Gentee compiler.

Gentee is a procedural, high-level language. Its syntax has much in common w ith the syntax of C/C++. (This should help users master many of Gentee's features quickly.) Like the **Java** or **C#** languages, a source program is compiled into object code, w hich is then executed by a virtual machine.

Table of contents

- Gentee Language in BNF

Identifiers

Identifiers are names that are used to refer to variables, types, functions, methods, etc. Identifiers can consist of alphanumeric characters and the underscore character. A name may begin only with a letter or the underscore character. It is permissible to use letters of the English alphabet as well as characters whose code is between 0x80 and 0xFF, but we recommend that you use only letters of the English alphabet to ensure that variable names are displayed correctly on other computers. The length of a name is limited to 255 characters. Sample valid names: **_my12, temp, MainFunction**. Names are case-sensitive: **MyFunc** and **myfunc** are two different names.

The language has some reserved names that cannot be used as identifiers. These are called keywords, and they are used to define constructions or objects in the language. The **keywords** are listed below:

as, break, case, continue, default, define, do, elif, else, extern, for, foreach, fornum, func, global, goto, if, ifdef, import, include, label, method, of, operator, return, sizeof, subfunc, switch, text, this, type, while, with, inherit.

Numbers

The Gentee language has several numeric types. There are several w ays to specify natural numbers or integers.

Decimal form

The most w idely used form.

Example: **65, -45367, 0**

Hexadecimal form

Numbers must begin w ith **0X** or **0x**. Characters from A to F can be used in upper or low er case.

Example: **0xÂA23, 0x1d2f, 0XFFFFFF**

Binary form

Numbers in binary form must begin w ith **0b** or **0B** and consist only of 0 or 1.

Example: **0b11001, 0B1010110110, 0b10101011000011**

Character code

You can specify a specific character instead of the number corresponding to it, by enclosing the character in single quotation marks.

Example: **'A', '(', 'k', '2', '='**

Gentee also has types called **long** and **ulong**. Each type occupies 8 bytes. To define such numbers, add **L** or **l** at the end.

Example: **23l, 0xfaafd45fff67fffL, -24363627252652L**

Real numbers

There are tw o types of real numbers: **double** and **float**. A number w ith a decimal point or w ith a mantissa is of the double type. To define a number of the float type, you should add **F** or **f** at the end. To specify a number of the double type w ithout a decimal point and a mantissa, you should add **D** or **d** at the end.

Examples of double numbers: **123.122, -123.2å-2, 789D**

Examples of float numbers: **12.75f, 0.55F, -78F**

Strings

Strings are define w ith a pair of double quotation marks in the language. If tw o stings come in a row , they w ill be combined into one string. By default, constant strings cannot be specified in the Unicode encoding. Gentee has Unicode strings (**ustr**) and you can use the **UTF-8** encoding in constant strings for later conversion into Unicode. Simple string variables are defined w ith the **str** type specified.

```
"It is a simple string
consisting of two lines."
```

There is a special character '\' that allow s you to perform various operations or substitutions. You can see the list of commands w ith the special character below .

\\ The command character output.

```
"c:\\temp\\readme.txt"
```

\\" A single quotation mark.

```
"This is \"Super Team\"!"
```

\n Line feed, code 0x0A.
\r Carriage return, code 0x0D.
\t Horizontal tabulation, code 0x09.
\l End-Of-Line - combination \r\n. It might be useful for output to a text file.
\0XX Combination of command character and zero follow ed by a number of character in hexadecimal notation makes any character w ith code from 0 to 255 put in a string.
\#Remove the preceding carriage returns or spaces and tab characters. Only either carriage returns or spaces and tab characters are removed depending on w hat precedes the special character. **\ 0xd 0xa**If a string just ends w ith the special character, the carriage return characters that follow w ill be deleted. It is convenient to split a too long string this w ay.

```
"Line 1\r\nLine 2\l Line \033 \
Line 3 too"
```

**\*...*\\** Comments. You can insert any comments into the string.
**\$macro$** In-line insertion of preprocessor macro. The last dollar sign '$' is optional if this sign is follow ed neither by letter nor by digit.

```
"Name: \$NAME Company: \$COMPANY \*Users name and company*\"'
```

\(expression) Outputting a result of the expression. In parentheses there might be an expression of any type, w here string conversion occurs.
\< filename > Content of the specified file is inserted. File name in the angle brackets must be specified as a macro string, i.e. ignoring the command character.

```
"5 + 10 = \( 5 + 10 ) Variable = \( var )\l \<c:\temp\my.txt>"
```

\[idname]If you have a long string and w ant to disable the special character in some part of it, specify any combination of any characters in square brackets. You do not even have to specify any additional character. To enable the special character later, just specify the same combination in square brackets.

```
"\[] \k\l\m [] \$NAME$ \[.S] \o\p\r [.S] \$COMPANY"
```

Note, there is also a macro string. Like the string, a macro string is enclosed in quotation marks; moreover, they are preceded by a dollar sign '**$**'. Unlike the string, the macro string does not use a command character, but it replaces macros w hich appear in a string. This type of a string is very appropriate for pointing file paths.

```
define {
 mypath = $"c:\myfolder\subfolder"
 myname = "application"
 myext = "exe"
}
...
s = $"$mypath\$myname$123.$myext"
s1 = "\$mypath\\\$myname$123.\$myext"
// s = s1 = c:\myfolder\subfolder\application123.exe
```

Binary data

Binary data is defined w ith a pair of single quotation marks. Numbers in the decimal and hexadecimal form and strings can be elements of binary data. Numbers can be separated by spaces, commas, carriage returns and semicolons. The **buf** type corresponds to binary data.

Combinations w ith the special '\' character are used to specify various elements.

***...*\\** Comments. You can insert any comments into the binary data.

**\$macro$** Macro value is inserted into the binary data. If the last sign '$' is follow ed by neither a digit nor a letter, it is considered to be optional.

\(expression) A result expression is inserted. An expression of any type enclosed in parentheses is to be converted into the binary data.

\< filename > Contents of the required file is inserted. A file name enclosed w ithin angle brackets must be w ritten as a macro string, i.e. the command character is ignored.

\"ñòðîêà" The macro string is inserted into the binary data. It is important to note that the Null character is not appended to the end of a string. The Null character is appended if the string is enclosed in parentheses \("string").

\h Insertion mode of numbers in hexadecimal notation. The numbers 2, 4, 8, are follow ed then, w hich indicate the total number size in bytes. If the number size is not specified, numbers are considered to be bytes. Keep in mind that hex digits are read in byte-read mode by default.

\i The read mode of decimal numbers. Numbers can be represented in floating-point notation in this mode. The size number of 2, 4 or 8 can also be indicated after i.

```
'5 \(50 + 45) afdcCCAB FF \* comments *\
 \h 567, 12 ; \"string" 45 \i4 255 3 +356 -1 45.56'
'0 FF fe fd ab cd 1a 2b 3c 4d 5e 6f \<c:\temp\my.exe>'
```

Macros

Macros are constants that are substituted during compilation. Macros can be used as identifiers, numbers, strings, binary data and collections. To substitute a macro for its value, you should specify the name of the macro betw een the '$' characters. If a macro is follow ed by a character that cannot be used in a name, you can leave out the '$' character at the end. Macros are not variables and you cannot assign any values to them. Macros are defined w ith the **define** command. You can also use macros for conventional compilation in the **ifdef** statement.

```
define {
   a = "str"
   b = 10
}
...
print( "\$a$ing \( $b + 20 )" )
```

There are the follow ing predefined macros. You cannot redefine them.

Predefined macros

$_FILE	The full name of the current source file.
$_LINE	The current line of the source file.
$_DATE	The current date in the format DDMMYYYY.
$_TIME	The current time in the format HHMMSS.
$_WINDOWS	Equals 1 in Window s.
$_LINUX	Equals 1 in Linux.

Related links

- The define command
- The ifdef command
- Macro expressions

Collections

Collections make it possible to store data of different types together. Besides, they can be used to initiate arrays and any other structures. Also, collections can be used to pass an undefined number of parameters of different types to functions and methods. The **collection** type corresponds to collections. Collections are defined w ith braces **%{ ... }**. You can specify different types of data or other collections separating them by commas insides braces. Global variables can be initialized by collections w hich contain only constants.

```
global
{
   arrstr  months = %{"January", "February", "March", "April", "May",
   "June", "July", "August", "September", "October", "November", "December" }
}
```

In order to initialize a structure w ith the help of collection, the appropriate assignment operator is required to be defined.

```
type test
{
   uint  num
   str    string
}

operator test =( test left, collection right )
{
   if right.gettype( 0 ) != uint : return left
   left.num = right.val( 0 )
   if right.gettype( 1 ) != str : return left
   left.string = right.val( 1 )->str
   return left
}
```

After that, a value is assigned to the fields, as follow s:

```
test  myt
myt = %{ 10, "test string" }
```

Using the collection argument in the function, you are able to pass a variable number of arguments of different types.

```
func outvals( collection cl )
{
   uint i
   fornum i,*cl
   {
      print("\(i) = \(cl[ i ])\n")
   }
}
```

The function call has the form.

```
outvals( %{ 10, 20, 30, 40 })
```

The program structure. Preprocessor

A program in the Gentee language can consist of one or several files. The main element of the program is a command. The command starts on a new line, most commands contain blocks enclosed in curly braces { }. It is possible to divide all commands into four groups by their purpose.

Preprocessor commands

The preprocessor is responsible for the substitution of macro values, the replacement of service characters and conditional compilation. The preprocessor performs its operations right during the compilation of the current fragment of the source code.

The define command	The definition of macros.
The ifdef command	Conditional compilation.

Executable code commands

These commands contain statements and are responsible for the executable part of the program.

The extern command	Predefined functions, methods and operations.
func	Function.
Method declaration: method	Method declaration.
Redefining operator operations	Operator redefinition.
Properties	Function-property.
text	Text function declaration.

Definitions of types and global variables

The type command	Declaring type.
The global command	Global variable declaration.

Other commands

The include command	Including Gentee files.
The import command	Importing functions from DLL.
The public and private commands	Name protection commands.

This is an example of the simplest program.

```
/* Example */

define
{
   NAME = "John"
}

func main<main>
{
   print("Hello, \$NAME!")
   getch()
}
```

Comment. Character substitution

When running, the compiler deletes all comments, replaces macros w ith their values and replaces the formatting characters.

/*...*/ Comments can appear anyw here. A comment begins w ith a forw ard slash/asterisk combination **/*** and is terminated by end comment delimiter ***/**.
// Single-line comments. These comments are terminated by the End-of-Line characters.

```
/*
 This is a comment.
*/
a = 4 + 5  // This is a comment too.
```

; The new line character is the separating character betw een expressions and statements. A semicolon is replaced w ith a new line character. You can use this character if you w ant to put several statements on one line.
: A colon is replaced w ith an opening curly brace and a closing curly brace is added at the end of the current line.

```
// These examples are equal
if a == 10 : a = b + c; c = d + e

if a == 10
{
   a = b + c
   c = d + e
}
```

The define command

The **define** command is used to specify macros. You can assign a constant to a macro, w hich applies to the follow ing types of constants: a number, a string, a binary data or an identifier name, furthermore, you can assign a macro to the value of another macro. Later on, the name of the macro should be specified as **$macroname** or **$macroname$** for it to be replaced w ith its value. It is possible to redefine a macro in other **define**. Macros are defined enclosed w ithin curly braces, and each line contains only one macro definition. The macro definition consists of a name follow ed by the equal sign = and the appropriate constant or an expression. We recommend that you use only uppercase letters in the names of macros.

```
define
{
   A = 0xFFFF; B = 3.0
   NAME = "First and Last Name:"
   ID = idname
   BB = $B
}
```

Attributes

You can specify the **export** and **namedef** attributes for **define**. Use the **export** attribute if you distribute the module as byte code (a **.ge** file) and w ant to make it possible to use these macros in other programs. If **define** has the **namedef** attribute, all its macros can be used w ithout specifying the '**$**' character.

```
define <export namedef>
{
   FALSE = 0
   TRUE = 1
}

func uint my( uint param )
{
   if param >20 : return FALSE
   if param <10 : return $FALSE  // $FALSE == FALSE
   return TRUE
}
```

Specifying name for define

You can specify a name for **define**. In this case, it is possible to access macros both directly and specifying the define name. It is made in order to avoid conflicts betw een macros from various modules. In this case, access to a macro looks like this: **$definename.macroname**.

```
// file1.g
define myflag< export >
{
   FLAG1 = 0xFFF0
   FLAG2 = 0xFFF1
}
// file2.g
define flags
{
   FLAG1 = 0x0001
   FLAG2 = 0x0002
}

func uint my( uint param )
{
   if param & $myflag.FLAG1
   { ... }
   if param & $flags.FLAG1
   { ... }
}
```

Enumeration

Gentee has no separate command for defining enumerations. You can use the define command for that. If a macro has no value assigned to it, its value becomes one time greater than the value of the previous macro. If there is no previous macro or it is not an integer, the value of the current macro is set to 0. Macros can be separated by spaces in case of enumeration.

```
define
```

```
{
   VAL0 VAL1 VAL2 // VAL2 = 2

   ID1 = 100
   ID2 ID3 ID4
   ID5  // ID5 = 104
}
```

Expressions

Not only numbers, but also expression results can be assigned to macros. Either constants or other macros can be operands in expressions. You can take a look at the full list of possible operations on the **Macro expressions** page.

```
define
{
   VAL0 = 10 + 245
   VAL1 = $VAL0 + ( 12 - 233 )
   VAL2 = $VAL1 & 0xFFFF
   SUMMARY = $VAL0 | $VAL1 | $VAL2
}
```

Related links

- The ifdef command
- Macros
- Macro expressions

The ifdef command

The **ifdef** command of conditional compilation allow s you to include and exclude some parts of the program for compilation depending on some conditions. A conditional expression must follow the **ifdef** keyw ord and the part of the program that should be compiled if the condition is met (not equal to 0) should come after it in curly braces. You can use an expression consisting of macros and constants as a condition (a number, a string, a binary data). You can take a look at all possible operation for expressions on the **Macro expressions** page.

In the example below the **myfunc** function w ill be compiled if the macro **$MODE** is a number not equal to 0 and not an empty string.

```
ifdef $MODE
{
   func myfunc( uint param)
   { ... }
}
```

You can use **ifdef** not only on the top embedment level, but also inside any other command and even inside expressions. Besides, it is possible to embed **ifdef** commands inside each other.

```
func myfunc( uint param )
{
   uint i = param
   ifdef $ABC == 3 || $NAME == "Private"
   {
      i *= 2 + ifdef !$MODE { 100 } else {200}
   }
   ...
}
```

elif and else

If the condition is false and another part of the program should be compiled, the **else** command is used. If there are more than tw o variants of compilation, you can use the **elif** command w ith an additional condition. You can have several **elif** commands in a row and the **else** command at the end.

```
define
{
   ifdef $MODE == 5
   {
       NAME = "Public"
       MODE= 10
   }
   elif  $MODE == 4
   {
       NAME = "Debug"
   }
   elif  $MODE > 5 : NAME = "Private"
   else : NAME = "Unknown"
}
```

Related links

- The define command
- Macros
- Macro expressions

Macro expressions

When you define macros with the help of **The define command** and in **The ifdef command**, you can use simple expressions with constants and macros. Operands must be of the same type except for logical operations **&&** and **||**. It is possible to use parentheses to specify the order of calculating the expression.

Operation	Type of operands	Type of result
Arithmetic operators		
+	int uint long ulong float double	int uint long ulong float double
-	int uint long ulong float double	int uint long ulong float double
*	int uint long ulong float double	int uint long ulong float double
/	int uint long ulong float double	int uint long ulong float double
Bit operators		
&	int uint long ulong	int uint long ulong
|	int uint long ulong	int uint long ulong
^	int uint long ulong	int uint long ulong
Logical operators		
&&	int uint long ulong float double str(1 if the length >0) buf(1 if the length >0)	int uint
||	int uint long ulong float double str(1 if the length >0) buf(1 if the length >0)	int uint
Comparison operators		
==	int uint long ulong float double str buf	int uint
!=	int uint long ulong float double str buf	int uint
>=	int uint long ulong float double	int uint
<=	int uint long ulong float double	int uint
>	int uint long ulong float double	int uint
<	int uint long ulong float double	int uint
Unary operators		
+	int uint long ulong float double	int uint long ulong float double
-	int uint long ulong float double	int long float double
~	int uint long ulong	int uint long ulong
!	int uint long ulong float double str(1 if the length >0) buf(1 if the length >0)	int uint

```
7 + $YEAR - 2000
2.3 * ( VAL1 - $VAL0 / 2.0 )
$VALFLAG | 0xff00
$MODE1 || ( $MODE2 == 3 && $COMPILE == "WINDOWS" )
$PROGNAME != "My Application" && $PROG != "Debug"
```

Related links

- The define command
- Macros
- The ifdef command

The include command

The **include** command is used to include additional files w ith source code in the Gentee language or w ith already compiled byte code. You can include ready-made libraries and use their functions after that or combine several modules into one project. If you specify a file w ith the **.ge** extension that contains compiled byte code, it is included w ithout additional compilation. If some file is included more than once, the compiler ignores the repeated inclusions of the file.

Included files are listed inside curly braces, either one file on a line or they must be separated by commas. You can specify both absolute and relative paths to files. The names of files are strings that is w hy it is necessary to either double the '\' character or put '**$**' before the braces.

```
include
{
   "myfile1.g"
   $"c:\path\myfile2.g"
   "c:\\mylib\\mylib.g"
   $"$MYLIB\library.g"
   $"..\src\library.g"
}
```

The **include** command can be used in any place of the program and in any Gentee files. You can specify **include** inside the **ifdef** command.

```
ifdef $MYPROG
{
   include : "myfile1.g"
}
// OR
include
{
   ifdef $MYPROG : "myfile1.g"
}
```

You can configure the compiler profiles in such a w ay that you alw ays include certain files and then you do not have to define them w ith **include**. You can also list directories to search for files in a profile. In this case, it w ill be enough for you to specify only the file names in the **include** command and the compiler w ill automatically find them in these directories.

The import command

The **import** command allow s you to export functions from DLL. The keyw ord import is follow ed by DLL filename, w hich contains imported functions, and afterw ards w e open the description block. Each line of the block contains a description of the imported function, i.e. a type of the return value, if any, and a function name are aligned w ith parameters separated by commas and enclosed in parentheses. You can substitute a new function name for the name of the imported file. To rename the function, you need to use -> after the description and a new name. When function is imported, calling DLL function is made in the same w ay as calling function w ritten in Gentee.

```
import "kernel32.dll"
{
   uint CloseHandle( uint )
   uint CopyFileA( uint, uint, uint ) -> CopyFile
   uint CreateFileA( uint, uint, uint, uint, uint, uint, uint ) -> CreateFile
   uint CreateProcessA( uint, uint, uint, uint, uint, uint, uint, uint,
                        STARTUPINFO, PROCESS_INFORMATION ) -> CreateProcess
}
```

If you are going to run the Gentee program from your ow n EXE file, you can use functions from the EXE module. To do it, specify the name of the DLL file as an empty string and read about passing the addresses of the functions to be imported in the **Configuring and running Gentee** section.

Attributes

cdeclare

Means that the **__cdecl** functions are imported. By default, the imported functions are considered to be the **__stdcall** functions.

```
import "myfile.dll" <cdecl>
{
   ...
}
```

link

In this case, a required .dll file w ill be included in a .ge file; w hile launching a program the .dll file is w ritten to the temporal directory w here the program load it. The .dll file w ill be deleted after the program has ended. In other w ords, if you don't w ant some extra .dll files to be distributed, but you doubt if the files have been stored before in a user's computer, this attribute w ill be helpful for you. It is desirable that the complete path to the .dll file should be specified.

```
import $"c:\mypath\myfile.dll" <link>
{
   ...
}
```

exe

This attribute should be used if you get to know the relative path from your program to the .dll file. This is an example illustrated my.dll loading from the subdirectory *Plugins*.

```
import $"plugins\my.dll" <exe>
{
   ...
}
```

The public and private commands

All functions, metods, types or other elements of the Gentee language become publicly available by default after they have been defined. Take advantage of the **private** command in order to make elements be accessible only w ithin the file, w here they have been defined. All language elements that follow this command, w ill be accessible before the current .g file has compiled. After that, names of these elements w ill be deleted, you w ill be unable to find the elements by specifying their names. The **public** command makes the next elements be publicly available. You can use either **public** or **private** in the source as necessary. These commands are likely to be used for functions, methods, operators, types and global variables.

```
private
func str mylocal
{
   ...
}

public
func str myfunc
{
   ...
   mylocal()

}
```

Types and variables

Gentee is a strongly-typed language that is w hy types occupy a very important place in programming in Gentee. All types can be divided into three groups: **numeric** types, **structural** types and the **reserved** type.

Numeric types

All numeric types are built into the language. **uint** is the most w idespread numeric type. The Gentee language has neither pointers nor logic type, the **uint** performs their functions. The **byte, ubyte, short, ushort** types are considered as int or uint types (depending on the sign) w hen arithmetic operations are performed. If you specify them as fields in structural types, they w ill occupy the corresponding number of bytes.

Type name	Size of type	Minimum	Maximum	Comments
Integer types				
byte	1(4)	-128	+127	signed
ubyte	1(4)	0	+255	unsigned
short	2(4)	-32768	+32767	signed
ushort	2(4)	0	+65535	unsigned
int	4	-2147483648	+2147483647	signed
uint	4	0	+4294967295	unsigned
long	8	-2^63	+2^63 - 1	signed
ulong	8	0	+2^64 - 1	unsigned
Floating types				
float	4	(+ or -)10E-37	(+ or -)10E38	;
double	8	(+ or -)10E-307	(+ or -)10E308	

Structure types

Structure types are defined by the **type** command. Types string (**str**), binary data (**buf**), collection (**collection**) are embedded into the language. A lot of types are defined in the standard and other libraries (arrays, hashes etc).

Type reserved

The **reserved** type is of special significance, w hich belongs neither to the fundamental types nor to the structure ones. This type is denoted by the array of bytes, w hich is defined and used as the array. The distinctive feature of the reserved type is that, the memory space is reserved w here it has been defined. For example, you can specify a field in a structure *reserved field[50]*. This means that a memory space of 50 bytes w ill be reserved in the structure. If you specify the same code inside a function then you reserve 50 bytes in the stack for this local variable. The size of memory reservation allow s up to 65 535 bytes. Bear in mind that you should not use an expression in order to specify the required size. It is a constant number that must be enclosed in square brackets.

The type command

Structure types are defined by using the **type** command. This command is follow ed by the specified type name and fields description in braces.There can be one or more fields of the same type defined in each string of the block. First, a type name is specified, w hich is follow ed by field names separated by commas or spaces. The field can have a numeric type as w ell as the previously defined structure type. Fields of the structure type are organized in memory as they have been described in the source code; if the field has a structure type, the structure of this type is completely embedded in the final structure. When fields are defined, dimensions separated by commas and enclosed in square brackets and the item type follow ed the keyw ord **of** can be determined. To get or assign a field value for a variable, its name should be specified after a full stop.

```
type customer
{
   str     name, last_name
   uint    age
   arrstr phones[ 5 ]
}
...
customer cust1 //
cust1.name = "Tom"
cust1.age = 30
cust1.phones[ 0 ] = "3332244"
```

Attributes

index

Types can contain other elements, like a string array. You can specify w hat type of elements can be included in the object of this type by default. To do this, assign a corresponding type to this attribute. If elements have the same type by default (for example, tree), w rite **index = this**.

```
type arrstr <index=str  inherit = arr>
{
   ...
}
```

inherit

You can inherit types. You have to use the attribute **inherit = èìÿòèïà**. See more details in **Type inheritance**.

protected

Gentee makes it possible to restrict access to fields of the type from other modules. The specified **protected** attribute is used for this purpose. In this case, all fields of the type w ill be accessible before the current file has compiled. Otherw ise, fields of this type w ill be unaccessible.

```
type mytype <protected>
{
   ...
}
```

Additional features

For any structure type you can define methods that w ill allow you to

- Perform additional actions during initialization and deletion of a variable
- Specify **of** w hen describing variables of this type
- Use square brackets w hen addressing individual elements
- Use **foreach** to scan elements of this type.

These methods are described in **System type methods**.

Related links

- Type inheritance
- System type methods

Type inheritance

Gentee allow s you to inherit structure types. For this purpose you have to specify an attribute **inherit** w ith the name of the parent type.

```
type mytype <inherit = str>
{
   uint i
   uint k
}
```

Specify an empty curly brackets or a collon if a new type does not have additional fields.

```
type mynewtype <inherit = mytype> :
```

You cannot inherit base numeric types and the type *reserved*. The type inheritance allow s you to get fields of any parent type.

```
type my <inherit = mytype>
{
   str name
}
...
my m
m.i++
```

Also, you can call methods or functions of all parent types. The compiler finds a suitable method or function w hen you call some function or method. For example, there are the follow ing functions

```
func print( mytype mt, uint i )
{
   print("MYTYPE PARAMETER = \( mt.i + i )\n")
}

func print( mytype mt )
{
   print("MYTYPE = \( mt.i )\n")
}

func print( my m )
{
   print("MY = \( m.i )\n")
}
```

You have

```
my mm

print( mm, 20 )
print( mm )
```

The first *print* outputs **MYTYPE PARAMETER = 20** and the second *print* outputs **MY = 0**, but nor **MYTYPE = 0**. The situation w ith methods or operators is like. If you need to call just a parent method or a function then use the typecasting operator '**->**' w ith the parent typename. **print(mm->mytype)** displays **MYTYPE = 0**.

So, **Gentee** gives you the such main object-oriented programming features as the **inheritance** and the **polymorphism**.

Related links

- The type command

System type methods

For each type you can define methods that w ill simplify the w ork w ith variables of this type and increase its possibilities. Lets take some abstract type.

```
type test<index = uint >
{
   uint mem
   str  name
   uint itype
   ubyte dim0
   ubyte dim1
   uint  count
}
```

Initialization

In Gentee the initialization of variables and fields of any type is automatic. If you w ant to perform additional actions during initialization of a type variable, define the method **init**. We should note that all number fields are initialized as zeroes, and fields of other types are also initialized according to descriptions of those types. For example, if the field has a **str** type, it w ill be initialized w ith an empty string at once.

```
method test test.init
{
   this.mem = malloc( 4096 )
   this.name = "TEST"
   itype = uint
   return this
}
```

Deletion

If before deleting a variable of this type you w ant to perform additional actions, specify them in the method **delete**.

```
method test.delete : mfree( this.mem )
```

Using the of operator

Lets assume that a variable of this type can contain variables of another type. In this case you should have an opportunity to indicate it w hen you describe the variable. For example, **test mytest of double**. You should define the **oftype** method for the compiler to understand the **of** operator. It should have a parameter giving the element type.

```
method test.oftype( uint itype )
{
   this.itype = itype
}
```

Specifying size and dimension

Lets assume that w hen you describe a variable you w ant to create several elements at the same time and also specify the dimension of this variable. For example, **test mytest[10,20] of double**. To do this, you should describe one **array** method for each possible dimension.

```
method test.array( uint first )
{
   this.count = first
   this.dim0 = first
}

method test.array( uint first second )
{
   this.array( first * count )
   this.dim0 = first
   this.dim1 = second
}
```

Addressing by an index

If you w ant to get the i-th element of the variable of this type using brackets, you should describe one **index** method for each dimension. You can specify not only numbers, but any other types as indexes. To do this, you only need to define a **index** method w ith a parameter of a corresponding type. Note that the **index** method must **return the pointer** to the element it finds!

```
method uint test.index( uint first )
{
   return this.mem + first * sizeof( this.itype )
}

method uint test.index( uint first second )
{
   return this.index( this.dim0 * first + second )
}

method uint test.index( str num )
{
   return this.index( uint( num ))
}

...

test mytest[10]

mytest["0"] = 10
mytest[1] = 20
print("0 = \( mytest[0] ) 1 = \(mytest[ "1" ])")
```

Using the foreach operator

The Gentee language has a **foreach** operator that scans all elements of a variable of specified type. If you w ant to use this operator for your type, you should define the **eof, first, next** methods w ith a **fordata** parameter. The **icur** field of **fordata** stores the index of the current element during scanning. You should zero it in the **first** method and increase in the **next** method.

```
method uint test.eof( fordata fd )
{
   return ?( fd.icur < this.count, 0,  1 )
}

method uint test.first( fordata fd )
{
   return this.index( fd.icur = 0 )
}

method uint test.next( fordata fd )
{
   return this.index( ++fd.icur )
}
...
test mytest[10]
uint sum

foreach curtest, mytest
{
   sum += curtest
}
```

Redefining operators

You can use all kinds of operations like **=, +, *, ==, !=, * etc.** for variables of any type. To do this, you need to describe corresponding commands of **operator**. You can find more details at the Redefining operator operations page.

```
operator test =( test left, collection right )
{
   uint i
   fornum i=0, *right
   {
      if right.gettype(i) == uint
      {
         left[i] = right[i]->uint
      }
   }
   return left
}

...

test mytest[10] = %{ 0, 1, 2, 3, 4, 5, 99, 8 }
```

Related links

- The type command
- Redefining operator operations
- foreach statement
- Method declaration: method

The global command

Global variables are declared by using the **global** command. All necessary variables defined w ithin the curly brackets follow the global command. You can put variables of the same type together in a single line; first, you specify a type name, w hich is follow ed by variable names separated by either a comma or a space. For example,

```
global
{
   uint g_cur summary mode
   str  name = "John", g_result, company
}
```

If the variable type supports the use of **of** and brackets, you can specify those additional parameters w hen you describe a global variable. Besides, number variables, along w ith strings **str** and binary data **buf** can be initialized at the moment w hen they are described w ith the help of the assignment operation '='. When you initialize variables, you can use macroexpressions. By default, the variable is initialized w ith zeroes or by calling the corresponding initialization function.

You can address any global variable from the moment its declared in further functions and methods.

```
global
{
   str a b = "My string", c
   uint num = 25 * $DIF, num2
   double  dx = $DX + 0.1
   arr x[ 10 ] of int
   arrstr months = %{"January", "February", "March", "April", "May",
                     "June", "July", "August", "September", "October",
                     "November", "December" }
}
```

Related links

- Macro expressions
- System type methods

Local variables

Local variables serve for temporary storage of intermediate results w hen a function or a method is executed. A local variable can be declared in any part of the function body including nested blocks taken in braces. Each variable must be given its ow n type declaration in a new line, that contains a specified type name and variable names separated by commas.

If the variable type supports the use of **of** and brackets, you can specify those additional parameters w hen you describe a local variable. Besides, number variables, along w ith strings **str** and binary data **buf** can be initialized at the moment w hen they are described w ith the help of the assignment operation '**=**'. When you initialize variables, you can use macroexpressions. By default, the variable is initialized w ith zeroes or by calling the corresponding initialization function.

```
func myfunc( uint param, str name )
{
   str a b = "My string" + name, c
   uint i = 25 * param + 3
   uint k = 10, l = 2
   arr x[ k, l ] of uint
   arrstr months = %{"January", "February", "March", "April", "May",
                     "June", "July", "August", "September", "October",
                     "November", "December" }
   ...
}
```

Scope of local variables

The scope of a local variable extends from its declaration to the end of the block in w hich it w as declared, including nested blocks. Global and local variables are likely to be redefined; in other w ords, w ithin a block a new ly declared variable shares the same name as the variable previously declared. It is possible that the new variable may be of another type. The last-mentioned variable w ill be available till the end of the current block, and the previously declared variable becomes hidden. Once the block ends, the variable that w as subsequently hidden is again available. Actually, the objects declared as local ones are automatically created w hen the block begins execution, and destroyed w hen the block ends. You can create objects w ith the help of the new service function. In this case, a programmer should keep an eye on deleting objects, using the destroy function. As local variables are deleted w hen w e exit the function, you can only return numeric local variables.

```
func myfunc
{
   uint a = 10
   ... // a == 10
   {
      ... // a == 10
      uint a = 3
      ... // a == 3
      while ...
      {
         ... // a == 3
      }
      ... // a == 3
   }
   ... // a == 10
}
```

Related links

- Returning variables
- System type methods

Function declaration: func

A function consists of tw o parts: a declaration and a function body. When you declare a function, you specify the keyw ord **func**, the type of its return value, its name, its attributes enclosed in angle brackets and its parameters enclosed in parentheses. Only the function name is required. When you do not specify the type of its return value, the function does not return any values.

A body of a function or a method is everything included in braces that follow the function description. The function body can contain subfunctions, expressions, constructions and descriptions of local variables.

```
func  uint sum( uint left right )
{
    return left + right
}
```

Attributes

entry

This attribute is specified for functions that must be started automatically before the main function is called.

main

This attribute is specified for the main function w ith w hich the program is started. If there are several functions w ith this attribute, the last function w ith **main** attribute w ill be called. The main function is run after all **entry** functions are called. Functions that have **main** or **entry** attribute should not have parameters.

```
func  uint myprog<main>
{
   print("Hello, World!")
   getch()
}
```

result

Gentee does not allow returning a structural type from a function if it belongs to (is described inside) this function. This attribute makes it possible to evade this restriction. You can find more details about using it at the Returning variables page.

alias

If you need to get and transmit a function, method or operator identifier somew here, you can use this attribute. As functions and methods can have the same names, but different parameters, finding the necessary function can lead to some difficulties. You can assign an **alias** name to the attribute and use this name as a variable function identifier.

```
func uint myfunc_verylongname<alias = myfunc>( uint param )
{
   return param * 10
}

func str mystring<result>
{
   result = "Result string"
}

func main<main>
{
        print( "Val= \( myfunc->func( 10 ))")
        print( mystring() )
        getch()
}
```

Parameters

Each parameter declaration is a comma-delimited series of parameter names w ith the type identifiers specified after a type name, then follow ed by a comma or space and a new type name and parameters. If a function takes no parameters, omit the identifier list and the parentheses in its declaration. You can define functions w ith the same name but w ith different parameters. In this case, w hen you call a function the compiler looks for a function w ith the same name and parameters.

When you describe parameters, you can use brackets to specify the dimension and the **of** operator. When you describe such parameters, you do not have to specify a precise number of elements in brackets.

```
func uint myfunc( uint a b c, byte d, str st1 st2, arr marr[,] of uint )
{
   ...
}
```

Addressing the parameters is the same as addressing local variables. All **numeric types** are given to a function or a method **by value**. That means you can change the value of the parameter w ithout any consequences. All **structure types** are given **by reference**. In this case all the changes you have made w ill happen to the original variable that you passed as a parameter.

```
func str myadd( str left )
{
   left += " OK!"
   return left
}

func main<main>
{
   str val
   myadd( val = "Process" )
   print( val )
}
```

Related links

- Returning variables
- Local variables
- Subfunction declaration: subfunc

Method declaration: method

You can define different **methods** for any types. Any method is a function associated w ith an object of the appropriate type, that the method should operate on. A method is defined by specifying the keyw ord **method** follow ed by the name of the return type (if it is required), an object type and a method name follow ed by a separating period. Like declaring a function, you should specify method parameters follow ed by its body: **object.methodname(parameters)**.

The parameter **this** is created automatically w ithin the method; furthermore, this parameter contains the object to w hich the given method is called. The parameter **this** has the same type as the object does.

```
method uint str.islast( uint ch )
{
    return this[ *this - 1 ] == ch
}

func main<main>
{
   str mystr
   ...
   if mystr.islast('\')
   {
      ...
   }
}
```

You can specify **result** and **alias** attributes for **method** like for functions. Methods are responsible for object initialization and destruction, getting index and type conversion as w ell as for other operations. See more details on the System type methods page.

Type conversion

Type conversion is also declared w ith the help of the methods. A source type is specified as the object type of the method and a destination type of the object is specified as the method name. If the destination type is structured you must use **result** attribute.

```
// uint -> str
method str uint.str < result >
{
   result.out4( "%u", this )
}

// str -> uint
method uint str.uint
{
   uint end
   return strtoul( this.ptr(), &end, 0 )
}

func main<main>
{
   str mystr

   uint a = uint( "100" )
   mystr = str( a )
}
```

Related links

- Function declaration: func
- System type methods

Redefining operator operations

Gentee enables objects to do new operations using the existing operators (**=, ==, +=, +, *, <, == etc.**). Moreover, the statement priority keeps permanent. The operation processing is executed w ith the help of special function-operators w hich include the keyw ord operator. Then you should specify the result type, the operator represented by characters and one or tw o parameters w hich are subject to the operation (either unary or binary). The parameters type coincides w ith the operands type, thus the parameters w ill contain the operand values. If the operation is considered to be binary, the first parameter represents the left operand and the second parameter represents the right one. Operands can have different types. If the result of the operation is a new object (for example, adding) then you must use **result** attribute. Also, you can define **alias** attribute if you need that.

If you w ant to describe comparison operators for your type then you can take only **==, < and >** operators . Operators **!=, >=, <=** may not be described and are compiled to ==, < and > automatically.

```
operator str +<result>( str left right )
{
   ( result = left ) += right
}

operator str +=( str left, int val )
{
   return left.out4( "%i", val )
}

func main<main>
{
   str dest = "Zero", a="One", b="Two"

   print( ( dest = a + b )+= 323 )
}
```

Related links

- Function declaration: func

Declaring text function

The **text** command is used specifically to w ork w ith text data. It allow s you to generate text of any complexity and size.

When you declare a text-function, you specify the keyw ord **text**, the attributes enclosed in angle brackets and the parameters enclosed in parantheses. The attributes as w ell as the parameters are optional. A text-function does not return the value. The attributes as w ell as the parameters are declared in the same w ay as functions. The body of the text function (the output text) starts in a new line after declaring the text function and goes either to the end of the file or to the follow ing service characters: **\!**.

As for the simple function, strings enclosed in double quotes are inserted into the source code; as for the text function, on the contrary, the source code is inserted into the text.**A text function** outputs a text to a console or to a string. It is subject to the text function call.

Console output

Output to the console is carried out w ith the help of the **@** unary operation.
@nametextfunc(parameters)

String output

Output to a string is carried out w ith the help of the **@** binary operation w here the output string is specified on the left. The result of the text function w ill be added to the string.
stemp @ nametextfunc(parameters)

Additional features

The service character and the commands operated in a string are used in a text function. Furthermore, a text function uses the follow ing additional commands.

\! The end of a text function. By default, a text function goes to the end of the file.

\@name(...) Calling another text function. The output mode (to a console or to a string) is not changed.

\{...} Insertion of the code block. You can specify the source code enclosed in braces as in the function body. This block fits the block of the low est level of the function and you can declare subfunctions there. Use operation **@"string"** in the code block to output a string to the current output stream.

```
text hello( uint count )
Must be \(count) strings
\{
   uint i
   fornum i, count : @"\(i + 1) Hello, World!\n"
}Welcome to Gentee!\!

func b <main>
{
   @hello(3)   // Write to console
   @"Press any key...\n"
   getch()

   str out
   out @ hello( 5 )
   print( out )
   getch()
}
```

Current output

You can use the current output string by the using **this**. If **this** equals zero then the console is the current output of the text function.

Related links

- Function declaration: func
- Strings

Properties

Gentee provides you w ith the **property** in order to get or set values of the fields of the structured types. Using the properties you can hide a direct access to the fields and perform additional calculations in order to get a field value or set a field value w ith the help of the assignment operator. A property name must differ from a field name, because a direct access to a field has a higher priority; otherw ise, a field value w ill be got or set.

The **get** property, that returns a value, must contain no arguments.

```
type mytype
{
   str val
}

property str  mytype.value
{
   return this.val
}
```

The **set** property, that defines a value, must contain one argument. Also, the set property can return a value.

```
property str mytype.value( str newval )
{
   if *newval : this.val = newval
   else : this.val = "empty"
   return this.val
}
```

A property name is specified in the same w ay as a field in order to call a property. The *set* property is called if it is specified on the left side of the assignment operator; otherw ise, the *get* property is called.

```
func myfunc
{
   mytype myt

   myt.value = "New value"   // set
   print( myt.value )        // get
}
```

Related links

- Function declaration: func

The extern command

You cannot call any function before its definition. The **extern** command provides you w ith preliminary declaration of a function, a method, a property or an operator. The command allow s you to call a function before it has been defined. For example, a recursive function call from another function.

The keyw ord **extern** is follow ed by the block that contains function declaration. Each line of the block contains either function, method, operator or property declaration, excluding their bodies.

```
extern
{
   func uint b( uint i )
   func uint c( str in )
}

func uint a( uint i )
{
   return b( 2 * i ) + c( "OK OK" )
}

func uint b( uint i )
{
   return i + 20
}

func uint c( str in )
{
   uint ret i

   fornum i,*in
   {
      if in[i] == 'K' : ret++
   }
   return ret
}
```

Related links

- Function declaration: func

Subfunction declaration: subfunc

Subfunctions are defined in the body of the function w ith the help of the **subfunc** construction. A subfunction is defined in the low est level of the embedded block of the body. You can call a subfunction only from the function body as w ell as from other subfunctions of the given function. It is impossible to define another subfunction and to call itself recursively, because local variables are considered to be static. A subfunction is able to redefine other functions' names. A subfunction is actually called like a function; moreover, a subfunction as w ell as its parameters are declared in much the same w ay as the function, except for the lack of attributes. You can use local variables of the function w ithin the subfunction.

Subfunctions are very usefull w hen you need to execute the same code some times inside the function but you do not w ant to describe an independent function.

```
func uint myfunc( int par )
{
   int locvar
   subfunc int mysubfunc( int subpar )
   {
      return locvar + par + subpar
   }

   locvar = mysubfunc( 5 )
   par = mysubfunc( 10 ) + mysubfunc( 20 )
}
```

Related links

- Function declaration: func

Returning variables

Gentee prevents returning local variables from functions and methods if the variables are not of the **numeric** data type. All structural local variables are deleted as soon as the function has finished executing. For example, if the next function is called, the error occurs.

```
func str func1
{
   return "Result string"
}
```

In such cases, the **result** attribute can be used. The attribute enables you to return a result value from functions or methods. Furthermore, using the attribute avoids defining and sending unnecessary local variables. Take advantage of this attribute in order to use the *result* variable, that w ill be returned as soon as the function has finished executing. If a function has the **result** attribute, the **return** instruction is not required or it must contain no expression.

```
func str myfunc<result>
{
   result = "Result string"
}

func main<main>
{
   print( myfunc())
}
```

Note that a function or a method of this type is called after a temprorary variable has been actually created in the calling block. The variable is sent to the function w here it is used as the **result** variable.

Related links

- Function declaration: func

Statements

A function (method, operator, property) body contains some statements that are used to interrupt the sequential execution of a program. Some of them contain blocks w hich also have nested statements.

There are several types of statements, as follow s:

Conditional statements

if-elif-else	Conditional statement.
switch	Case statement.

Loop statements

while-do	Simple loop statement.
do-while	Simple loop statement, evaluated at the bottom.
for	Loop statement that provides initialization and increment clauses.
fornum	Loop statement that executes a finite number of iterations and has autoincrement.
foreach	Loop used for enumerating elements.

Instructions of unconditional transfer of control

return	Function termination.
break	Loop termination.
continue	Immediate transfer of control to the next loop iteration.
label	Label definition.
goto	Unconditional transfer of control to a label.

Other statements

with	Short fields management.

if-elif-else statement

The statement consists of the follow ing parts:

if

The **if** part contains the **if** keyw ord, a conditional expression and the block executed if condition is TRUE. If the condition is FALSE, control passes to the next part **elif**.

elif

The **elif** part contains the **elif** keyw ord, a conditional expression and the block executed if condition is TRUE. The statement is likely to contain some **elif** parts follow ed one after another.

else

The **else** part contains the **else** keyw ord and the block executed if the condition of the **if** part as w ell as the condition of all **elif** parts are FALSE.

The **elif** and the **else** operators are optional.

The value of a conditional expression must be numeric. The value is **TRUE** if it is nonzero.

```
//if
if a == 1
{
   b = 10
}

//if and else
if a == 10 && b > 20 : b = 10
else
{  b = 0 }

//if elif else
if a == b+10
{
   ...
   b = 10
}
elif a > 2 { b = 100 }
elif a != 1 || b == 32 : b=1000
else : b = 0
```

Related links

- Statements

switch statement

The **switch** construction allow s you to perform different operations in case an expression has different values. The **switch** keyw ord is follow ed by the initial expression that is calculated and stored as the sw itch value. Then you enumerate **case** constructions in curly braces w ith all possible values and the source code that should be executed. One **case** can have several possible values separated w ith a comma in case of w hich it w ill be executed. After executing the **case** block w ith the matching value , the program goes to the statement coming after **switch**. The rest of **case** blocks are not checked.

```
switch a + b
{
   case 0, 1, 2
   {   ...   }
   case 3
   {   ...   }
   case 4,10,12
   {   ...   }
}
```

If you w ant to execute some operations in case none of the **case** blocks is executed, insert the **default** construction at the end of **switch**. The **default** statement can appear only once and should come after all **case** statements.

```
switch ipar
{
   case 2,4,8,16,32
   {   ...   }
   case k, k + 1
   {   ...   }
   default
   {
      ...
   }
}
```

Additional features

The **switch** construction can be used not only for numeric expressions, but also for any types supporting the comparison operation **==**.

The same as **case**, it is possible to use the **label** label for an unconditional jump inside **switch**. Labels that appear after the case keyw ord enable you to enter the appropriate case case-block from another case-block.

```
switch name
{
   case "John", "Steve"
   label a0
   {
      ...
   }
   case "Laura", "Vanessa"
   {
      ...
      if name == "Laura" : goto a0
   }
   default
   {
      ...
   }
}
```

Related links

- Statements
- label and goto instructions

while and do statements

while

The **w hile** statement is a simple loop. The w hile statement has the follow ing parts: a **w hile** keyw ord, a conditional expression and a loop body (block). The execution of the loop body is repeated until the value of the expression evaluates to FALSE. The loop is never executed, if the value is zero w hen the test is performed for the first time.

```
a = 0
while a < 5
{
   ñ += a
   a++
}
```

do-while

The **do-w hile** statement contains the **do** keyw ord, a loop body, the **w hile** keyw ord and a conditional expression. The execution of the loop body is also repeated until the value of the expression evaluates to FALSE. Unlike the **w hile** statement, the test is performed after the execution of the loop body is completed and the iteration occurs at least once.

```
a = 4
do
{
   ...
   a--
} while a
```

There are special operators for the loop terminating w hen it is required. See more details on the return, break, continue instructions page.

Related links

- Statements
- return, break, continue instructions

for and fornum statements

for

The **for** statement consists of the **for** keyw ord, a sequence of three expressions separated by commas, a loop body.

```
for exp1, exp2, exp3
{
   ...
}
```

exp1 is an optional initialization expression. It is usually used for assigning the initial value to the counter variable.
exp2 is a conditional expression. The loop executes as long as the condition is TRUE.
exp3 is an optional increment expression. Actually, this expression increments or decrements the value of the counter.

The statement defined above can be performed w ith the help of the w hile loop as follow s:

```
exp1
while exp2
{
   ...
   exp3
}
```

The follow ing loops does the same actions.

```
for i=0, i<100, i++
{
   a += i
}

i = 0
for , i<100,
{
   a += i++
}
```

fornum

If the loop counter i is incremented by one and the highest value of the counter is defined before the loop iteration starts, the **fornum** statement is used in place of the **for** statement.

The **fornum** keyw ord is follow ed by a counter variable name, then the assignment operator and the expression (the initial value of the counter) can be used. If there are not any assignment operators, the initial value of the counter remains unchanged. Any integer should be treated as a counter variable. A comma delimited expression is specified,its result defines the loop termination. This expression is evaluated once before the loop iteration starts. The loop executes as long as the value of the counter is less than the value of the expression. Then the loop body follow s. By default, the increment operation (value of the counter is incremented by one) is appended to the loop body by the compiler.

```
fornum i=0,100
{
   a += i
}
```

Related links

- Statements
- return, break, continue instructions

foreach statement

The **foreach** loop is used to w ork w ith objects containing some number of elements. The type of an object must have the **first, next, eof** methods. See more details on the System type methods page. With the **foreach** construction, it is possible to go through all elements in the initial object.

You specify the name of the variable that w ill point to each next element after the **foreach** keyw ord. After that the object including the loop and then the body of the loop come separated w ith a comma. If the object contains elements of the numeric type, the index variable w ill contain values. If the object consists of items of the structural type, the index variable w ill point to each next element. If you change the index variable in this case, the corresponding element in the object w ill be changes as w ell.

```
arrstr names = %{"John","Steve","Laura", "Vanessa"}

foreach curname,names
{
   print("\( curname )\n")
}
```

Related links

- Statements
- System type methods

return, break, continue instructions

return

The **return** instruction is used either to return a function value or to terminate the execution of a function. The exit may be from anyw here w ithin the function body, including loops or nested blocks. If the function returns a value, the **return** instruction is required, furthermore it contains the expression of the appropriate type.

```
func uint myfunc
{
   ...
   fornum i, 100
   {
      if error : return 0
      ...
   }
   return a + b
}
```

break

The **break** instruction terminates the execution of the loop. **break** is likely to be located w ithin nested blocks. If a program contains several nested loops, break w ill exit the current loop.

```
while b > c
{
   for i = 100, i > 0, i--
   {
      if !myfunc( i )
      {
          break   //exit from for
      }
   }
   b++
}
```

continue

The **continue** instruction may occur w ithin loops and attempts to transfer control to the loop expression, w hich is used to increment or decrement the counter (for the follow ing loops: for, fornum, foreach) or to the conditional expression (for w hile and do-w hile loops); moreover, the execution of the loop body is not completely executed. The instruction executes only the most tightly enclosing loop, if this loop is nested.

```
fornum i, 100
{
   if i > 10 && i < 20
   {
      continue
   }
   a += i // The given expression is not evaluated if i>10 and i<20
}
```

Related links

- w hile and do statements
- for and fornum statements
- foreach statement

label and goto instructions

The **label** and **goto** insrustions perform an unconditional transfer of control w ithin the function body.

label

The appearance of the **label** instruction in the source program declares a label. The keyw ord **label** is follow ed by a name - an identifier label. Labels define w here to jump for the **goto** command. The label has scope limited to the block in w hich it is declared, therefore the goto instruction transfers control to the label either inside the current block or in blocks of higher levels. Control transferred to the label may occur before the label is declared.

goto

You can use the **goto** command to jump to the specified label. You should specify the name of the label to continue executing the program from after each **goto** keyw ord.

```
func myfunc
{
   ...
   {
     goto mylabel
     ...
     label mylabel
     ...
     goto finish
   }
   ...
   label finish
}
```

with statement

The **with** construction allows you to simplify addressing the fields of a variable of the structural type. Let us take the following example.

```
customer mycust

mycust.id = i++
mycust.name = "John"
mycust.country = "US"
mycust.phone = "999 999 999"
mycust.email = "john@domain.com"
mycust.check = mycust.id + 100
```

As you can see, you have to specify the name of the variable each time. **with** allows you to drop the variable name inside its block. To do it, specify the variable name after the **with** keyword and you will be able to specify only the point and the name of the corresponding field in curly braces. You can embed **with** constructions inside each other.

```
customer mycust

with mycust
{
   .id = i++
   .name = "John"
   .country = "US"
   .phone = "999 999 999"
   .email = "john@domain.com"
   .check = .id + 100
}
```

Arithmetic operators

There are three groups of arithmetic operators.

Arithmetic operators

+	Addition. **10 + 34 = 44**
-	Subtraction. **100 - 25 = 75**
*****	Multiplication. **11 * 5 = 55**
/	Division. Dividing one integer into another, any fractional portion is truncated. **10 / 3 = 3**
%	Residue of division. The operation **a % b** returns the remainder (modulus) obtained by dividing a into b or 0, if result is a w hole number. The modulus operator % is only used to perform division of tw o integers. **14 % 4 = 2**
-(un)	Unary negation operator. This operation change a sign of integer or decimal numbers. **-10 = -10**

```
a = ( 54 + b ) * (( 2*c - 235 ) / 3 )
b =  a % 10 + 0xFF00
```

Increment and decrement operators

The operators **++** and **--** are unary operators and deal w ith only integers.

++	The increment operator. This operator is expressed in tw o notations: the prefix-form **++i** and the postfix form **i++**. In the prefix form, variable i is incremented by the integer value 1, new value of variable i is used in the expression evaluation; in the postfix form, the increment takes place after the value of variable i is used in the expression evaluation.
--	The decrement operator. The prefix notation is **--i** - the variable is decremented by one and the result is this decremented value. The postfix notation is **i--** - the decrement occurs after the value of variable is used in expression evaluation.

```
i = ++k
while i++ < 100
{
   sum += l--
}
```

Bitwise operators

These bitw ise operators perform manipulation on integer operands.

&	Bitw ise-AND (binary). **0x124 & 0x107 = 0x104**
	)
^	Bitw ise-exclusive-OR (binary). **0x124 ^ 0x107 = 0x23**
<<	Bitw ise shift left (binary). The bitw ise shift operators shift their left operand left or right by the number of positions the right operand specifies, bits vacated by the shift operation are zero-filled. **0x124 << 2 = 0x490**
>>	Bitw ise shift right (binary). **0x124 >> 2 = 0x92**
~	Bitw ise negation (unary). **~0x124 = 0xFFFFFEDB**

```
a = b & 0x0020 + ñ | $FLAG_CHECK
rand=( 16807 * rand ) % 0x7FFFFFFF ) % ( end - begin + 1 ) + begin
```

You can define these operators for any types. See more details on the Redefining operator operations page.

Related links

- Redefining operator operations

Logical operators

Logical operators

These logical operators perform manipulation on integer operands. The result of a logical operation is the integer of uint type, w hich has either **0** value -the result is **FALSE** or **1** value - the result is **TRUE**.

&&	Logical-AND (binary). Returns 0 if at least one operand equals 0.
||	Logical-OR (binary). Returns 1 if at least one operand does not equal 1.
!	Logical negation (unary). Returns 0 if the operans is not 0, and returns 1 if the operand equals 0.

```
if a < 10 && ( b >= 10 || !c ) && k
{
   if a || !b
   {   ...    }
}
```

Comparison operators

The result of this operation is the integer of uint type, w hich has either **0** value -the result is **FALSE** or **1** value - the result is **TRUE**.

==	Equality.
!=	Inequality.
>	Greater-than.
<	Less-than.
>=	Greater-than-or-equal-to.
<=	Less-than-or-equal-to.
%<, %>, %<=, %>=, %==, %!=	The operators are used to compare tw o operands alternatively. For example, using these operators you can compare strings by a case-insensitive value (no uppercase preference).

```
while i <= 100 && name %== "john"
{
   if name == "stop" : return i < 50
   ...
}
```

You can define these operators for any types. See more details on the Redefining operator operations page.

Assignment operators

The assignment operators are considered to be the binary operators. The left-hand operand of an assignment operation must be a variable, item of array, field of structure etc. These operators have right-to-left associativity.

=	Simple assignment.
+=	Addition assignment. **a += b => a = a + b**
-=	Subtraction assignment. **a -= b => a = a - b**
*=	Multiplication assignment. **a *= b => a = a * b**
/=	Division assignment. **a /= b => a = a / b**
%=	Remainder assignment. **a %= b => a = a % b**
&=	Bitw ise-AND assignment. **a &= b => a = a & b**
	=)
^=	Bitw ise-exclusive-OR assignment. **a ^=b => a = a ^ b**
>>=	Right-shift assignment. **a >>= b => a = a >> b**
<<=	Left-shift assignment. **a <<= b => a = a << b**

As you have already noticed, except "simple assignment" you can perform the assignment w ith an operation, that is after a binary operation of the right-hand operand and the left-hand operand is performed, the result is assigned into the left operand.

```
a = 10
a += 10 + 23 // a = 43
a *= 2  // a = 86

if a = 2 // TRUE !!!
{...}
if a == 2 // TRUE if a equals 2
{...}
```

One and the same expression can contain several assignment operations, each of w hich returns the assigned value. In this case, the assignment operation is performed from right to left.

```
a = 10 + b = 20 + c = 3
// result: ñ=3, b=23, a=33
a = ( b += 10 )
```

You can define these operators for any types. See more details on the Redefining operator operations page.

Type reduction

The as operator

The **as** operator executes tw o functions: to assign a value to a variable and to redefine a type. This operator is binary, that has right-to-left associativity. The left-hand operand must be a local variable of uint type. Depending on the value of the right-hand operand, it can be operated in tw o possible w ays:

The first way

The right-hand operand is a structure type name. A value of the local variable is not modified, but its type is redefined w ith the required one; the variable is assumed to store an object's address, moreover this variable can be treated as the object, ignoring the pointer operation **->**.

```
str  mystr
uint a

a = &mystr
a as str
a = "New value"
```

The second way

The right-hand operand is an expression that returns an object. An address of the object is assigned to the variable, w hich redefines its type w ith the object's type. The object type must be different from numeric types.

```
str mystr
uint a

a as mystr
a = "New value"
```

The variable type w ill be redefined either until the end of the current block or until the next operation **as** w ith the variable occurs.

Operator ->

Often you need just to specify that a variable is of a certain structural type. In this case, you can use the **->** statement w ith the name of the required structural type. Together w ith the type name, you can specify the dimension in square brackets and the type of items w ith the help of **of**. The variable **->** is applied to can be of not only the **uint** type, but any structural type.

```
func myfunc( uint mode, uint obj )
{
   str  ret
   uint val

   switch mode
   {
      case 0: myproc( obj->arrstr )
      case 1: print( obj->str )
      case 2: obj->mytest.mytest2str( ret )
      case 3
      {
         val = (obj->arr[,] of ubyte )[1,1]
      }
   }
}
```

Type ñonversion

By default, only integral types **byte,ubyte, short, ushort, int, uint** are automatically converted into each other, you should use express conversion for other types. For an expression of the type to be converted to another type, it is necessary to specify a type name, to w hich data w ill be converted, and the expression enclosed in parentheses; moreover, the conversion w ill occur if the specified source type has the appropriate method. See more details about such methodson the Method declaration: method page.

```
str a = "10"
uint b

b = uint( a )
```

Related links

- Method declaration: method
- Fields and pointers

Fields and pointers

Addressing fields

The **.** statement (dot) is used to get or set the value of a field or to call a method or a property. You should specify the name of the field or property after the dot. You should specify parameters in parentheses in case you call a method.

```
type customer
{
   str    name, last_name
   uint   age
   arrstr phones[ 5 ]
}
...
customer cust1
cust1.name = "Tom"
cust1.age = 30
cust1.phones[ 0 ] = "3332244"
cust1.process()
```

Addresses and pointers

The unary operator **&** gives the address of a local or global variable as w ell as the address (identifier) of a function. The operation returns the value of uint type. How ever, if the result of any operation is an object, for example the function w hich returns a string, the address-of operator is also apllied to the obtained object. The address-of operator **&**, applied to the object (structure), returns the address of the required object and is used for a type cast to uint type.

```
uint a b
str mystr
...
a = &mystr
b = &getsomestr
b->func( a ) // equals getsomestr( mystr )
```

You should use the **->** statement to get a value by its address. The first operand must be the name of the numeric type and the left operand must point to the value of the corresponding numeric type.

```
int a = 10, b
uint addra

addra = &a
b = addra->int // b = 10
addra->int = 3 // a = 3
```

Array element operation

Many structures or objects can include elements of other types. You can use square brackets **[ ]** to access the elements of an object (array elements, string characters). If an object is a multidimensional one, its dimensions are separated w ith commas. Elements are counted starting from zero. For you to be able to apply this operation to a variable, its type must have the corresponding **index** methods. See more details on the System type methods page.

```
arr myarr[ 10, 10, 10] of byte
str mystr = "abcdef"

myarr[ i, k+3, 4 ] = 'd'
myarr[ 0, 0, 0 ] = mystr[i]
```

Related links

- System type methods
- Type reduction

Calling functions and methods

Calling function and method

A function call includes the name of the function being called and the arguments enclosed in parentheses and separated by commas. If the function does not have any arguments, the empty parentheses follow the function name. If either a function or a method returns a value, the function or method call may be used in the expression. The **.** operator is used to call a method, then the arguments are listed in the same w ay as the arguments of a function: the variable, w hich stores a structure, is follow ed by the point, then you specify the name of the method and the arguments enclosed in parentheses.

```
a = my.mymethod( myfunc( a, b + c ))
a = b->mystruct.mymethod( d )
```

Function call via a pointer

A variable of **uint** type can store the address (identifier) of a function. In order to get the address of a function, the **&** operator is used, w hich is follow ed by the name of the function w ithout parentheses. A function call includes the **->func** and the arguments listed inside parentheses. In this case, you ought to keep an eye on the number and types of the arguments, because the compiler is not able to verify the arguments. The same w ay you can call methods and operators.

```
a = &myfunc
a->func( c, d )
```

Gentee allow s you to call functions by their address. For example, you can get the function address w hen you use Window s API function **GetProcAddress**. Use the **->stdcall** and the arguments listed inside parentheses. If the function has **cdecl** type then use the keyw ord **cdecl** instead of **stdcall**.

```
a = GetProcAddress( mylib, "myfunc".ptr())
a->stdcall( 1, b )
```

Text-function call

The **@** operator is applied to call a text-function. This operation can be either unary or binary.

Unary operation

@ name(...)
If a text-function is called from the simple function or method, the text function w ill output data to a console. In case of calling the function from another text-function, data w ill be outputted to the same place as the current text-function. A text function may be called w ithout the unary @ operator. In other w ords, the text function is called in the same w ay as the simple function.

Binary operation

dest @ name(...)
If the **@** operator is applied as a binary operator, the left-hand operand must be a string, to w hich data w ill be outputted from the text function. In the example illustrated above, dest is a variable or an expression of the str type. Actually, data are appended to a destination string.

The **@** operator is used not only for the text-function call, but also for the string output. If the right-hand operand is either a variable or an expression of the str type, this string w ill be outputted to the console or w ill be appended to a destination string as described above.

```
str a
@mytext( 10 )     // Console output
a @ mytext( 20 ) // string output
@"My text"        // print( "My text" )
```

Related links

- Fields and pointers
- Function declaration: func
- Method declaration: method
- Declaring text function

The conditional operator ?

The conditional operator **?** operates in the similar w ay as the **if-else** statement, but the conditional operator can be located in the expression. The operator is a ternary operator (it takes three operands). The operands separated by commas are enclosed in parentheses; the first logical (integer) expression is evaluated. If the value is nonzero (TRUE), the second expression w ill be evaluated, the value of w hich w ill be a result of the conditional operation. Otherw ise, the third operand w ill be evaluated, the value of w hich w ill be returned.

```
r = ?( a == 10, a, a + b )
if a >= ?( x, 0xFFF, ?( y < 5 && y > 2, y, 2*b )) + 2345
{
   ...
}
```

Late binding operation

The **~** operation is used for late binding. This operation has a lot in common w ith the . operator (used to access a field value or method call); how ever, at compile time it is sometimes difficult to define all methods and fields of an object, w hereas w hile executing a program a particular method of an object is called for being assigned a field/method name, types and values of parameters. The late binding operation is actually applied for **COM objects**.

An object identifier is the left-hand operand of the ~ operation, that is used for maintaining late binding; the right-hand operand is a field/method name, that is used either for setting up a property (e.g. **excapp~Visible**) or for calling a method (e.g. **excapp~Cells(3,2)**).

An object can maintain the follow ing kinds of late binding:

- elementary method call **excapp~Quit**, w ith/w ithout parameters;
- set value **excapp~Cells(3, 2) = "Hello World!"**;
- get value **vis = uint(excapp~Visible)**;
- call chain **excapp~WorkBooks~Add**, equals the follow ing expressions
  tmpobj = excapp~WorkBooks
  tmpobj~Add

A shortcoming of late binding is that the compiler cannot check if either fields/methods or types are specified properly; it causes problems for troubleshooting.

Have a look at the example of using late binding, w here the **COM library** is applied.

```
include { "olecom.ge"}
...
oleobj excapp
excapp.createobj( "Excel.Application", "" )
excapp.flgs = $FOLEOBJ_INT
excapp~Visible = 1
excapp~WorkBooks~Add
excapp~Cells( 3, 2 ) = "Hello World!"
```

Table of operator precedence

As a rule, all statements are executed from left to right, but there is such a concept as statement priority. If the next statement has a higher priority, the statement w ith a higher priority is executed first. For example, multiplication has a higher priority and **4 + 5 * 2** is **14**, but if w e use parentheses, **(4 + 5) * 2** is **18**.

Character operation	Associativity
The highest priority	
() [] . ~ ->	Left to right
! &(un) *(un) -(un) ~(un) ++ -- @(un)	Right to left
% * /	Left to right
+ - @	Left to right
<< >>	Left to right
< > <= >= %< %> %<= %>=	Left to right
!= == %== %!=	Left to right
&	Left to right
^	Left to right
|	Left to right
&&	Left to right
||	Left to right
?(,,)	Left to right
= += -= *= /= %= &= |= ^= >>= <<= as	Right to left
The low est priority	

Parentheses **()** change the order in w hich expressions are evaluated. You can use square brackets in order to deal w ith the elements of the array or the indexed elements, for example, a character in a string. The unary operators include **!, &, *, -, ~, ++, --**. It is the prefix notation that is used for all unary operators, except increments. As for increment operations **++** and **--**, they can be occured either in the prefix or in the postfix notation. The follow ing operators **&, *, -, @, ~** are likely to be binary as w ell as unary operators. All other operators are binary (taking tw o operands).

Gentee Language in BNF

You can use ANSI character set from 0 to 255 in a source code. ANSI character set from 32 to 128 are specified in the diagram defined above, other characters are represented in hexadecimal notation, for example 0x09 - a tab character. Some preprocessor commands are not shown in these diagrams.

<binary digit> ::= '0' | '1'

<decimal digit> ::= <binary digit> | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

<hexadecimal digit> ::= <decimal digit> | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'a' | 'b' | 'c' | 'd' | 'e' | 'f'

<byte> ::= <hexadecimal digit><hexadecimal digit>

<decimal number> ::= <decimal digit> {<decimal digit>}

<hexadecimal number> ::= '0' ('x' | 'X') <hexadecimal number> {<hexadecimal number>}

<binary number> ::= '0' ('b' | 'B') <binary number> {<binary number>}

<character code> ::= '''<any character>'''

<floating point number> ::= <decimal number>'.'[<decimal number>]

<real number> ::= ['-'] (<floating point number> | <floating point number> ('e' | 'E') ['+' | '-'] <decimal number>) ['d' | 'D']

<natural number> ::= <decimal number> | <hexadecimal number> | <binary number> | <character code>

<integer number> ::= ['-'] <natural number> ['l' | 'L']

<number> := <integer number> | <floating point number> | <real number>

<letter> ::= 'A' | 'B' | ... | 'Z' | 'a' | 'b' | ... | 'z' | 0x80 | 0x81 | ... | 0xFF

<space> ::= 0x20

<tabulation> ::= 0x09

<end-of-line> := 0x0D 0x0A

<delimiter> ::= '!' | '"' | '#' | '$' | '%' | '&' | ''' | '(' | ')' | '*' | '+' | ',' | '-' | '.' | '/' | '<' | '=' | '>' | '?' | '@' | '[' | '\' | ']' | '^' | '_' | '|' | '}' | '{' | '~' |
<tabulation> | <space> | <end-of-line>

<character> ::= <decimal digit> | <letter> | <delimiter>

<name> ::= (<letter> | '_') {<letter> | '_' | <decimal digit>}

<function name> ::= <name>

<variable name> ::= <name>

<type name> ::= <name>

<field name> ::= <name>

<method name> ::= <name>

<attribute name> ::= <name>

<macro name> ::= <name>

<str character> ::= <tabulation> | <space> | '!' | '#' | ... | '[' | ']' | ... | 0xFF

<ext str character> ::= <str character> | '\'

<any character> ::= <ext str character> | '"'

<macrostring element> ::= {<str character>|<end-of-line>| '$'<macro name>}

<const string element> ::= <str character> | <end-of-line> | '\\' | '\"' | '\a' | '\b' | '\f' | '\n' | '\r' | '\t' | '\l' | '\0'<byte> | '\$'<macro name>'$' | '\$"' {<macrostring element>} '"' | '\#' | '\=' ('\' | '/' | '~' | '^' | '&' | ':') | '\%['[{<ext str character>}]']'{<ext str character>}[{<ext str character>}] | '\[' {<any character>}']' | '\<'{<macrostring element>}'>'

<string element> ::= {<ext str character>} | '\('<expression>')'

<const string> ::= '"' { <const string element> | '\!'{<const string element>| '"'}'\!' }'"'

<string> ::= '"' { <string element>| '\!'{<string element>| '"'}'\!' }'"'

<const binary data element> ::= '\h'<space> [('2' | '4' | '8') <space>] | '\i'<space> [('2' | '4' | '8') <space>] | <hexadecimal digit> {<hexadecimal digit>} { (<space> | ',' | <end-of-line>) <hexadecimal digit> {<hexadecimal digit>} } | <integer number> { (<space> | ',' | <end-of-line>) <integer number> } | '\'<const string> | '\[' {<any character>}']' | '\$'<macro name>'$' | '\$"' {<macrostring element>} '"' | '\<'{<macrostring element>}'>'

<binary data element> ::= <const binary data element> | '\('<expression>')'

<const binary data> ::= ''' {<const binary data element>} '''

<binary data> ::= ''' {<binary data element>} '''

<const collection> ::= '%{' <constant> {','<constant>} '}'

<collection> ::= '%{' <expression> {','<expression>} '}'

<constant> ::= <number> | <const string> | <const binary data> | <const collection>

<array> ::= ['[' {','} ']'] [of <type name>]

<object> ::= <variable name> | <pointer> | <array element> | <field> | <function call> | <method call> | <expression> | <late binding>

<pointer> ::= <expression> '->' <type name> [<array>]

<parameters> ::= <expression> {','<expression>}

<array element> ::= <object>'['<parameters>']'

<field> ::= [<object>]'.'<field name>

<late binding> ::=<object>'~' (<field name> | <method name>'(' [<parameters>] ')')

<function call> ::= (<function name> | <expression>'->'func) '(' [<parameters>] ')'

<method call> ::= [<object>]'.'<method name>'(' [<parameters>] ')'

<lvalue> ::= <object> | <variable name> | <pointer> | <array element> | <field>

<as operation> ::= <variable name>'as' ((<type name>[<array>]) | <object>)

<operand> ::= <lvalue> | <constant> | <string> | <binary data> | <collection> | <function call> | <method call> | <type name> | <late binding>

<increment operator> ::= '++' | '--'

<assignment operator> ::= '=' | '%=' | '&=' | '*=' | '+=' | '-=' | '/=' | '<<=' | '>>=' | '^=' | '|='

<unary operator> ::= '+' | '-' | '*' | '!' | '~' | '@'

<binary operator> ::= '==' | '!=' | '>' | '<' | '<=' | '>=' | '&&' | '||' | '&' | '%' | '*' | '/' | '+' | '-' | '<<' | '>>' | '^' | '%==' | '%!=' | '%>' | '%<' | '%<=' | '%>=' | '@'

<operator> ::= <increment operator> | <assignment operator> | <unary operator> | <binary operator>

<assignment expression> ::= <lvalue><assignment operator><expression>

<lvalue expression> ::= '&'<lvalue> | '&'<function name> | <increment operator><lvalue> | <lvalue><increment operator>

<question> ::= '?''(' <expression>',' [<expression>] ',' [<expression>] ')'

<expression> ::= <operand> | <assignment expression> | <lvalue expression> | <expression><binary operator> [<end-of-line>] <expression> | '('<expression>')' | <unary operator><expression> | <as operation> | <question>

<variable declaration> ::= <variable name> ['['<expression> { [','] <expression>} ']'] [of <type name>]

<variable list> ::= <variable declaration> ['='<expression>','<variable list>] | <variable declaration> [[',']<variable list>]

<variables declaration> ::= <type name><variable list><end-of-line>

<if> ::= **if** <expression><block> {**elif** <expression> <block>} [**else** <block>]

<while> ::= **while** <expression><block>

<dowhile> ::= **do** <block> **while** <expression>

<for> ::= **for** [<expression>] **','**<expression>**','** [<expression>] <block>

<fornum> ::= **fornum** <variable name> [**'='**<expression>] **','**<expression><block>

<foreach> ::= **foreach** [<type name>] <variable name> [<array>]**','**<expression><block>

<return> ::= **return** [<expression>]

<label> ::= **label** <name>

<goto> ::= **goto** <name>

<switch> ::= **switch** <expression>**'{'** {**case** <expression> {**','**<expression>} {<label>} <block>} [**default** {<label>} <block>] **'}'**

<block contents> ::= <block command> {<end-of-line><block command>}

<block> ::= **'{'**<block contents>**'}'**

<block command> ::= {<label>} (<block> | <expression> | <variables declaration> | <if> | <for> | <fornum> | <while> | <dowhile> | <foreach> | <switch> | **break** | **continue** | <return>)

<parameter declaration> ::= <variable name> [<array>]

<parameters declaration> ::= <type name><parameter declaration> { [**','**] <parameter declaration>} [<parameters declaration>]

<attributes> ::= **'<'**<attribute name> { [**','**] <attribute name>} **'>'**

<subfunction> ::= **subfunc** [<type name>] <function name> [**'('** [<parameters declaration>] **')'**] <block>

<function body> ::= **'{'** (<block command> | <subfunction>) <block command> {<end-of-line> (<block command> | <subfunction>) } **'}'**

<function declaration> ::= **func** [<type name>] <function name> [<attributes>] [**'('** [<parameters declaration>] **')'**]

<func> ::= <function declaration><function body>

<method declaration> ::= **method** [<type name>] <type name>**'.'**<method name>[<attributes>] [**'('** [<parameters declaration>] **')'**]

<method> ::= <method declaration><function body>

<property declaration> ::= **property** [<type name>] <type name>**'.'**<method name> [<attributes>] [**'('** [<parameters declaration>] **')'**]

<property> ::= <property declaration><function body>

<operator declaration> ::= **operator** <type name> <operator> [<attributes>] **'('** <parameters declaration> **')'**

<operator function> ::= <operator declaration><function body>

<text-function declaration> ::= **text** <function name> [<attributes>][**'('** [<parameters declaration>] **')'**]

<text-function body> ::= { <const string element> | **'\@'**<function name>**'('** [<parameters>]> | **'\('**<expression>**')'** | **'\{'**(<block command> | <subfunction>) <block command> {<end-of-line> (<block command> | <subfunction>) }**'}'** }[**'\!'**]

<text> ::= <text-function declaration><end-of-line><text-function body>

<macro declaration> ::= <macro name> [**'='** (<constant>|<name>)]

<define> ::= **define** [<name>][<attributes>] **'{'**<macro declaration> {<end-of-line><macro declaration>} **'}'**

<macro expression> ::= **'$'**<macro name> | <constant> | **'!'**<macro expression> | **'('**<macro expression>**')'** | <macro expression> (**'&&'** | **'||'** | **'=='** | **'!='**) <macro expression>

<ifdef> ::= **ifdef** <macro expression> **'{'** ... **'}'** { **elif** <macro expression> **'{'** ... **'}'** } [**else** **'{'** ... **'}'**]

<file name> ::= **'"'** {<str character>} **'"'**

<include> ::= **include** **'{'**<file name> {<end-of-line><file name>} **'}'**

<imported function declaration> ::= [<type name>] <function name> **'('** [<type name> { **','**<type name> }] **')'** [**'->'** <function name>]

<import> ::= **import** <file name> [<attributes>] **'{'** <imported function declaration> { <end-of-line><imported function declaration> } **'}'**

<field declaration> ::= <field declaration> [<array declaration>]

<fields declaration> ::= <type name><field declaration> {[**','**] <field declaration>}<end-of-line>

<type> ::= **type** [<attributes>] **'{'**<fields declaration>{<fields declaration>} **'}'**

<array declaration> ::= [**'['**<natural number> { [**','**] <natural number>} **']'**] [**of** <type name>]

<global variable declaration> ::= <variable name> [<array declaration>][**'='** <constant>]

<global variables declaration> ::= <type name><global variable declaration> {[**','**] <global variable declaration>}<end-of-line>

<global> ::= **global** **'{'** {<global variables declaration>} **'}'**

<public> ::= **public**

<private> ::= **private**

<extern> ::= **extern** **'{'** {(<function declaration> | <method declaration> | <operator declaration> | <property declaration>)<end-of-line>} **'}'**

<command> ::= <define> | <func> | <method> | <text> | <operator function> | <property> | <include> | <type> | <global> | <extern> | <import> | <public> | <private> | <ifdef>

<program> ::= <command> {<end-of-line><command>}

How to launch Gentee

This section deals w ith the follow ing questions

- Ways to run Gentee programs.
- Compiler configuration and options.
- Creating EXE files.
- Integration w ith other programming languages. In particular, using gentee.dll.

Table of contents

Quick Launch

You may use any text editor to w rite and edit the source code of your Gentee program, w hich you should then save w ith the **.g** extension. You w ill then be able to run it easily in Explorer or any file manager, by double-clicking it or pressing the *Enter* key. Files w ith the **.ge** extension (compiled Gentee programs) are run in the same w ay. The **.ge** extension allow s you to run programs faster, because they do not require additional compilation.

You w ill find some sample programs in the Gentee source files. Using a file manager or Explorer, open the directory to w hich you installed Gentee (the default is *C:\Program Files\Gentee*), and select the **Samples** subdirectory to see a list of examples.

You can create shortcuts in **Start->Programs** or **Desktop** for frequently-used Gentee programs, to launch them more easily and quickly. Use the extension **.g** or **.ge** to make the file executable.

Related links

- Using '#!' command
- Compilation profiles

Launch from Command Line

A program in the Gentee language is compiled and run w ith the console application **gentee.exe**. Command line options don't cover all gentee features. So, use Compilation profiles for specifing advanced parameters of the compilation.

`gentee.exe [switches] <Gentee file> [arguments]`

`switches`

Compiler options. You can use the follow ing options during compilation.

-a	The compiler translates the bytecode to assembler. At this moment, it does NOT translate ALL bytecode to assembler but this option can increase the speed of some programs in several times.
-c	Only compilation. Do not run the program after the compilation.
-d	Add the debug information into the byte-code.
-m <macros>	Defining compilation macros. You can define the necessary compilation macros after **-m** . You should use **'\'** before quotation marks. Macro definitions must be separated w ith a semicolon. Example: **-m "MODE=1;NAME=\"My Company, Inc\""**
-f	Create a **.ge** file w ith byte code. It w ill be created in the same directory and w ill have the same name.
-n	Ignore the command '#!' in the first string of the file. See Using '#!' command.
-o <GE or EXE filename>	Specify a name for the compiled file. In this case, next part should be the name of the output file. This feature is used if you w ant the file w ith bytecode or exe file to have a name (or destination) different from the source file. By default, the compiled byte-code is stored in the file w ith **.ge** extension.
-p <profile name>	Use profile parameters from the file **gentee.ini**. See Compilation profiles.
-s	Do not display service message during compilation or running.
-t	Automatically convert text to the ÎÅÌ encoding (DOS encoding) w hen displaying it on the console.
-d	Include debug information into the byte-code.
-w	Wait for pressing key at the end of the compilation.
-z[d][n][u]	Optimize a byte-code (-f or -x compatible) **-zd** - Delete defines. **-zn** - Delete names. **-zu** - Delete no used objects. **-z** equals **-zdnu**. Combine -zd, -zn and -zu.
-x[d][g][a][r]	Create executable EXE file. **-xd** - Dynamic usage of gentee.dll. **-xg** - Make a gui application. In default a console application is created. **-xa** - Specify this option if your program or its part is compiled w ith -a option. **-xr** - Specify it if you w ant that the bytecode is translated to assembler each time w hen you run the program. Don't use this parameter w ith -a option. **-xdgr** - Combine -xd,-xr and -xg.
-i <icon file>	Link .ico file (-x compatible). Example *-i "c:\data\myicon.ico"*
-r <res file>	Link .res file (-x compatible). Example *-r "c:\data\myres.res"*

`Gentee file`

This parameter is a required one and must define the name of the compilation file or the file w ith byte code to be executed.

`arguments`

All parameters after the name of the file being run are command line parameters that w ill be passed over to the program being run.

Examples

```
gentee.exe -t myfile.g
gentee.exe -s myapp.g "command line argument" 10 20
gentee.exe -o "c:\temp\app.ge" -c myapp.ge "command line argument"
gentee.exe -p myprofile "c:\my programs\myfile.g"
```

Related links

- Using '#!' command
- Compilation profiles

Using #!' command'

Under Linux, '#!' in the first line is used to start the compiler. Under Window s, you can also use the first line in a file to start any programs, including those used for compiling w ith the necessary parameters. If you click such a .g file or press Enter, the specified command line w ill be executed. It allow s you to avoid using additional batch files (.bat) and specify compiling options different from default options.

You can specify both absolute and relative paths to the program and the file you w ant to start. You can specify **%1** as the full name of the current file. If the path contains spaces, you should enclose it in double quotation marks.

Examples

```
#!gentee.exe -s hello.g
#!gentee.exe -t -f "%1"
#!"C:\My Application\my.bat" "%1"
#!ge2exe.exe "%1"
```

Using profiles

You can specify profile parameters at the beginning of the source .g file. Parameters must be described from the first line and the first character of the line must be . See names of parameters in Compilation profiles.

Example

```
#output = %EXEPATH%\gentee-x.exe
#norun = 1
#exe = 1 d g
#optimizer = 1
#wait = 3
#res = ..\..\res\exe\version.res
```

Compilation profiles

Besides specifying compilation options directly w hen running Gentee program, you can store all necessary parameters in a separate profile and you w ill only have to specify the name of this profile w hen you run the compiler. Profiles must be described in the text file **gentee.ini** located in one directory w ith **gentee.exe**. The profile name is specified after the option **-p**. For example: **gentee.exe -p myoptions test.g**. By default, the compilation profile named **default** is used w hen you run a Gentee program.

You can specify a compilation profile at the beginning of your .g source file. See Using '#!' command for more details.

asm = <0 1>	If 1, the compiler translates the bytecode to assembler. At this moment, it does NOT translate ALL bytecode to assembler but this option can increase the speed of some programs in several times.
silent = <0 1>	If 1, do not display service messages during the compilation or launch.
charoem = <0 1>	If 1, convert strings into the OEM (DOS) encoding w hen displaying them on the console.
debug = <0 1>	If 1, the debug information w ill be included into the byte-code during the compilation.
gefile = <0 1>	If 1, create a **.ge** file during the compilation.
norun = <0 1>	If 1, do not run the program after the compilation.
numsign = <0 1>	If 0, ignore the first string w ith **#!** in the body of the program being run.
output = <.ge or .exe filename>	You can specify the full path and name of the creating **.ge** or **.exe** file here.
define = <macro = value>	The parameter is used to define compilation macros. You can specify some **define** parameters: **define1,define2,define3...**.
include = <.g or .ge file>	You can specify additional **.g** or **.ge** files that w ill be added at the beginning of the compilation. It is the same as using the **include** command in a Gentee program. You can specify some include files w ith **include1,include2,include3...**.
libdir = <directory>	The parameter allow s you to specify the search path for **.g** or **.ge** files included in the program. If the path is specified, it is enough to specify only the file name. You can specify some search directories w ith **libdir1,libdir2,libdir3...**.
wait = <0 1..n>	If 1, the compiler w ill w ait for pressing key at the end of the compilation. If you specify a number greater 1 then the compiler w ill be w ait <w ait> seconds and close the console w indow .
optimizer = <0 1 (d n u)>	If 1, the byte-code w ill be optimized. You can specify the additional parameters **d**, **n** or **u** after 1 divided by a space character. **d** - Delete defines. **n** - Delete names. **u** - Delete no used objects. For example: **optimizer = 1 d n u**
exe = <0 1 (d g r a)>	If 1, the executable EXE file w ill be created. You can specify the additional parameters **d g a r** divided by a space character. **d** - Dynamic usage of gentee.dll. **g** - Make a gui application. By default a console application is created. **a** - Specify this option if your program or its part w as compiled w ith asm option. **r** - Specify this parameter if you w ant that the bytecode is translated to assembler each time w hen you run the program. Don't use this parameter if your program has been compiled w ith asm option. For example: **exe = 1 d g r**
icon = <.ico file>	You can specify additional **.ico** files for EXE file. It is possible to specify some icon files w ith **icon1,icon2,icon3...**.
res = <.res file>	You can specify additional resource **.res** files for EXE file. It is possible to specify some resource files w ith **res1,res2,res3...**.
args = <parameter>	Command line parameter for launching of the program . It is possible to specify some command line parameters w ith **args1,args2,args3...**.

Additional features

You can use the follow ing predefined parameters.

%GNAME%	The name of the compiling Gentee file w ithout the extension.
%GPATH%	The full path to Gentee file.
%EXEPATH%	The full path to the **gentee.exe** compiler.

Example

```
[default]
```

```
charoem = 1
gefile = 0
libdir = %EXEPATH%\lib
libdir1 = %EXEPATH%\..\lib\vis
include = %EXEPATH%\lib\stdlib.ge

[myoptions]
charoem = 1
output = c:\My Files\Programs\%GNAME%.ge
libdir = %EXEPATH%\lib
include = %EXEPATH%\lib\stdlib.ge
include1 = c:\mylibs\mylib.g
define = MODE = 1
define1 = COMPANY = "My Company, Inc."
```

Library Reference

Table of contents

Array	Array.
Array Of Strings	Array of strings.
Array Of Unicode Strings	Array of unicode strings.
Buffer	Binary data.
Clipboard	These functions are used to work with the Windows clipboard.
Collection	Collection.
COM/OLE	Working with COM/OLE Object.
Console	Console library.
CSV	Working with CSV data.
Date & Time	Functions for working with date and time.
Dbf	This library is used to work with **dbf** files.
Files	File system functions.
FTP	FTP protocol.
Gentee API	Gentee API functions for the using of gentee.dll.
Hash	Hash (Associative array).
HTTP	HTTP protocol.
INI File	INI files.
Keyboard	These functions are used to emulate the work of the keyboard.
Math	Mathematical functions.
Memory	Functions for memory management.
ODBC (SQL)	Data Access (SQL queries) Using ODBC.
Process	Process, shell, arguments and environment functions.
Registry	Working with the Registry.
Socket	Sockets and common internet functions.
Stack	Stack.
String	Strings.
String - Filename	Filename strings.
String - Unicode	Unicode strings.
System	System functions.
Thread	This library allows you to create threads and work with them.
Tree	Tree object.
XML	XML file processing.

Array

Array. You can use variables of the **arr** type for w orking w ith arrays. The **arr** type is inherited from the **buf** type.

- Operators
- Methods

Operators

*** arr**	Get the count of items.
foreach var,arr	Foreach operator.
arr of type	Specifying the type of items.
arr[i]	Getting **[i]** item of the array.

Methods

arr.clear	Clear an array.
arr.cut	Reducing an array.
arr.del	Deleting item(s).
arr.expand	Add items to an array.
arr.insert	Insert elements.
arr.move	Move an item.
arr.sort	Sorting an array.

* arr

Get the count of items.

```
operator uint * (
   arr left
)
```

Return value

Count of array items.

Related links

- Array

foreach var,arr

Foreach operator. You can use **foreach** operator to look over items of the array.

```
foreach variable,array {...}
```

Related links

- Array

arr of type

Specifying the type of items. You can specify **of** type w hen you describe **arr** variable. In default, the type of the items is **uint**.

```
method arr.oftype (
   uint itype
)
```

Related links

- Array

arr[i]

- method uint arr.index(uint i)
- method uint arr.index(uint i, uint j)
- method uint arr.index(uint i, uint j, uint k)

Getting **[i]** item of the array.

```
method uint arr.index (
   uint i
)
```

Return value

The **[i]** item of the array.

---

arr[i,j]

Getting **[i,j]** item of the array.

```
method uint arr.index (
   uint i,
   uint j
)
```

Return value

The **[i,j]** item of the array.

---

arr[i,j,k]

Getting **[i,j,k]** item of the array.

```
method uint arr.index (
   uint i,
   uint j,
   uint k
)
```

Return value

The **[i,j,k]** item of the array.

Related links

- Array

arr.clear

Clear an array. The method removes all items from the array.

```
method arr arr.clear()
```

Return value

Returns the object w hich method has been called.

Related links

- Array

arr.cut

Reducing an array. All items exceeding the specified number w ill be deleted.

```
method arr.cut (
   uint count
)
```

Parameters

count	The number of items left in the array.

Related links

- Array

arr.del

- method arr.del(uint num)
- method arr arr.del(uint from, uint count)

Deleting item(s). The method removes an item w ith the specified number.

```
method arr.del (
   uint num
)
```

Parameters

num	The number of item starting from 0.

---

arr.del

The method removes items from the array.

```
method arr arr.del (
   uint from,
   uint count
)
```

Parameters

from	The number of the first item being deleted (from 0).
count	The count of the items to be deleted.

Return value

Returns the object w hich method has been called.

Related links

- Array

arr.expand

Add items to an array.

```
method uint arr.expand (
   uint count
)
```

Parameters

count	The number of items being added.

Return value

The index of the first added item.

Related links

- Array

arr.insert

- method arr.insert(uint id)
- method uint arr.insert(uint from, uint count)

Insert elements. The method inserts an element into the array at the specified index.

```
method arr.insert (
   uint id
)
```

Parameters

id	The index of the element needs to be inserted.

arr.insert

The method inserts elements into the array at the specified index.

```
method uint arr.insert (
   uint from,
   uint count
)
```

Parameters

from	The index of the first inserted element starts at zero.
count	The amount of elements are required to be inserted.

Return value

The index of the first inserted item.

Related links

- Array

arr.move

Move an item.

```
method arr.move (
   uint from,
   uint to
)
```

Parameters

from	The current index of the item starting from zero.
to	The new index of the item starting from zero.

Related links

- Array

arr.sort

Sorting an array. Sort array items according to the sorting function. The function must have tw o parameters containing pointers to tw o compared items. It must return **int** less than, equal to or greater than zero if the left value is less than, equal to or greater than the first one respectively.

```
method arr arr.sort (
   uint sortfunc
)
```

Parameters

sortfunc	Sorting function.

Return value

Returns the object w hich method has been called.

Related links

- Array

Array Of Strings

Array of strings. You can use variables of the **arrstr** type for w orking w ith arrays of strings. The **arrstr** type is inherited from the **arr** type. So, you can also use methods of the arr type.

- Operators
- Methods
- Related Methods
- Type

Operators

arrstr = type	Convert types to the array of strings.
str = arrstr	Convert an array of strings to a multi-line string.
arrstr += type	Append types to an array of strings.

Methods

arrstr.insert	Insert a string to an array of strings.
arrstr.load	Add lines to the array from multi-line string.
arrstr.read	Read a multi-line text file to array of strings.
arrstr.replace	Replace substrings for the each item.
arrstr.setmultistr	Create a multi-string buffer.
arrstr.sort	Sort strings in the array.
arrstr.unite...	Unite strings of the array.
arrstr.write	Write an array of strings to a multi-line text file.

Related Methods

buf.getmultistr	Convert a buffer to array of strings.
str.lines	Convert a multi-line string to an array of strings.
str.split	Splitting a string.

Type

arrstr	The main structure of array of strings.

arrstr = type

- operator arrstr =(arrstr dest, str src)
- operator arrstr =(arrstr dest, arrstr src)
- operator arrstr =(arrstr left, collection right)

Convert types to the array of strings. Convert a multi-line string to an array of strings.

```
operator arrstr = (
   arrstr dest,
   str src
)
```

Return value

The array of strings.

arrstr = arrstr

Copy one array of strings to another array of strings.

```
operator arrstr = (
   arrstr dest,
   arrstr src
)
```

arrstr = collection

Copy a collection of strings to the array of strings.

```
operator arrstr = (
   arrstr left,
   collection right
)
```

Related links

- Array Of Strings

str = arrstr

Convert an array of strings to a multi-line string.

```
operator str = (
   str dest,
   arrstr src
)
```

Return value

The result string.

Related links

- Array Of Strings

arrstr += type

- operator arrstr +=(arrstr dest, str new str)
- operator arrstr +=(arrstr dest, arrstr src)

Append types to an array of strings. The operator appends a string at the end of the array of strings.

```
operator arrstr += (
   arrstr dest,
   str newstr
)
```

Return value

Returns the object w hich method has been called.

---

arrstr += arrstr

The operator appends one array of strings to another array of strings.

```
operator arrstr += (
   arrstr dest,
   arrstr src
)
```

Related links

- Array Of Strings

arrstr.insert

Insert a string to an array of strings.

```
method arrstr arrstr.insert (
   uint index,
   str newstr
)
```

Parameters

index	The index of the item w here the string w ill be inserted.
newstr	The inserting string.

Return value

Returns the object w hich method has been called.

Related links

- Array Of Strings

arrstr.load

- method arrstr arrstr.load(str input, uint flag)
- method arrstr arrstr.loadtrim(str input)

Add lines to the array from multi-line string.

```
method arrstr arrstr.load (
   str input,
   uint flag
)
```

Parameters

input	The input string.
flag	Flags.

$ASTR_APPEND	Adding strings. Otherw ise, the array is cleared before loading.
$ASTR_TRIM	Delete characters <= space on the left and on the right.

Return value

Returns the object w hich method has been called..

arrstr.loadtrim

Add lines to the array from multi-line string w ith trimming.

```
method arrstr arrstr.loadtrim (
   str input
)
```

Parameters

input	The input string.

Related links

- Array Of Strings

arrstr.read

Read a multi-line text file to array of strings.

```
method uint arrstr.read (
   str filename
)
```

Parameters

filename	The filename.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Array Of Strings

arrstr.replace

Replace substrings for the each item. The method looks for strings from one array and replace to strings of another array for the each string of the array.

```
method arrstr arrstr.replace (
   arrstr aold,
   arrstr anew,
   uint flags
)
```

Parameters

aold	The strings to be replaced.
anew	The new strings.
flags	Flags.

$QS_IGNCASE	Case-insensitive search.
$QS_WORD	Search the w hole w ord only.
$QS_BEGINWORD	Search w ords w hich start w ith the specified pattern.

Return value

Returns the object w hich method has been called.

Related links

- Array Of Strings

arrstr.setmultistr

- method buf arrstr.setmultistr(buf dest)
- method buf arrstr.setmultistr <result>

Create a multi-string buffer. The method w rites strings to a multi-string buffer w here strings are divided by zero character.

```
method buf arrstr.setmultistr (
   buf dest
)
```

Parameters

dest	The result buffer.

Return value

The result buffer.

The method creates a multi-string buffer w here strings are divided by zero character.

```
method buf arrstr.setmultistr <result>
```

Return value

The new result buffer.

Related links

- Array Of Strings

arrstr.sort

Sort strings in the array.

```
method arrstr.sort (
   uint mode
)
```

Parameters

mode	Specify 1 to sort w ith ignore-case sensitive. In default, specify 0.

Related links

- Array Of Strings

arrstr.unite...

- method str arrstr.unite(str dest, str separ)
- method str arrstr.unite(str dest)
- method str arrstr.unitelines(str dest)

Unite strings of the array. The method unites all items of the array to a string w ith the specified separator string.

```
method str arrstr.unite (
   str dest,
   str separ
)
```

Parameters

dest	The result string.
separ	A separator of the strings.

Return value

The result string.

---

arrstr.unite

The method unites all items of the array to a string.

```
method str arrstr.unite (
   str dest
)
```

Parameters

dest	The result string.

---

arrstr.unitelines

The method unites items of the array to a multi-line string. It inserts new -line characters betw een the each string of the array.

```
method str arrstr.unitelines (
   str dest
)
```

Parameters

dest	The result string.

Related links

- Array Of Strings

arrstr.write

Write an array of strings to a multi-line text file.

```
method uint arrstr.write (
   str filename
)
```

Parameters

filename	The filename.

Return value

The size of w ritten data.

Related links

- Array Of Strings

arrstr

The main structure of array of strings.

```
type arrstr <inherit=arr index=str>
{
}
```

Related links

- Array Of Strings

Array Of Unicode Strings

Array of unicode strings. You can use variables of the **arrustr** type for w orking w ith arrays of unicode strings. The **arrustr** type is inherited from the **arr** type. So, you can also use methods of the arr type.

- Operators
- Methods
- Related Methods
- Type

Operators

arrustr = type	Convert types to the array of unicode strings.
ustr = arrustr	Convert an array of unicode strings to a multi-line unicode string.
arrustr += type	Append types to an array of unicode strings.

Methods

arrustr.insert	Insert a unicode string to an array of unicode strings.
arrustr.load	Add lines to the array of unicode strings from multi-line unicode string.
arrustr.read	Read a multi-line text file to array of unicode strings.
arrustr.setmultiustr	Create a multi-string buffer.
arrustr.sort	Sort unicode strings in the array.
arrustr.unite...	Unite unicode strings of the array.
arrustr.write	Write an array of unicode strings to a multi-line text file.

Related Methods

buf.getmultiustr	Convert a buffer to array of unicode strings.
ustr.lines	Convert a multi-line unicode string to an array of unicode strings.
ustr.split	Splitting a unicode string.

Type

arrustr	The main structure of array of unicode strings.

arrustr = type

- operator arrustr =(arrustr dest, ustr src)
- operator arrustr =(arrustr dest, arrustr src)
- operator arrustr =(arrustr left, collection right)

Convert types to the array of unicode strings. Convert a multi-line unicode string to an array of unicode strings.

```
operator arrustr = (
   arrustr dest,
   ustr src
)
```

Return value

The array of unicode strings.

---

arrustr = arrustr

Copy one array of unicode strings to another array of unicode strings.

```
operator arrustr = (
   arrustr dest,
   arrustr src
)
```

---

arrustr = collection

Copy a collection of strings (simple or unicode) to the array of unicode strings.

```
operator arrustr = (
   arrustr left,
   collection right
)
```

Related links

- Array Of Unicode Strings

ustr = arrustr

Convert an array of unicode strings to a multi-line unicode string.

```
operator ustr = (
   ustr dest,
   arrustr src
)
```

Return value

The result string.

Related links

- Array Of Unicode Strings

arrustr += type

- operator arrustr +=(arrustr dest, ustr new str)
- operator arrustr +=(arrustr dest, arrustr src)

Append types to an array of unicode strings. The operator appends a unicode string at the end of the array of unicode strings.

```
operator arrustr += (
   arrustr dest,
   ustr newstr
)
```

Return value

Returns the object w hich method has been called.

---

arrustr += arrustr

The operator appends one array of unicode strings to another array of unicode strings.

```
operator arrustr += (
   arrustr dest,
   arrustr src
)
```

Related links

- Array Of Unicode Strings

arrustr.insert

Insert a unicode string to an array of unicode strings.

```
method arrustr arrustr.insert (
   uint index,
   ustr newstr
)
```

Parameters

index	The index of the item w here the string w ill be inserted.
newstr	The inserting unicode string.

Return value

Returns the object w hich method has been called.

Related links

- Array Of Unicode Strings

arrustr.load

- method arrustr arrustr.load(ustr input, uint flag)
- method arrustr arrustr.loadtrim(ustr input)

Add lines to the array of unicode strings from multi-line unicode string.

```
method arrustr arrustr.load (
   ustr input,
   uint flag
)
```

Parameters

input	The input unicode string.
flag	Flags.

$ASTR_APPEND	Adding strings. Otherw ise, the array is cleared before loading.
$ASTR_TRIM	Delete characters <= space on the left and on the right.

Return value

Returns the object w hich method has been called..

arrustr.loadtrim

Add lines to the array of unicode strings from multi-line unicode string w ith trimming.

```
method arrustr arrustr.loadtrim (
   ustr input
)
```

Parameters

input	The input unicode string.

Related links

- Array Of Unicode Strings

arrustr.read

Read a multi-line text file to array of unicode strings.

```
method uint arrustr.read (
   str filename
)
```

Parameters

filename	The filename.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Array Of Unicode Strings

arrustr.setmultiustr

Create a multi-string buffer. The method w rites unicode strings to a buffer.

```
method buf arrustr.setmultiustr (
   buf dest
)
```

Parameters

dest	The result buffer.

Return value

The result buffer.

Related links

- Array Of Unicode Strings

arrustr.sort

Sort unicode strings in the array.

```
method arrustr.sort (
   uint mode
)
```

Parameters

mode	Specify 1 to sort w ith ignore-case sensitive. In default, specify 0.

Related links

- Array Of Unicode Strings

arrustr.unite...

- method ustr arrustr.unite(ustr dest, ustr separ)
- method ustr arrustr.unitelines(ustr dest)

Unite unicode strings of the array. The method unites all items of the array to a unicode string w ith the specified separator string.

```
method ustr arrustr.unite (
   ustr dest,
   ustr separ
)
```

Parameters

dest	The result unicode string.
separ	A separator of the strings.

Return value

The result unicode string.

---

arrustr.unitelines

The method unites items of the array to a multi-line unicode string. It inserts new -line characters betw een the each string of the array.

```
method ustr arrustr.unitelines (
   ustr dest
)
```

Parameters

dest	The result unicode string.

Related links

- Array Of Unicode Strings

arrustr.write

Write an array of unicode strings to a multi-line text file.

```
method uint arrustr.write (
   str filename
)
```

Parameters

filename	The filename.

Return value

The size of w ritten data.

Related links

- Array Of Unicode Strings

arrustr

The main structure of array of unicode strings.

```
type arrustr <inherit=arr index=ustr>
{
}
```

Related links

- Array Of Unicode Strings

Buffer

Binary data. It is possible to use variables of the **buf** type for w orking w ith memory. Use this type if you w ant to store and manage the binary data.

- Operators
- Methods

Operators

*** buf**	Get the size of the memory being used.
buf[i]	Getting byte **<i>** from the buffer.
buf = buf	Copying data from one buffer into another.
buf + buf	Putting tw o buffers together and creating a resulting buffer.
buf += type	Appending types to the buffer.
buf == buf	Comparison operation.
buf(type)	Converting types to buf.

Methods

buf.align	Data alignment.
buf.append	Data addition.
buf.clear	Clear data in the object.
buf.copy	Copying.
buf.crc	Calculating the checksum.
buf.del	Data deletion.
buf.expand	Expansion.
buf.free	Memory deallocation.
buf.findch	Find a byte in a binary data.
buf.getmultistr	Convert a buffer to array of strings.
buf.getmultiustr	Convert a buffer to array of unicode strings.
buf.insert	Data insertion.
buf.ptr	Get the pointer to memory.
buf.read	Reading from a file.
buf.replace	Replacing data.
buf.reserve	Memory reservation.
buf.write	Writing to a file.
buf.writeappend	Appending data to a file.

* buf

Get the size of the memory being used.

```
operator uint * (
   buf left
)
```

Return value

The size of the used memory.

Related links

- Buffer

buf[i]

Getting byte **<i>** from the buffer.

```
method uint buf.index (
   uint i
)
```

Return value

The value of byte i of the memory data.

Related links

- Buffer

buf = buf

Copying data from one buffer into another.

```
operator buf = (
   buf left,
   buf right
)
```

Return value

The result buffer.

Related links

- Buffer

buf + buf

Putting tw o buffers together and creating a resulting buffer.

```
operator buf +<result> (
   buf left,
   buf right
)
```

Return value

The new result buffer.

Related links

- Buffer

buf += type

- operator buf +=(buf left, buf right)
- operator buf +=(buf left, ubyte right)
- operator buf +=(buf left, uint right)
- operator buf +=(buf left, ushort right)
- operator buf +=(buf left, ulong right)

Appending types to the buffer. Append **buf** to **buf** => **buf += buf**.

```
operator buf += (
   buf left,
   buf right
)
```

Return value

The result buffer.

buf += ubyte

Append **ubyte** to **buf** => **buf += ubyte**.

```
operator buf += (
   buf left,
   ubyte right
)
```

buf += uint

Append **uint** to **buf** => **buf += uint**.

```
operator buf += (
   buf left,
   uint right
)
```

buf += ushort

Append **ushort** to **buf** => **buf += ushort**.

```
operator buf += (
   buf left,
   ushort right
)
```

buf += ulong

Append **ulong** to **buf** => **buf += ulong**.

```
operator buf += (
   buf left,
   ulong right
)
```

Related links

- Buffer

buf == buf

- operator uint ==(buf left, buf right)
- operator uint !=(buf left, buf right)

Comparison operation.

```
operator uint == (
   buf left,
   buf right
)
```

Return value

Returns **1** if the buffers are equal. Otherw ise, it returns **0**.

---

buf != buf

Comparison operation.

```
operator uint != (
   buf left,
   buf right
)
```

Return value

Returns **0** if the buffers are equal. Otherw ise, it returns **1**.

Related links

- Buffer

buf(type)

Converting types to buf. Convert **uint** to **buf** => **buf(uint)**.

```
method buf uint.buf<result>
```

Return value

The result buffer.

Related links

- Buffer

buf.align

Data alignment. The method aligns the binary data and appends zeros if it is required.

```
method buf buf.align
```

Return value

Returns the object w hich method has been called.

Related links

- Buffer

buf.append

Data addition. The method adds data to the object.

```
method buf buf.append (
   uint ptr,
   uint size
)
```

Parameters

ptr	The pointer to the data to be added.
size	The size of the data being added.

Return value

Returns the object w hich method has been called.

Related links

- Buffer

buf.clear

Clear data in the object. This method sets the size of the binary data to zero.

```
method buf buf.clear()
```

Return value

Returns the object w hich method has been called.

Related links

- Buffer

buf.copy

Copying. The method copies a binary data into the object.

```
method buf buf.copy (
   uint ptr,
   uint size
)
```

Parameters

ptr	The pointer to the data being copied.
size	The size of the data being copied.

Return value

Returns the object w hich method has been called.

Related links

- Buffer

buf.crc

Calculating the checksum. The method calculates the checksum of data for an object of the **buf**.

```
method uint buf.crc
```

Return value

The checksum is returned.

Related links

- Buffer

buf.del

Data deletion. The method deletes part of the buffer.

```
method buf buf.del (
   uint offset,
   uint size
)
```

Parameters

offset	The offset of the data being deleted.
size	The size of the data being deleted.

Return value

Returns the object w hich method has been called.

Related links

- Buffer

buf.expand

Expansion. The method increases the size of memory allocated for the object.

```
method buf buf.expand (
   uint size
)
```

Parameters

size	The requested additional size of memory. It is an additional size to be reserved in the buffer.

Return value

Returns the object w hich method has been called.

Related links

- Buffer

buf.free

Memory deallocation. The method deallocates memory allocated for the object and destroys all data.

```
method buf buf.free()
```

Return value

Returns the object w hich method has been called.

Related links

- Buffer

buf.findch

Find a byte in a binary data.

```
method uint buf.findch (
   uint offset,
   uint ch
)
```

Parameters

offset	The offset to start searching from.
ch	A unsigned byte to be searched.

Return value

The offset of the byte if it is found. If the byte is not found, the size of the buffer is returned.

Related links

- Buffer

buf.getmultistr

- method arrstr buf.getmultistr(arrstr ret, arr offset)
- method arrstr buf.getmultistr(arrstr ret)

Convert a buffer to array of strings. Load the array of string from multi-string buffer w here strings are divided by zero character.

```
method arrstr buf.getmultistr (
   arrstr ret,
   arr offset
)
```

Parameters

ret	The result array of strings.
offset	The array for getting offsets of strings in the buffer. It can be 0->>arr.

Return value

The result array of strings.

---

buf.getmultistr

Load the array of string from multi-string buffer w here strings are divided by zero character.

```
method arrstr buf.getmultistr (
   arrstr ret
)
```

Parameters

ret	The result array of strings.

Related links

- Buffer

buf.getmultiustr

Convert a buffer to array of unicode strings. Load the array of string from multi-string buffer w here strings are divided by zero character.

```
method arrustr buf.getmultiustr (
   arrustr ret
)
```

Parameters

ret	The result array of unicode strings.

Return value

The result array of unicode strings.

Related links

- Buffer

buf.insert

- method buf buf.insert(uint offset, buf value)
- method buf buf.insert(uint offset, uint ptr, uint size)

Data insertion. The method inserts one buf object into another.

```
method buf buf.insert (
   uint offset,
   buf value
)
```

Parameters

offset	The offset w here data w ill be inserted. If the offset is greater than the size, data is added to the end to the buffer.
value	The **buf** object w ith the data to be inserted.

Return value

Returns the object w hich method has been called.

buf.insert

The method inserts one memory data into the buffer.

```
method buf buf.insert (
   uint offset,
   uint ptr,
   uint size
)
```

Parameters

offset	The offset w here data w ill be inserted. If the offset is greater than the size, data is added to the end to the buffer.
ptr	The pointer to the memory data to be inserted.
size	The size of the data to be inserted.

Related links

- Buffer

buf.ptr

Get the pointer to memory.

```
method buf buf.ptr()
```

Return value

The pointer to the allocated memory of the binary data.

Related links

- Buffer

buf.read

Reading from a file. The method reads data from the file.

```
method uint buf.read (
   str filename
)
```

Parameters

filename	Filename.

Return value

The size of the read data.

Related links

- Buffer

buf.replace

Replacing data. The method replaces binary data in an object.

```
method buf buf.replace (
   uint offset,
   uint size,
   buf value
)
```

Parameters

offset	The offset of the data being replaced.
size	The size of the data being replaced.
value	The **buf** object w ith new data.

Return value

Returns the object w hich method has been called.

Related links

- Buffer

buf.reserve

Memory reservation. The method increases the size of memory allocated for the object.

```
method buf buf.reserve (
   uint size
)
```

Parameters

size	The summary requested size of memory. If it is less than the current size, nothing happens. If the size is increased, the current data is saved.

Return value

Returns the object w hich method has been called.

Related links

- Buffer

buf.write

Writing to a file. The method w rites data to the file.

```
method uint buf.write (
   str filename
)
```

Parameters

filename	Filename.

Return value

The size of the w ritten data.

Related links

- Buffer

buf.writeappend

Appending data to a file. The method appends data to the specified file.

```
method uint buf.writeappend (
   str filename
)
```

Parameters

filename	Filename.

Return value

The size of the w ritten data.

Related links

- Buffer

Clipboard

These functions are used to w ork w ith the Window s clipboard. For using this library, it is required to specify the file clipboard.g (from lib\clipboard subfolder) w ith include command.

```
include : $"...\gentee\lib\clipboard\clipboard.g"
```

- Methods

clipboard_gettext	Gets a string from the clipboard.
clipboard_empty	Clear the clipboard.
clipboard_settext	Copies a string into the clipboard.

Methods

buf.getclip	Copy the clipboard data to buf variable.
buf.setclip	Copy the data of the buf variable to the clipboard.
str.getclip	Copy the clipboard data to str variable if the clipboard contains text data.
str.setclip	Copy a string to the clipboard.
ustr.getclip	Copy the clipboard data to unicode str variable if the clipboard contains unicode text data.
ustr.setclip	Copy a unicode string to the clipboard.

clipboard_gettext

Gets a string from the clipboard.

```
{
```

Parameters

data	Result string.

Return value

Returns the parameter **data**.

Related links

- Clipboard

clipboard_empty

Clear the clipboard.

```
func uint clipboard_empty
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Clipboard

clipboard_settext

Copies a string into the clipboard.

```
func uint clipboard_settext (
   str data
)
```

Parameters

data	The string for copying into the clipboard.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Clipboard

buf.getclip

Copy the clipboard data to buf variable.

```
method uint buf.getclip (
   uint cftype
)
```

Parameters

cftype	The type of the clipboard data.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Clipboard

buf.setclip

Copy the data of the buf variable to the clipboard.

```
method uint buf.setclip (
   uint cftype locale
)
```

Parameters

cftype	The type of the buf data.
locale	Locale identifier. It can be 0.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Clipboard

str.getclip

Copy the clipboard data to str variable if the clipboard contains text data.

```
method uint str.getclip()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Clipboard

str.setclip

Copy a string to the clipboard.

```
method uint str.setclip()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Clipboard

ustr.getclip

Copy the clipboard data to unicode str variable if the clipboard contains unicode text data.

```
method uint ustr.getclip()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Clipboard

ustr.setclip

Copy a unicode string to the clipboard.

```
method uint ustr.setclip()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Clipboard

Collection

Collection. You can use variables of the **collection** type for w orking w ith collections. Collection is an object w hich can contains objects of different types. The **collection** type is inherited from the **buf** type. So, you can also use methods of the buf type.

- Operators
- Methods
- Types

Operators

*** collection**	Gets the amount of elements in the collection.
collection[i]	Gets a value of a collection element.
collection = collection	Collection copying.
collection += collection	Appends elements of a collection to another collection.
collection + collection	Putting tw o collections together and creating a resulting collection.
foreach var,collection	Foreach operator.

Methods

collection.append	Append an object or a numeric value to the collection.
collection.clear	Delete all items from the collection.
collection.gettype	Gets an element type of a collection.
collection.ptr	Gets a pointer to a collection element.

Types

colitem	The structure is used in **foreach** operator.

* collection

Gets the amount of elements in the collection.

```
operator uint * (
   collection left
)
```

Return value

The count of the collection items.

Related links

- Collection

collection[i]

Gets a value of a collection element. Don't use if the collection contains double, ulong or long types.

```
method uint collection.index (
   uint ind
)
```

Return value

A value of the collection element.

Related links

- Collection

collection = collection

Collection copying.

```
operator collection = (
   collection left,
   collection right
)
```

Related links

- Collection

collection += collection

Appends elements of a collection to another collection.

```
operator collection += (
   collection left,
   collection right
)
```

Related links

- Collection

collection + collection

Putting tw o collections together and creating a resulting collection.

```
operator collection +<result>  (
   collection left,
   collection right
)
```

Return value

The new result collection.

Related links

- Collection

foreach var,collection

Foreach operator. You can use **foreach** operator to look over items of the collection. The variable **var** has colitem type.

```
foreach variable,collection {...}
```

Related links

- Collection

collection.append

Append an object or a numeric value to the collection.

```
method uint collection.append (
   uint value,
   uint itype
)
```

Parameters

value	The value of the 32-bit number or the pointer to 64-bit number or the ponter to any other object.
itype	The type of the appending value.

Return value

An index of the appended item.

Related links

- Collection

collection.clear

Delete all items from the collection.

```
method collection collection.clear()
```

Return value

Returns the object w hich method has been called.

Related links

- Collection

collection.gettype

Gets an element type of a collection.

```
method uint collection.gettype (
   uint ind
)
```

Parameters

ind	Element index starts at zero.

Return value

An element type of a collection or zero on error.

Related links

- Collection

collection.ptr

Gets a pointer to a collection element.

```
method uint collection.ptr (
   uint ind
)
```

Parameters

ind	Element index starts at zero.

Return value

A pointer to a collection element, or zero on error.

Related links

- Collection

colitem

The structure is used in **foreach** operator. The variable of the foreach operator has this type.

```
type colitem
{
   uint oftype
   uint val
   uint hival
   uint ptr
}
```

Members

oftype	The type of the item.
val	The value of the item.
hival	The hi-uint of the value. It is used if the value is 64-bit.
ptr	The pointer to the value.

Related links

- Collection

COM/OLE

Working w ith COM/OLE Object. The COM library is applied for w orking w ith the **COM/OLE objects**, the **IDispatch** interface and maintains late binding operations. For using this library, it is required to specify the file olecom.g (from lib\olecom subfolder) w ith include command.

```
include : $"...\gentee\lib\olecom\olecom.g"
```

- Operators
- Methods
- VARIANT Methods

COM/OLE description	A brief description of COM/OLE library.
VARIANT	VARIANT type.

Operators

type = VARIANT	Assign operation.
VARIANT = type	Assign operation.
type(VARIANT)	Conversion.

Methods

oleobj.createobj	The method creates a new COM object.
oleobj.getres	Result of the last operation.
oleobj.iserr	Enables to define w hether or not an error occurs w hile w orking w ith a COM object.
oleobj.release	Releasing the COM object.

VARIANT Methods

variant.arrcreate	Creating the SafeArray array.
variant.arrfromg	Assigning a value to an element of the SafeArray array.
variant.arrgetptr	Obtaining a pointer to an element of the SafeArray array.
variant.clear	Clears the variable contents, the storage area is released if necessary.
variant.ismissing	Checks if the variant is "missing" (optional) parameter of the method.
variant.isnull	Enables to define w hether or not a variable is NULL.
variant.setmissing	Sets the "missing" variant.

COM/OLE description

A brief description of COM/OLE library. This library also contains the support of the VARIANT type, used for data transmitting from/to COM objects. Variables of the **oleobj** type are used for w orking w ith the COM objects; furthermore, each variable of this type has one appropriate COM object. A COM objects method is called w ith the help of the ~ late binding operation. There are tw o w ays of binding a COM object w ith a variable , as follow s:

1. The oleobj.createobj method is used for creating a new COM object:

```
oleobj excapp
excapp.createobj( "Excel.Application", "" )
```

2. Binding a variable w ith the existing COM object (child) is returned by another COM object method call:

```
oleobj workbooks
workbooks = excapp~WorkBooks
```

The **oleobj** object can maintain the follow ing kinds of late binding:

- elementary method call **excapp~Quit**, w ith/w ithout parameters;
- set value **excapp~Cells(3, 2) = "Hello World!"**;
- get value **vis = uint(excapp~Visible)**;
- call chain **excapp~WorkBooks~Add**, equals the follow ing expressions

```
oleobj workbooks
workbooks = excapp~WorkBooks
workbooks~Add
```

The method call can return only the **VARIANT** type, and the appropriate assignment operators and type cast operators are used to convert data to basic Gentee types. Parameters of the COM objects methods call as w ell as the assigned values are automatically converted to the appropriate VARIANT types. The follow ing Gentee types can be used - **uint, int, ulong, long, float, double, str, VARIANT**.

Use the oleobj.release method in order to release the COM object; otherw ise, the COM object is released w hen the variable is deleted; also the object is released w hen the variable is bound w ith another COM object. Have a look at the example of using the COM object

```
include : $"...\olecom.g"
func ole_example
{
   oleobj excapp
   excapp.createobj( "Excel.Application", "" )
   excapp.flgs = $FOLEOBJ_INT
   excapp~Visible = 1
   excapp~WorkBooks~Add
   excapp~Cells( 3, 2 ) = "Hello World!"
 }
```

The oleobj object has properties, as follow s:

- uint **flgs** are flags. Flags value can be set or obtained; the property can contain the **$FOLEOBJ_INT** flag, i.e. w hen transmitting data to the COM object the unsigned Gentee type of uint is automatically converted to the signed type of VARIANT(VT_I4)
- uint **errfunc** is an error handling function. A function address can be assigned to this property, so using the COM object this function w ill be called as long as an error occurs; furthermore, this function must have a parameter of the uint type, that contains an error code.

All child objects automatically inherit the **flgs** property as w ell as the **errfunc** property.

Related links

- COM/OLE

VARIANT

VARIANT type. **VARIANT** is a universal type that is used for storing various data and it enables different programs to exchange data properly. This type represents a structure consisted of tw o main fields: the first field is a type of the stored value, the second field is the stored value or the pointer to a storage area. The **VARIANT** type is defined as follow s:

```
type VARIANT {
   ushort vt
   ushort wReserved1
   ushort wReserved2
   ushort wReserved3
   ulong  val
 }
```

vt is a type code of the contained value (type constants VT_*: $VT_UI4, $VT_I4, $VT_BSTR ...);
val is a field used for storing values

The library provides only some of the operations of the VARIANT type, how ever, you can use the fields of the given structure. The example illustrates creation of the VARIANT(VT_BOOL) variable:

```
VARIANT bool
....
bool.clear()
bool.vt = $VT_BOOL
(&bool.val)->uint = 0xffff// 0xffff - VARIANT_TRUE
```

This example show s VARIANT operations

```
uint val
str  res
oleobj ActWorkSheet
VARIANT vval

....
vval = int( 100 )         //VARIANT( VT_I4 ) is being created
excapp~Cells(1,1) = vval //equals excapp~Cells(1,1) = 100

vval = "Test string"      //VARIANT( VT_BSTR ) is being created
excapp~Cells(2,1) = vval //equals excapp~Cells(1,1) = "Test string"

val = uint( excapp~Cells(1,1)~Value ) //VARIANT( VT_I4 ) is converted to uint
res = excapp~Cells(2,1)~Value         //VARIANT( VT_BSTR ) is converted to str
ActWorkSheet = excapp~ActiveWorkSheet //VARIANT( VT_DISPATCH ) is converted
 to oleobj
```

Related links

- COM/OLE

type = VARIANT

- operator str = (str left, VARIANT right)
- operator oleobj = (oleobj left, VARIANT right)

Assign operation. **str = VARIANT(VT_BSTR)**.

```
operator str =  (
   str left,
   VARIANT right
)
```

Return value

The result string.

---

oleobj = VARIANT

Assign operation. **oleobj = VARIANT(VT_DISPATCH)**.

```
operator oleobj =  (
   oleobj left,
   VARIANT right
)
```

Return value

The result **oleobj**.

Related links

- COM/OLE

VARIANT = type

- operator VARIANT = (VARIANT left, uint right)
- operator VARIANT = (VARIANT left, int right)
- operator VARIANT = (VARIANT left, float right)
- operator VARIANT = (VARIANT left, double right)
- operator VARIANT = (VARIANT left, long right)
- operator VARIANT = (VARIANT left, ulong right)
- operator VARIANT = (VARIANT left, str right)
- operator VARIANT = (VARIANT left, VARIANT right)

Assign operation. **VARIANT = uint**.

```
operator VARIANT =  (
   VARIANT left,
   uint right
)
```

Return value

VARIANT(VT_UI4).

VARIANT = int

Assign operation: **VARIANT = int**.

```
operator VARIANT =  (
   VARIANT left,
   int right
)
```

Return value

VARIANT(VT_I4).

VARIANT = float

Assign operation: **VARIANT = float**.

```
operator VARIANT =  (
   VARIANT left,
   float right
)
```

Return value

VARIANT(VT_R4).

VARIANT = double

Assign operation: **VARIANT = double**.

```
operator VARIANT =  (
   VARIANT left,
   double right
)
```

Return value

VARIANT(VT_R8).

VARIANT = long

Assign operation: **VARIANT = long**.

```
operator VARIANT =  (
   VARIANT left,
   long right
)
```

Return value

VARIANT(VT_I8).

VARIANT = ulong

Assign operation: **VARIANT = ulong**.

```
operator VARIANT =  (
   VARIANT left,
   ulong right
)
```

Return value

VARIANT(VT_UI8).

VARIANT = str

Assign operation: **VARIANT = str**.

```
operator VARIANT =   (
   VARIANT left,
   str right
)
```

Return value

VARIANT(VT_BSTR).

VARIANT = VARIANT

Assign operation: **VARIANT = VARIANT**.

```
operator VARIANT =   (
   VARIANT left,
   VARIANT right
)
```

Return value

VARIANT.

Related links

- COM/OLE

type(VARIANT)

- method str VARIANT.str <result>
- method ulong VARIANT.ulong
- method long VARIANT.long
- method uint VARIANT.uint
- method int VARIANT.int
- method float VARIANT.float
- method double VARIANT.double

Conversion. **str(VARIANT)**.

```
method str VARIANT.str <result>
```

Return value

The result **str** value.

VARIANT.ulong

Conversion: **ulong(VARIANT)**.

```
method ulong VARIANT.ulong
```

Return value

The result **ulong** value.

VARIANT.long

Conversion: **long(VARIANT)**.

```
method long VARIANT.long
```

Return value

The result **long** value.

VARIANT.uint

Conversion: **uint(VARIANT)**.

```
method uint VARIANT.uint
```

Return value

The result **uint** value.

VARIANT.int

Conversion: **int(VARIANT)**.

```
method int VARIANT.int
```

Return value

The result **int** value.

VARIANT.float

Conversion: **float(VARIANT)**.

```
method float VARIANT.float
```

Return value

The result **float** value.

VARIANT.double

Conversion: **double(VARIANT)**.

```
method double VARIANT.double
```

Return value

The result **double** value.

Related links

- COM/OLE

oleobj.createobj

The method creates a new COM object. Example:

```
oleobj excapp
excapp.createobj( "Excel.Application", "" )
//is equal to excapp.createobj( "{00024500-0000-0000-C000-000000000046}", "" )
|
excapp.flgs = $FOLEOBJ_INT
excapp~Visible = 1
```

```
method uint oleobj.createobj (
   str name,
   str mashine
)
```

Parameters

name	An object name, or the string representation of an object identifier - "{...}".
mashine	A computer name w here the required object is created; if the current string is empty, the object is created in the current computer.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- COM/OLE

oleobj.getres

Result of the last operation. This method is applied for getting an error code or a w arning; the code is the C type of HRESULT.

```
method uint oleobj.getres()
```

Return value

Returns the HRESULT code of the last COM object operation.

Related links

- COM/OLE

oleobj.iserr

Enables to define w hether or not an error occurs w hile w orking w ith a COM object.

```
method uint oleobj.iserr()
```

Return value

Returns the HRESULT code of the last COM object operation.

Related links

- COM/OLE

oleobj.release

Releasing the COM object. The method deletes the bond betw een the variable and the COM object and releases the COM object.

```
method oleobj.release()
```

Related links

- COM/OLE

variant.arrcreate

Creating the SafeArray array. This method creates the **Safe Array** array in the variable of the VARIANT type. VARIANT is an element of the array. Values can be assigned to the array elements using the variant.arrfromg method. An element of the array can be obtained w ith the help of the variant.arrgetptr method.

The example uses SafeArray

```
VARIANT v
//An array with 3 lines and 2 columns is being created
v.arrcreate( %{3,0,2,0} )

v.arrfromg( %{0,0, 0.1234f} )
v.arrfromg( %{0,1, int(100)} )
v.arrfromg( %{2,1, "Test" } )
...
//The array is being transmitted to the COM object
excapp~Range( excapp~Cells( 1, 1 ), excapp~Cells( 3, 2 ) ) = v
```

SafeArray allow s you to group data, that makes data exchange w ith the COM object faster.

```
method uint VARIANT.arrcreate (
   collection bounds
)
```

Parameters

bounds	The collection that contains array parameters. Tw o numbers are specified for each array dimension: the first number - an element quantity, the second number - a sequence number of the first element in the dimension.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- COM/OLE

variant.arrfromg

Assigning a value to an element of the SafeArray array. Example

```
v.arrfromg( %{0,0, 0.1234f} )
v.arrfromg( %{0,1, int(100)} )
v.arrfromg( %{2,1, "Test" } )
```

```
method uint VARIANT.arrfromg (
   collection item
)
```

Parameters

item	The collection that contains "coordinates" of an element; the last element of the collection - the assigned value.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- COM/OLE

variant.arrgetptr

Obtaining a pointer to an element of the SafeArray array.

```
method uint VARIANT.arrgetptr (
   collection item
)
```

Parameters

item	The collection that contains "coordinates" of an element.

Return value

The method returns address of an array element, if error occurs it returns zero.

Related links

- COM/OLE

variant.clear

Clears the variable contents, the storage area is released if necessary. The VARIANT type is equal to VT_EMPTY. This method is automatically called before a new value has been set .

```
method VARIANT.clear()
```

Related links

- COM/OLE

variant.ismissing

Checks if the variant is "missing" (optional) parameter of the method.

```
method uint VARIANT.ismissing()
```

Return value

The method returns 1, if the VARIANT variable is "missing".

Related links

- COM/OLE

variant.isnull

Enables to define w hether or not a variable is NULL. This method enables you to define w hether or not a variable is NULL - the VARIANT(VT_NULL) type.

```
method uint VARIANT.isnull()
```

Return value

The method returns 1, if the VARIANT variable is of the VT_NULL type, otherw ise, it returns zero.

Related links

- COM/OLE

variant.setmissing

Sets the "missing" variant. The method sets the variant variable as "missing" (optional) parameter.

```
method VARIANT.setmissing()
```

Related links

- COM/OLE

Console

Console library. Functions for w orking w ith the console.

congetch	Displaying text and w aiting for a keystroke.
congetstr	Getting a string after text is displayed.
conread	Get a string entered by the user.
conrequest	Displaying a multiple choice request on the console.
conyesno	Displaying a question on the console.

congetch

Displaying text and w aiting for a keystroke.

```
func uint congetch (
   str output
)
```

Parameters

output	Message text.

Return value

The function returns the value of the pressed key.

Related links

- Console

congetstr

Getting a string after text is displayed. Get the string entered by the user w ith some text displayed before that.

```
func str congetstr (
   str output,
   str input
)
```

Parameters

output	Text for displaying.
input	The variable of the str type for getting data.

Return value

Returns the parameter **input**.

Related links

- Console

conread

Get a string entered by the user.

```
func str conread (
   str input
)
```

Parameters

input	The variable of the str type for getting data.

Return value

Returns the parameter **input**.

Related links

- Console

conrequest

Displaying a multiple choice request on the console.

```
func uint conrequest (
   str output,
   str answer
)
```

Parameters

output	Request text.
answer	Enumerating possible answ er letters. Answ er variants are separated by '|'. For example, "Nn|Yy"

Return value

The function returns the number of the selected variant beginning from 0.

Related links

- Console

conyesno

Displaying a question on the console.

```
func uint conyesno (
   str output
)
```

Parameters

output	Question text.

Return value

The function returns 1 if the answ er is 'yes' and 0 otherw ise.

Related links

- Console

CSV

Working w ith CSV data. Variables of the **csv** type allow you to w ork w ith data in the csv format.

string1_1,"string1_2",string1_3
string2_1,"string2_2",string2_3

The **csv** type is inherited from **str** type. So, you can use string methods and operators. For using this library, it is required to specify the file csv.g (from lib\csv subfolder) w ith include command.

```
include : $"...\gentee\lib\csv\csv.g"
```

- Operators
- Methods

Operators

foreach var,csv	Foreach operator.

Methods

csv.append	Adds a string to a csv object.
csv.clear	Clear the csv data object.
csv.read	Read data from a csv file.
csv.settings	Set separating and limiting characters for csv data.
csv.write	Writing csv data to a file.

foreach var,csv

Foreach operator. Looking through all items w ith the help of the **foreach** operator. An element in an object of the **csv** type is an array of strings **arrstr**. Each string is split into separate elements by the separator and these elements are w ritten into the passed array.

```
csv mycsv
uint i k
...
foreach item, mycsv
{
   print( "Item: \(++i)\n" )
   fornum k = 0, *item
   {
      print( "\(item[k])\n")
   }
}
```

```
foreach variable,csv {...}
```

Related links

- CSV

csv.append

Adds a string to a csv object.

```
method csv.append (
   arrstr arrs
)
```

Parameters

arrs	The array of strings containing the elements of a string. All strings w ill be combined into one record and added to the **csv** object.

Related links

- CSV

csv.clear

Clear the csv data object.

```
method uint csv.clear()
```

Related links

- CSV

csv.read

Read data from a csv file.

```
method uint csv.read (
   str filename
)
```

Parameters

filename	Filename.

Return value

The size of the read data.

Related links

- CSV

csv.settings

Set separating and limiting characters for csv data.

```
method csv.settings (
   uint separ,
   uint open,
   uint close
)
```

Parameters

separ	Separator. Comma by default.
open	The left limiting character. Double quotes by default.
close	The right limiting character. Double quotes by default.

Related links

- CSV

csv.write

Writing csv data to a file.

```
method uint csv.write (
   str filename
)
```

Parameters

filename	The name of the file for w riting. If the file already exists, it w ill be overw ritten.

Return value

The size of the w ritten data.

Related links

- CSV

Date & Time

Functions for w orking w ith date and time.

Operators

datetime = datetime	Copying datatime structure.
datetime += uint	Adding days to a date.
datetime -= uint	Subtracting days from a date.
datetime - datetime	Difference betw een tw o dates as days and time.
datetime + datetime	Adding tw o dates as days and time.
datetime == datetime	Comparison operations.
datetime < datetime	Comparison operation.
datetime > datetime	Comparison operation.

Functions

abbrnameofday	Get the short name of a w eekday in the user's language.
days	The number of days betw een tw o dates.
daysinmonth	The number of days in a month.
firstdayofweek	Get the first day of a w eek for the user's locale.
getdateformat	Get date in the specified format.
getdatetime	Getting date and time as strings.
gettimeformat	Get time in the specified format.
isleapyear	Leap year check.
nameofmonth	Get the name of a month in the user's language.

Methods

datetime.dayofweek	Get the w eekday.
datetime.dayofyear	Get the number of a particular day in the year.
datetime.fromstr	Convert string like **SSMMHHDDMMYYYY** to datetime structure.
datetime.gettime	Getting the current date and time.
datetime.getsystime	Getting the current system date and time.
datetime.normalize	Normalizing a datetime structure.
datetime.setdate	Specifying a date.
datetime.tostr	Convert a datetime structure to string like **SSMMHHDDMMYYYY**.

File time functions and operators

filetime = filetime	Copying filetime structure.
filetime == filetime	Comparison operations.
filetime < filetime	Comparison operation.

filetime > filetime	Comparison operation.
datetimetoftime	Converting date from datetime into filetime.
ftimetodatetime	Converting date from filetime into datetime.
getfiledatetime	Getting date and time as strings.

Types

datetime	The datetime structure.
filetime	The filetime structure.

datetime = datetime

Copying datatime structure.

```
operator datetime = (
   datetime left,
   datetime right
)
```

Return value

The result datetime.

Related links

- Date & Time

datetime += uint

Adding days to a date.

```
operator datetime += (
   datetime left,
   uint next
)
```

Return value

The result datetime.

Related links

- Date & Time

datetime -= uint

Subtracting days from a date.

```
operator datetime -= (
   datetime left,
   uint next
)
```

Return value

The result datetime.

Related links

- Date & Time

datetime - datetime

- operator datetime -<result>(datetime left, datetime right)
- operator datetime -=(datetime left, datetime right)

Difference betw een tw o dates as days and time. All values are positive numbers.

```
operator datetime -<result> (
   datetime left,
   datetime right
)
```

Return value

The result datetime.

---

datetime -= datetime

Difference betw een tw o dates as days and time. All values are positive numbers.

```
operator datetime -= (
   datetime left,
   datetime right
)
```

Return value

The result datetime.

Related links

- Date & Time

datetime + datetime

- operator datetime +<result>(datetime left, datetime right)
- operator datetime +=(datetime left, datetime right)

Adding tw o dates as days and time. All values are positive numbers.

```
operator datetime +<result> (
   datetime left,
   datetime right
)
```

Return value

The result datetime.

---

datetime += datetime

Adding one datetime to another datetime structure.

```
operator datetime += (
   datetime left,
   datetime right
)
```

Return value

The result datetime.

Related links

- Date & Time

datetime == datetime

- operator uint ==(datetime left, datetime right)
- operator uint !=(datetime left, datetime right)

Comparison operations.

```
operator uint == (
   datetime left,
   datetime right
)
```

Return value

Returns **1** if the datetimes are equal. Otherw ise, it returns **0**.

---

datetime != datetime

Comparison operation.

```
operator uint != (
   datetime left,
   datetime right
)
```

Return value

Returns **0** if the datetimes are equal. Otherw ise, it returns **1**.

Related links

- Date & Time

datetime < datetime

- operator uint <(datetime left, datetime right)
- operator uint <=(datetime left, datetime right)

Comparison operation.

```
operator uint < (
   datetime left,
   datetime right
)
```

Return value

Returns **1** if the first datetime is less than the second one. Otherw ise, it returns **0**.

---

datetime <= datetime

Comparison operation.

```
operator uint <= (
   datetime left,
   datetime right
)
```

Return value

Returns **1** if the first datetime is less or equal the second one. Otherw ise, it returns **0**.

Related links

- Date & Time

datetime > datetime

- operator uint >(datetime left, datetime right)
- operator uint >=(datetime left, datetime right)

Comparison operation.

```
operator uint > (
   datetime left,
   datetime right
)
```

Return value

Returns **1** if the first datetime is greater than the second one. Otherw ise, it returns **0**.

---

datetime >= datetime

Comparison operation.

```
operator uint >= (
   datetime left,
   datetime right
)
```

Return value

Returns **1** if the first datetime is greater or equal the second one. Otherw ise, it returns **0**.

Related links

- Date & Time

abbrnameofday

Get the short name of a w eekday in the user's language.

```
func str abbrnameofday (
   str ret,
   uint dayofweek
)
```

Parameters

ret	The string for getting the result.
dayofweek	The number of the w eekday. 0 is Sunday, 1 is Monday...

Return value

Returns the parameter **ret**.

Related links

- Date & Time

days

The number of days betw een tw o dates.

```
func int days (
   datetime left,
   datetime right
)
```

Parameters

left	The first date for comparison.
right	The second date for comparison.

Return value

Returns the number of days betw een tw o dates. If the first date is greater than the second one, the return value w ill be negative.

Related links

- Date & Time

daysinmonth

The number of days in a month. Leap years are taken into account for February.

```
func uint daysinmonth (
   ushort year,
   ushort month
)
```

Parameters

year	Year.
month	Month.

Return value

Returns the number of days in the month.

Related links

- Date & Time

firstdayofweek

Get the first day of a w eek for the user's locale.

```
func uint firstdayofweek()
```

Return value

Returns the number of the w eekday. 0 is Sunday, 1 is Monday...

Related links

- Date & Time

getdateformat

Get date in the specified format.

```
func str getdateformat (
   datetime systime,
   str format,
   str date
)
```

Parameters

systime	The variable containing date.
format	Date format. It can contain the follow ing values:
date	The string for getting the date.

dd	Day as a number.
ddd	Weekday as an abbriviation.
dddd	The full name of a w eekday.
MM	Month as a number.
MMM	Month as an abbreviation.
MMMM	The full name of a month.
yy	The last tow digits in a year.
yyyy	Year.

Return value

Returns the parameter **date**.

Related links

- Date & Time

getdatetime

Getting date and time as strings. Get date and time in the current Window s string format.

```
func getdatetime (
   datetime systime,
   str date,
   str time
)
```

Parameters

systime	Datetime structure.
date	The string for getting the date. It can be 0->str.
time	The string for getting time. It can be 0->str.

Related links

- Date & Time

gettimeformat

Get time in the specified format.

```
func str gettimeformat (
   datetime systime,
   str format,
   str time
)
```

Parameters

systime	The variable containing time.
format	Time format. It can contain the follow ing values:
time	The string for getting time.

hh	Hours - 12-hour format.
HH	Hours -24-hour format.
mm	Minutes.
ss	Seconds.
tt	Time marker, such as AM or PM.

Return value

Returns the parameter **time**.

Related links

- Date & Time

isleapyear

Leap year check.

```
func uint isleapyear (
   ushort year
)
```

Parameters

year	The year being checked.

Return value

Returns 1 if the year is a leap one and 0 otherw ise.

Related links

- Date & Time

nameofmonth

Get the name of a month in the user's language.

```
func str nameofmonth (
   str ret,
   uint month
)
```

Parameters

ret	Result string.
month	The number of the month from 1.

Return value

Returns the parameter **ret**.

Related links

- Date & Time

datetime.dayofweek

Get the w eekday.

```
method uint datetime.dayofweek
```

Return value

Returns the w eekday. 0 is Sunday, 1 is Monday...

Related links

- Date & Time

datetime.dayofyear

Get the number of a particular day in the year.

```
method uint datetime.dayofyear
```

Return value

Returns the number of a particular day in the year.

Related links

- Date & Time

datetime.fromstr

Convert string like **SSMMHHDDMMYYYY** to datetime structure.

```
method datetime datetime.fromstr (
   str data
)
```

Parameters

data	The string to be converted.

Return value

Returns the object w hich method has been called.

Related links

- Date & Time

datetime.gettime

Getting the current date and time. The w eekday is set automatically.

```
method datetime datetime.gettime()
```

Return value

Returns the object w hich method has been called.

Related links

- Date & Time

datetime.getsystime

Getting the current system date and time.

```
method datetime datetime.getsystime()
```

Return value

Returns the object w hich method has been called.

Related links

- Date & Time

datetime.normalize

Normalizing a datetime structure. For example, if the hour parameter is 32 hours, it w ill equal 8 and the day parameter is increased by 1.

```
method datetime datetime.normalize()
```

Return value

Returns the object w hich method has been called.

Related links

- Date & Time

datetime.setdate

Specifying a date. The w eekday is set automatically.

```
method datetime datetime.setdate (
   uint day,
   uint month,
   uint year
)
```

Parameters

day	Day.
month	Month.
year	Year.

Return value

Returns the object w hich method has been called.

Related links

- Date & Time

datetime.tostr

Convert a datetime structure to string like **SSMMHHDDMMYYYY**.

```
method str datetime.tostr (
   str ret
)
```

Parameters

ret	The result string the datetime to be converted to.

Return value

Returns the parameter **`ret`**.

Related links

- Date & Time

filetime = filetime

Copying filetime structure.

```
operator filetime = (
   filetime left,
   filetime right
)
```

Return value

The result filetime.

Related links

- Date & Time

filetime == filetime

- operator uint ==(filetime left, filetime right)
- operator uint !=(filetime left, filetime right)

Comparison operations.

```
operator uint == (
   filetime left,
   filetime right
)
```

Return value

Returns **1** if the filetimes are equal. Otherw ise, it returns **0**.

---

filetime != filetime

Comparison operation.

```
operator uint != (
   filetime left,
   filetime right
)
```

Return value

Returns **0** if the filetimes are equal. Otherw ise, it returns **1**.

Related links

- Date & Time

filetime < filetime

- operator uint <(filetime left, filetime right)
- operator uint <=(filetime left, filetime right)

Comparison operation.

```
operator uint < (
   filetime left,
   filetime right
)
```

Return value

Returns **1** if the first filetime is less than the second one. Otherw ise, it returns **0**.

---

filetime <= filetime

Comparison operation.

```
operator uint <= (
   filetime left,
   filetime right
)
```

Return value

Returns **1** if the first filetime is less or equal the second one. Otherw ise, it returns **0**.

Related links

- Date & Time

filetime > filetime

- operator uint >(filetime left, filetime right)
- operator uint >=(filetime left, filetime right)

Comparison operation.

```
operator uint > (
   filetime left,
   filetime right
)
```

Return value

Returns **1** if the first filetime is greater than the second one. Otherw ise, it returns **0**.

---

filetime >= filetime

Comparison operation.

```
operator uint >= (
   filetime left,
   filetime right
)
```

Return value

Returns **1** if the first filetime is greater or equal the second one. Otherw ise, it returns **0**.

Related links

- Date & Time

datetimetoftime

Converting date from datetime into filetime.

```
func uint datetimetoftime (
   datetime dt,
   filetime ft
)
```

Parameters

dt	Datetime structure.
ft	The variable of the filetime type for getting the result.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Date & Time

ftimetodatetime

Converting date from filetime into datetime.

```
func datetime ftimetodatetime (
   filetime ft,
   datetime dt,
   uint local
)
```

Parameters

ft	A structure of the filetime type. Can be taken from the finfo structure.
dt	A datetime structure for getting the result.
local	Specify 1 if you need to take the local time into account.

Return value

Returns the parameter **dt**.

Related links

- Date & Time

getfiledatetime

Getting date and time as strings. Get the data and time of the last file modification as strings.

```
func getfiledatetime (
   filetime ftime,
   str date,
   str time
)
```

Parameters

ftime	A structure of the filetime type. Can be taken from the finfo structure.
date	The string for w riting date. It can be 0->str.
time	The string for w riting time. It can be 0->str.

Related links

- Date & Time

datetime

The datetime structure. An object of the datetime type is used to w ork w ith time. This type can contain information about date and time.

```
type datetime
{
   ushort year
   ushort month
   ushort dayofweek
   ushort day
   ushort hour
   ushort minute
   ushort second
   ushort msec
}
```

Members

year	Year.
month	Month.
dayofweek	Weekday. Counted from 0. 0 is Sunday, 1 is Monday...
day	Day.
hour	Hours.
minute	Minutes.
second	Seconds.
msec	Milliseconds.

Related links

- Date & Time

filetime

The filetime structure. The filetime type is used to w ork w ith time of files.

```
type filetime
{
   uint lowdtime
   uint highdtime
}
```

Members

lowdtime	Low uint value.
highdtime	High uint value.

Related links

- Date & Time

Dbf

This library is used to w ork w ith **dbf** files. The formats **dBase III** and **dBase IV** are supported. To be able to w ork, you should describe a variable of the **dbf** type. For using this library, it is required to specify the file dbf.g (from Lib subfolder) w ith include command.

```
include : $"...\gentee\lib\dbf\dbf.g"
```

- Operators
- Methods
- Field methods

Operators

* dbf	Get the number of records in the database.
foreach var,dbf	Foreach operator.

Methods

dbf.append	Adding a record.
dbf.bof	Determine is the current record is the first one.
dbf.bottom	Move to the last record.
dbf.close	Close a database.
dbf.create	Create a dbf file and open it.
dbf.del	Set/clear the deletion mark for the current record.
dbf.empty	Creating an empty copy.
dbf.eof	Determine is the current record is in the database.
dbf.geterror	Getting an error code.
dbf.go	Move to the record w ith the specified number.
dbf.isdel	Getting the record deletion mark.
dbf.open	Open a database (a dbf file).
dbf.pack	Pack a database.
dbf.recno	Getting the number of the current record.
dbf.skip	Moving to another record.
dbf.top	Move to the first record.

Field methods

dbf.f_count	Number of fields.
dbf.f_date	Getting a date.
dbf.f_decimal	Getting the size of the fractional part in a numerical field.
dbf.f_double	Getting a numerical value.
dbf.f_find	Getting the number of a field by its name.
dbf.f_int	Getting an integer value.
dbf.f_logic	Getting a logical value.
dbf.f_memo	Get the value of a memo field.
dbf.f_name	Get the name of the specified field.
dbf.f_offset	Get the offset of the field.

dbf.f_ptr	Pointer to data.
dbf.f_str	Getting a value.
dbf.f_type	Get the field type.
dbf.f_width	Get the w idth of the specified field.
dbf.fw_date	Writing a date.
dbf.fw_double	Writing a numerical value.
dbf.fw_int	Writing an integer value.
dbf.fw_logic	Writing a logical value.
dbf.fw_memo	Writing a value into a memo field.
dbf.fw_str	Writing a value.

* dbf

Get the number of records in the database.

```
operator uint * (
   dbf dbase
)
```

Return value

The number of records.

Related links

- Dbf

foreach var,dbf

Foreach operator. You can use **foreach** operator to look over all records of the database. **Variable** is a number of the current record.

```
foreach variable,dbf {...}
```

Related links

- Dbf

dbf.append

Adding a record. The method adds a record to a database.

```
method uint dbf.append()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.bof

Determine is the current record is the first one.

```
method uint dbf.bof()
```

Return value

1 is returned if the current record is the first one.

Related links

- Dbf

dbf.bottom

Move to the last record.

```
method uint dbf.bottom()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.close

Close a database.

```
method dbf.close()
```

Related links

- Dbf

dbf.create

Create a dbf file and open it.

```
method uint dbf.create (
   str filename,
   str fields,
   uint ver
)
```

Parameters

filename	The name of the dbf file being created.
fields	The description of database fields. The line containing the description of fields separated by a line break or ';' Field name,Field type,Width,Fractional part length for numbers The name of a field cannot be longer than 10 characters. Possible type fields:
ver	Version. 0 for dBase III or 1 for dBase IV.

$DBFF_CHAR	String.
$DBFF_DATE	Date.
$DBFF_LOGIC	Logical.
$DBFF_NUMERIC	Integer.
$DBFF_FLOAT	Fraction.
$DBFF_MEMO	Memo field.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.del

Set/clear the deletion mark for the current record.

```
method uint dbf.del
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.empty

Creating an empty copy. The method creates the same, but empty database.

```
method uint dbf.empty (
   str outfile
)
```

Parameters

filename	The full name of the dbf file being created.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.eof

Determine is the current record is in the database.

```
method uint dbf.eof (
   fordata fd
)
```

Parameters

fd	This parameter is used in forech operator. Specify 0->fordata.

Return value

Returns 1 if the current record is not defined/found and 0 otherw ise.

Related links

- Dbf

dbf.geterror

Getting an error code. Get the error code in case some method is finished unsuccessfully.

```
method uint dbf.geterror()
```

Return value

The code of the last error is returned.

$ERRDBF_OPEN	Cannot open dbf file.
$ERRDBF_READ	Cannot read dbf file.
$ERRDBF_POS	File position error.
$ERRDBF_EOF	There is not the current record.
$ERRDBF_WRITE	Cannot w rite dbf file.
$ERRDBF_FOVER	The length of the string being w ritten is greater than the size of the field.
$ERRDBF_TYPE	Incompatible field type.
$ERRDBT_OPEN	Cannot open dbt file.
$ERRDBT_READ	Cannot read dbt file.
$ERRDBT_POS	An error of positioning in the dbt file.
$ERRDBT_WRITE	Cannot w rite dbt file.

Related links

- Dbf

dbf.go

Move to the record w ith the specified number.

```
method uint dbf.go (
   uint num
)
```

Parameters

num	The required record number starting from 1.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.isdel

Getting the record deletion mark. Determine if the current record is marked as deleted.

```
method uint dbf.isdel()
```

Return value

1 is returned if the current record is marked as deleted.

Related links

- Dbf

dbf.open

Open a database (a dbf file).

```
method uint dbf.open (
   str name
)
```

Parameters

name	The name of the dbf file being opened.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.pack

Pack a database. The database is copied into a new file excluding records marked as deleted.

```
method uint dbf.pack (
   str outfile
)
```

Parameters

outfile	The name of the new dbf file.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.recno

Getting the number of the current record.

```
method uint dbf.recno()
```

Return value

The number of the current record or 0 if the record is not defined.

Related links

- Dbf

dbf.skip

Moving to another record. Move forw ard or backw ard for the specified number of records.

```
method uint dbf.skip (
   int step
)
```

Parameters

step	The step of moving. If it is less than zero, the move w ill be tow ard the beginning of the database.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.top

Move to the first record.

```
method uint dbf.top()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.f_count

Number of fields.

```
method uint dbf.f_count()
```

Return value

Returns the number of fields.

Related links

- Dbf

dbf.f_date

- method datetime dbf.f_date(datetime dt, uint num)
- method str dbf.f_date(str val, uint num)

Getting a date. Getting the date from the specified field of the current record into the structure datetime.

```
method datetime dbf.f_date (
   datetime dt,
   uint num
)
```

Parameters

dt	The structure for getting the date.
num	Field number beginning w ith 1.

Return value

Returns the parameter **dt**.

---

dbf.f_date

Getting the date from the specified field of the current record as a string.

```
method str dbf.f_date (
   str val,
   uint num
)
```

Parameters

val	The string for getting the date.
num	Field number beginning w ith 1.

Return value

Returns the parameter **val**.

Related links

- Dbf

dbf.f_decimal

Getting the size of the fractional part in a numerical field.

```
method uint dbf.f_decimal (
   uint num
)
```

Parameters

num	Field number beginning w ith 1.

Return value

The size of the fractional part.

Related links

- Dbf

dbf.f_double

Getting a numerical value. Get a numerical value of the double type from the specified field of the current record.

```
method double dbf.f_double (
   uint num
)
```

Parameters

num	Field number beginning w ith 1.

Return value

A value of the double type.

Related links

- Dbf

dbf.f_find

Getting the number of a field by its name.

```
method uint dbf.f_find (
   str name
)
```

Parameters

name	The name of the field.

Return value

The number of the field w ith the specified name or 0 in case of an error.

Related links

- Dbf

dbf.f_int

Getting an integer value. Get a numerical value of the int type from the specified field of the current record.

```
method int dbf.f_int (
   uint num
)
```

Parameters

num	Field number beginning w ith 1.

Return value

A number of the int type is returned.

Related links

- Dbf

dbf.f_logic

Getting a logical value. Get the value of the logical field from the current record.

```
method uint dbf.f_logic (
   uint num
)
```

Parameters

num	Field number beginning w ith 1.

Return value

Returns the value of the logical field.

$DBF_LFALSE	The value of the logical field is FALSE.
$DBF_LTRUE	The value of the logical field is TRUE.
$DBF_LUNKNOWN	The value of the logical field is undefined.

Related links

- Dbf

dbf.f_memo

Get the value of a memo field. Get the value of the memo field from the current record.

```
method uint dbf.f_memo (
   str val,
   uint num
)
```

Parameters

val	The string for w riting the value.
num	Field number beginning w ith 1.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.f_name

Get the name of the specified field.

```
method str dbf.f_name (
   uint num
)
```

Parameters

num	Field number beginning w ith 1.

Return value

Returns the name of the specified field.

Related links

- Dbf

dbf.f_offset

Get the offset of the field.

```
method uint dbf.f_offset (
   uint num
)
```

Parameters

num	Field number beginning w ith 1.

Return value

Returns the offset of this field.

Related links

- Dbf

dbf.f_ptr

Pointer to data. Get the pointer to the contents of this field from the current record.

```
method uint dbf.f_ptr (
   uint num
)
```

Parameters

num	Field number beginning w ith 1.

Return value

Returns the pointer to this field.

Related links

- Dbf

dbf.f_str

Getting a value. Get the value of the field from the current record as a string.

```
method str dbf.f_str (
   str val,
   uint num
)
```

Parameters

val	The string for getting the value.
num	Field number beginning w ith 1.

Return value

Returns the parameter **val**.

Related links

- Dbf

dbf.f_type

Get the field type.

```
method uint dbf.f_type (
   uint num
)
```

Parameters

num	Field number beginning w ith 1.

Return value

Returns the type of this field. It can be one of the follow ing values.

$DBFF_CHAR	String.
$DBFF_DATE	Date.
$DBFF_LOGIC	Logical.
$DBFF_NUMERIC	Integer.
$DBFF_FLOAT	Fraction.
$DBFF_MEMO	Memo field.

Related links

- Dbf

dbf.f_width

Get the w idth of the specified field.

```
method uint dbf.f_width (
   uint num
)
```

Parameters

num	Field number beginning w ith 1.

Return value

Returns the w idth of the field.

Related links

- Dbf

dbf.fw_date

Writing a date. Write a date into the specified field of the current record.

```
method uint dbf.fw_date (
   datetime dt,
   uint num
)
```

Parameters

dt	The structure datetime containing the date.
num	Field number beginning w ith 1.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.fw_double

Writing a numerical value. Write a numerical value into the specified field of the current record.

```
method uint dbf.fw_double (
   double dval,
   uint num
)
```

Parameters

dval	The number being w ritten.
num	Field number beginning w ith 1.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.fw_int

Writing an integer value. Write a value of the int type into the specified field of the current record.

```
method uint dbf.fw_int (
   int ival,
   uint num
)
```

Parameters

ival	The number being w ritten.
num	Field number beginning w ith 1.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.fw_logic

Writing a logical value. Write a logical value into the specified field of the current record.

```
method uint dbf.fw_logic (
   uint val,
   uint num
)
```

Parameters

val	Number 1 or 0.
num	Field number beginning w ith 1.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.fw_memo

Writing a value into a memo field. Write a value into the specified memo field of the current record.

```
method uint dbf.fw_memo (
   str val,
   uint num
)
```

Parameters

val	The string being w ritten.
num	Field number beginning w ith 1.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

dbf.fw_str

Writing a value. Write a value into the specified field of the current record.

```
method uint dbf.fw_str (
   str val,
   uint num
)
```

Parameters

val	The string being w ritten.
num	Field number beginning w ith 1.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Dbf

Files

File system functions.

- Methods
- Functions
- Search and fileinfo functions
- Related Methods

Methods

file.close	Close a file.
file.getsize	Get the size of the file.
file.gettime	Get the time w hen the file w as last modified.
file.open	Open a file.
file.read	Reading a file.
file.setpos	Set the current position in the file.
file.settime	Set time for a file.
file.write	Writing to a file.

Functions

copyfile	Copy a file.
copyfiles	Copying files and directories by mask.
createdir	Create a directory.
deletedir	Delete a directory.
deletefile	Delete a file.
delfiles	Deleting files and directories by mask.
direxist	Checking if a directory exists.
fileexist	Checking if a file exists.
getcurdir	Getting the current directory.
getdrives	Get the names of available disks.
getdrivetype	Get the type of a disk.
getfileattrib	Getting file attributes.
getmodulename	Get the file name of the currently running application.
getmodulepath	Get the path to the running EXE file.
gettempdir	Get the temporary directory of the application.
isequalfiles	Check if files are equal.
movefile	Rename, move a file or a directory.
setattribnormal	Setting the attribute $FILE_ATTRIBUTE_NORMAL.
setcurdir	Setting the current directory.
setfileattrib	Set file attributes.
verifypath	Verifying a path and creating all absent directories.

Search and fileinfo functions

finfo	File information structure.
ffind	File search structure.
foreach var,ffind	Foreach operator.
ffind.init	Initializing file search.
getfileinfo	Get information about a file or directory.

Related Methods

arrstr.read	Read a multi-line text file to array of strings.
arrstr.write	Write an array of strings to a multi-line text file.
buf.read	Reading from a file.
buf.write	Writing to a file.
buf.writeappend	Appending data to a file.
str.read	Read a string from a file.
str.write	Writing a string to a file.
str.writeappend	Appending string to a file.

file.close

Close a file.

```
method uint file.close( )
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Files

file.getsize

Get the size of the file.

```
method uint file.getsize( )
```

Return value

The size of the file less 4GB.

Related links

- Files

file.gettime

Get the time w hen the file w as last modified.

```
method uint file.gettime (
   filetime ft
)
```

Parameters

ft	The variable for getting the time of a file.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Files

file.open

Open a file.

```
method uint file.open (
   str name,
   uint flag
)
```

Parameters

name	The name of the file to be opened.
flag	The follow ing flags can be used.

$OP_READONLY	Open as read-only.
$OP_EXCLUSIVE	Open in the exclusive mode.
$OP_CREATE	Create the file.
$OP_ALWAYS	Create the file only if it does not exist.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Files

file.read

- method uint file.read(uint ptr, uint size)
- method uint file.read(buf rbuf, uint size)

Reading a file.

```
method uint file.read (
   uint ptr,
   uint size
)
```

Parameters

ptr	The pointer w here the file w ill be read.
size	The size of the data being read.

Return value

The function returns the size of the read data.

---

file.read

Reading a file.

```
method uint file.read (
   buf rbuf,
   uint size
)
```

Parameters

rbuf	The buffer w here data w ill be read. Reading is carried out by adding data to the buffer. It means that read data w ill be added to those already existing in the buffer.
size	The size of the data being read.

Return value

The function returns the size of the read data.

Related links

- Files

file.setpos

Set the current position in the file.

```
method uint file.setpos (
   int offset,
   uint mode
)
```

Parameters

offset	Position offset.
mode	The type of moving the position.

$FILE_BEGIN	From the beginning of the file.
$FILE_CURRENT	From the current position.
$FILE_END	From the end of the file.

Return value

The function returns the current position in the file.

Related links

- Files

file.settime

Set time for a file.

```
method uint file.settime (
   filetime ft
)
```

Parameters

ft	New time for the file.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Files

file.write

- method uint file.w rite(uint data, uint size)
- method uint file.w rite(buf rbuf)
- method uint file.w ritepos(uint pos, uint data, uint size)

Writing to a file.

```
method uint file.write (
   uint data,
   uint size
)
```

Parameters

data	The pointer to the memory w hich data w ill be w ritten.
size	The size of the data being w ritten.

Return value

The function returns the size of the w ritten data.

---

file.write

Writing to a file.

```
method uint file.write (
   buf rbuf
)
```

Parameters

rbuf	The buffer from w hich data w ill be w ritten.

Return value

The function returns the size of the w ritten data.

---

file.writepos

Writing to a file from the position.

```
method uint file.writepos (
   uint pos,
   uint data,
   uint size
)
```

Parameters

pos	The start position for w riting.
data	The pointer to the memory w hich data w ill be w ritten.
size	The size of the data being w ritten.

Return value

The function returns the size of the w ritten data.

Related links

- Files

copyfile

Copy a file.

```
func uint copyfile (
   str name,
   str newname
)
```

Parameters

name	The name of an existing file.
newname	A new file name and path. If the file already exists, it w ill be overw ritten.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Files

copyfiles

- func uint copyfiles(str src, str dir, uint flag, uint mode, uint process)
- func uint defcopyproc(uint code, uint left, uint right)

Copying files and directories by mask.

```
func uint copyfiles (
   str src,
   str dir,
   uint flag,
   uint mode,
   uint process
)
```

Parameters

src The names of mask of the files or directories being copied.

dir The directory w here files w ill be copied.

flag The combination of search and copy flags.

$FIND_DIR	Search only for directories.
$FIND_FILE	Search only for files.
$FIND_RECURSE	Search in all subdirectories.
$COPYF_RO	Overw rite files w ith the attribute read-only.
$COPYF_SAVEPATH	Keep relative paths w hile copying files from subdirectories.
$COPYF_ASK	Prompt before copying files already existing.

mode What to do if the file being copied already exists.

$COPY_OVER	Overw rite.
$COPY_SKIP	Skip.
$COPY_NEWER	Overw rite if new er.
$COPY_MODIFIED	Overw rite if modified.

procce ss The identifier of the function handling messages. You can use **&defcopyproc** as a default process function.

Return value

The function returns 1 if the copy operation is successful, otherw ise it returns 0.

defcopyproc

This is a default process function for **copyfiles**. You can develop and use your ow n process function like it.

```
func uint defcopyproc (
   uint code,
   uint left,
   uint right
)
```

Parameters

code The message code.

$COPYN_FOUND	The object for copying is found.
$COPYN_NEWDIR	A directory is created.
$COPYN_ERRDIR	Cannot create a directory.
$COPYN_ASK	Copy request.
$COPYN_ERRFILE	Error w hile creating a file.
$COPYN_NEWFILE	A file w as created.
$COPYN_BEGIN	Start copying file.
$COPYN_PROCESS	A file is being copied.

$COPYN_END	Copying is over.
$COPYN_ERRWRITE	Error w hile w riting a file.

left	Additional parameter.
right	Additional parameter.

Return value

You should return one of the follow ing values:

$COPYR_NOTHING	Do nothing.
$COPYR_BREAK	Break copying.
$COPYR_RETRY	Retry.
$COPYR_SKIP	Skip.
$COPYR_OVER	Write over.
$COPYR_OVERALL	Write over all files.
$COPYR_SKIPALL	Skip all files.

Related links

- Files

createdir

Create a directory.

```
func uint createdir (
   str name
)
```

Parameters

name	The name of the directory being created.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Files

deletedir

Delete a directory.

```
func uint deletedir (
   str name
)
```

Parameters

name	The name of the directory being deleted.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Files

deletefile

Delete a file.

```
func uint deletefile (
   str name
)
```

Parameters

name	The name of the file being deleted.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Files

delfiles

Deleting files and directories by mask. Directories are deleted together w ith all files and subdirectories. Be really careful w hile using this function. For example, calling

```
delfiles( "c:\\temp", $FIND_DIR | $FIND_FILE | $FIND_RECURSE )
```

w ill delete all files and directories named temp on the disk N: including a search in all directories. In this case temp is considered a mask and since the flag $FIND_RECURSE is specified, the entire disk C: w ill be searched. If you just need to delete the directory temp w ith all its subdirectories and files, you should call

```
delfiles("c:\\temp", $FIND_DIR )
```

Calling

```
delfiles( "c:\\temp\\*.tmp", $FIND_FILE )
```

w ill delete all files in the directory tmp leaving subdirectories.

```
func delfiles (
   str name,
   uint flag
)
```

Parameters

name	The name of mask for searching.
flag	Search and delete flags.

$FIND_DIR	Search only for directories.
$FIND_FILE	Search only for files.
$FIND_RECURSE	Search in all subdirectories.
$DELF_RO	Delete files w ith the attribute read-only.

Related links

- Files

direxist

Checking if a directory exists.

```
func uint direxist (
   str name
)
```

Parameters

name	Directory name.

Return value

The function returns 1, if the specified directory exists.

Related links

- Files

fileexist

Checking if a file exists.

```
func uint fileexist (
   str name
)
```

Parameters

name	Filename.

Return value

The function returns 1, if the specified file exists.

Related links

- Files

getcurdir

Getting the current directory.

```
func str getcurdir (
   str dir
)
```

Parameters

dir	The string for getting the result.

Return value

Returns the parameter **dir**.

Related links

- Files

getdrives

Get the names of available disks.

```
func arrstr getdrives <result>()
```

Return value

The array (arrstr) of the disk names.

Related links

- Files

getdrivetype

Get the type of a disk.

```
func uint getdrivetype (
   str name
)
```

Parameters

drive	The name of a disk w ith a closing slash. For example: **C:\**

Return value

Returns one of the follow ing values:

$DRIVE_UNKNOWN	Unknow n type.
$DRIVE_NO_ROOT_DIR	Invalid path to root.
$DRIVE_REMOVABLE	Removable disk.
$DRIVE_FIXED	Fixed disk.
$DRIVE_REMOTE	Netw ork disk.
$DRIVE_CDROM	CD/DVD-ROM drive.
$DRIVE_RAMDISK	RAM disk.

Related links

- Files

getfileattrib

Getting file attributes.

```
func uint getfileattrib (
   str name
)
```

Parameters

name	Filename.

Return value

The function returns file attributes. It returns 0xFFFFFFFF in case of an error.

$FILE_ATTRIBUTE_READONLY	Read-only.
$FILE_ATTRIBUTE_HIDDEN	Hidden.
$FILE_ATTRIBUTE_SYSTEM	System.
$FILE_ATTRIBUTE_DIRECTORY	Directory.
$FILE_ATTRIBUTE_ARCHIVE	Archive.
$FILE_ATTRIBUTE_NORMAL	Normal.
$FILE_ATTRIBUTE_TEMPORARY	Temporary.
$FILE_ATTRIBUTE_COMPRESSED	Compressed.

Related links

- Files

getmodulename

Get the file name of the currently running application.

```
func str getmodulename (
   str dest
)
```

Parameters

dest	The string for getting the name.

Return value

Returns the parameter **dest**.

Related links

- Files

getmodulepath

Get the path to the running EXE file.

```
func str getmodulepath (
   str dest,
   str subfolder
)
```

Parameters

dest	Result string.
subfolder	Additional path. This string w ill be added to the obtained result. It can be empty.

Return value

Returns the parameter **dest**.

Related links

- Files

gettempdir

Get the temporary directory of the application. When this function is called for the first time, in the temporary directory there w ill be created a directory named genteeXX, w here XX is a unique number for this running application. When the application is closed, the directory w ill be deleted w ith all its files.

```
func str gettempdir (
   str dir
)
```

Parameters

dir	The string for getting the result.

Return value

Returns the parameter **dir**.

Related links

- Files

isequalfiles

Check if files are equal. The function compares tw o files.

```
func uint isequalfiles (
   str left,
   str right
)
```

Parameters

left	The name of the first file to be compared.
right	The name of the second file to be compared.

Return value

The function returns 1 if the files are equal, otherw ise it returns 0.

Related links

- Files

movefile

Rename, move a file or a directory.

```
func uint movefile (
   str name,
   str newname
)
```

Parameters

name	The name of an existing file or a directory.
newname	A new file name and path.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Files

setattribnormal

Setting the attribute $FILE_ATTRIBUTE_NORMAL.

```
func uint setattribnormal (
   str name
)
```

Parameters

name	Filename.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Files

setcurdir

Setting the current directory.

```
func uint setcurdir (
   str dir
)
```

Parameters

dir	The name of the new current directory.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Files

setfileattrib

Set file attributes.

```
func uint setfileattrib (
   str name,
   uint attrib
)
```

Parameters

name	Filename.
attrib	File attributes.

$FILE_ATTRIBUTE_READONLY	Read-only.
$FILE_ATTRIBUTE_HIDDEN	Hidden.
$FILE_ATTRIBUTE_SYSTEM	System.
$FILE_ATTRIBUTE_DIRECTORY	Directory.
$FILE_ATTRIBUTE_ARCHIVE	Archive.
$FILE_ATTRIBUTE_NORMAL	Normal.
$FILE_ATTRIBUTE_TEMPORARY	Temporary.
$FILE_ATTRIBUTE_COMPRESSED	Compressed.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Files

verifypath

Verifying a path and creating all absent directories.

```
func uint verifypath (
   str name,
   arrstr dirs
)
```

Parameters

name	The name of the path to be verified.
dirs	An array for getting all the directories being created. It can be 0->arrstr.

Return value

The function returns 1 if directories have been verified and created successfully. In case of an error, the function returns 0 and the last dirs item contains the name w here there occurred an error w hile creating a directory.

Related links

- Files

finfo

File information structure. This structure is used by getfileinfo function and foreach operator.

```
type finfo
{
   str fullname
   str name
   uint attrib
   filetime created
   filetime lastwrite
   filetime lastaccess
   uint sizehi
   uint sizelo
}
```

Members

fullname	The full name of the file or directory.
name	The name of the file or directory.
attrib	File attributes.
created	Creation time.
lastwrite	Last modification time.
lastaccess	Last access time.
sizehi	High size uint.
sizelo	Low size uint.

Related links

- Files

ffind

File search structure. This structure is used in foreach operator. You must not modify fields of *ffind* variable. You must initialize it w ith ffind.init method.

```
type ffind <index = finfo>
{
   stack deep
   str initname
   str wildcard
   uint flag
}
```

Members

deep	Hidden data.
initname	Hidden data.
wildcard	Hidden data.
flag	Hidden data.

Related links

- Files

foreach var,ffind

Foreach operator. You can use **foreach** operator to look over files in some directory w ith the specified w ildcard. The finfo structure w ill be returned for each found file. You must call ffind.init before using **foreach**.

```
ffind fd
fd.init( "c:\\*.exe", $FIND_FILE | $FIND_RECURSE )
foreach finfo cur,fd
{
   print( "\( cur.fullname )\n" )
}
```

```
foreach variable,ffind {...}
```

Related links

- Files

ffind.init

Initializing file search. An object of the ffind type is used to search for files and directories by mask. Before starting the search, you should call the init method. After this it is possible to use the initiated object in the **foreach** loop. The finfo structure w ill be returned for each found file.

```
method ffind.init (
   str name,
   uint flag
)
```

Parameters

name	The mask for searching files and directories.
flag	The combination of the follow ing flags:

$FIND_DIR	Search only for directories.
$FIND_FILE	Search only for files.
$FIND_RECURSE	Search in all subdirectories.

Related links

- Files

getfileinfo

Get information about a file or directory.

```
func uint getfileinfo (
   str name,
   finfo fi
)
```

Parameters

name	The name of a file or directory.
fi	The structure finfo all the information w ill be w ritten to.

Return value

It returns 1 if the file is found, it returns 0 otherw ise.

Related links

- Files

FTP

FTP protocol. You must call inet_init function before using this library. For using this library, it is required to specify the file ftp.g (from lib\ftp subfolder) w ith include command.

```
include : $"...\gentee\lib\ftp\ftp.g"
```

- Common internet functions
- URL strings

ftp.close	Terminates the FTP connection.
ftp.command	Sends a command.
ftp.createdir	Creates a new directory.
ftp.deldir	Deletes a directory.
ftp.delfile	Deletes a file.
ftp.getcurdir	Retrieves the current directory.
ftp.getfile	Retrieves a file.
ftp.getsize	Retrieves the file size from the FTP server.
ftp.gettime	Retrieves the file time.
ftp.lastresponse	The last response from the FTP server.
ftp.list	List of files.
ftp.open	Establishes an FTP connection.
ftp.putfile	Stores a file on the FTP server.
ftp.rename	Renames a file.
ftp.setattrib	Sets the attributes.
ftp.setcurdir	Sets the current directory.

Common internet functions

inet_close	Closing the library.
inet_error	Getting an error code.
inet_init	Library initialization.
inet_proxy	Using a proxy server.
inet_proxyenable	Enabling/disabling a proxy server.
inetnotify_func	Message handling function.

URL strings

str.iencoding	Recoding a string.
str.ihead	Getting a header.
str.ihttpinfo	Processing a header.
str.iurl	The method is used to parse a URL address.

ftp.close

Terminates the FTP connection. The method terminates the connection on the FTP server.

```
method uint ftp.close()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.command

Sends a command. This methos is used to send the specified command directly to an FTP server. The response from the server can be received w ith help of the ftp.lastresponse method.

```
method uint ftp.command (
   str cmd
)
```

Parameters

cmd	The command text.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.createdir

Creates a new directory. The method creates a new directory on the FTP server.

```
method uint ftp.createdir (
   str dirname
)
```

Parameters

dirname	The name of the directory

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.deldir

Deletes a directory. This method deletes a directory stored on the FTP server.

```
method uint ftp.deldir (
   str dirname
)
```

Parameters

dirname	The name of the required directory

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.delfile

Deletes a file. The method deletes a file stored on the FTP server.

```
method uint ftp.delfile (
   str filename
)
```

Parameters

filename	The name of the required file.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.getcurdir

Retrieves the current directory. The method retrieves the current directory name from the FTP server.

```
method uint ftp.getcurdir (
   str dirname
)
```

Parameters

dirname	Result string.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.getfile

- method uint ftp.getfile(str filename, buf databuf, uint flag)
- method uint ftp.getfile(str srcname, str destname, uint flag)

Retrieves a file. The method retrieves files from the FTP server.

```
method uint ftp.getfile (
   str filename,
   buf databuf,
   uint flag
)
```

Parameters

filename	The dow nloaded file name.
databuf	The received data buffer. Data are not stored on a drive.
flag	Additional flags.

$FTP_BINARY	A binary file is dow nloaded.
$FTP_TEXT	A text file is dow nloaded. This is a default mode.
$FTP_STR	Appends zero to the end of received data.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

ftp.getfile

The method retrieves files from the FTP server.

```
method uint ftp.getfile (
   str srcname,
   str destname,
   uint flag
)
```

Parameters

srcname	The dow nloaded file name.
destname	A new file name on user's machine.
flag	Flags.

$FTP_BINARY	A binary file is dow nloaded.
$FTP_TEXT	A text file is dow nloaded. This is a default mode.
$FTP_CONTINUE	Proceeds w ith retrieving.
$FTP_SETTIME	Sets the same file times as on the FTP server.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.getsize

Retrieves the file size from the FTP server.

```
method uint ftp.getsize (
   str name,
   uint psize
)
```

Parameters

name	Filename.
psize	A pointer to uint value is used to store the file size.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.gettime

Retrieves the file time. Retrieves last w rite times for the file on the FTP server.

```
method uint ftp.gettime (
   str name,
   datetime dt
)
```

Parameters

name	Filename.
dt	The variable of datetime type is used to retrieve the file time.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.lastresponse

The last response from the FTP server. The method returns the last response from the FTP server.

```
method str ftp.lastresponse (
   str out
)
```

Parameters

out	Result string.

Return value

Returns the parameter **out**.

Related links

- FTP

ftp.list

List of files. The method retrieves a list of files and directories from the FTP server.

```
method uint ftp.list (
   str data,
   str mode
)
```

Parameters

list	Result string.
cmd	The command is used to retrieve a list of files.

"LIST"	Returns a list of files in the format of the LIST command.
"NLST"	Returns a list of filenames w ith no other information.
"MLSD"	Returns a list of files in the format of the MLSD command.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.open

Establishes an FTP connection. This method establishes an FTP connection w ith the server. This method must be called before other methods dealing w ith the FTP server are called.

```
method uint ftp.open (
   str url,
   str user,
   str password,
   uint flag,
   uint notify
)
```

Parameters

url	The name or address of the FTP server.
user	A user name. If the string is empty, anonymous connections are used.
password	A user passw ord. If the connection is anonymous, your e-mail address is required.
flag	Connection flags.

$FTP_ANONYM	Anonymous connection.
$FTP_PASV	Establishes a connection in passive mode.

notify	Function is used to receive notification messages. This parameter can be zero.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.putfile

Stores a file on the FTP server. This method is used to upload the required file from the remote host to the FTP server.

```
method uint ftp.putfile (
   str srcname,
   str destname,
   uint flag
)
```

Parameters

srcname	The name of the required source file.
destname	The name of a file stored on the FTP server.
flag	Flags. If the flag of the binary or text mode is not specified, the method makes effort to determine a file type.

$FTP_BINARY	A binary file is uploaded.
$FTP_TEXT	A text file is uploaded.
$FTP_CONTINUE	To proceed w ith file uploading.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.rename

Renames a file. This method renames a file or directory stored on the FTP server.

```
method uint ftp.rename (
   str from,
   str to
)
```

Parameters

from	The current name of the file or directory.
to	A new name.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.setattrib

Sets the attributes. This method sets the attributes for the file or the directory.

```
method uint ftp.setattrib (
   str name,
   uint mode
)
```

Parameters

name	The name of a file or directory.
mode	The attributes for the file.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

ftp.setcurdir

Sets the current directory. This method sets a new current directory.

```
method uint ftp.setcurdir (
   str dirname
)
```

Parameters

dirname	The name of a new directory.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- FTP

Gentee API

There is an option for softw are engineers to run programs in Gentee in their ow n applications. To do that, it is enough to connect the gentee.dll file. It contains several importable functions, w hich are responsible for compilation and execution of the programs.

- Types

gentee_call	Call the function from the bytecode.
gentee_compile	Program compilation.
gentee_deinit	End of w orking w ith gentee.
gentee_getid	Get the object's identifier by its name.
gentee_init	Initialization of gentee.
gentee_load	Load and launch the bytecode.
gentee_ptr	Get Gentee structures.
gentee_set	This function specifies some gentee parameters.

Types

gentee	The main structure of gentee engine.
compileinfo	The structure for the using in gentee_compile function.
optimize	The structure for the using in compileinfo structure.

gentee_call

Call the function from the bytecode. The bytecode should be previously loaded w ith the gentee_load or gentee_compile functions.

```
uint CDECLCALL gentee_call (
   uint id,
   puint result,
     ...
)
```

Parameters

id	The identifier of the called object. Can be obtained by gentee_getid function.
result	Pointer to the memory space, to w hich the result w ill be w ritten. It can be the pointer to **uint**, **long** or **double**.
...	Required parameters of the function.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Gentee API

gentee_compile

Program compilation. This function allow s to compile and run programs in Gentee.

```
uint STDCALL gentee_compile (
   pcompileinfo compinit
)
```

Parameters

compinit	The pointer to compileinfo structure w ith the specified compiling options.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Gentee API

gentee_deinit

End of w orking w ith gentee.dll. This function should be called w hen the w ork w ith Gentee is finished.

```
uint STDCALL gentee_deinit( void )
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Gentee API

gentee_getid

Get the object's identifier by its name.

```
uint CDECLCALL gentee_getid (
   pubyte name,
   uint count,
     ...
)
```

Parameters

name	The name of the object. If you w ant to find a method then use '**@**' at the beginning of the name. For example, "**@mymethod**". If you w ant to find an operator then use '**#**' at the beginning of the name. For example, "**#+=**".
count	The count of the follow ing parameters. If you w ant to find any object w ith the defined name then specify the follow ing flag.

GID_ANYOBJ	Find any object

...	Specify the sequence of the type's identifiers of the parameters. If the parameter of the function has "**of**" subtype then specify it in the HIWORD of the value.

Return value

Returns objects identifier or **0**, if the object w as not found.

Related links

- Gentee API

gentee_init

Initialization of gentee.dll. This function should be called before beginning to w ork w ith Gentee.

```
uint STDCALL gentee_init (
   uint flags
)
```

Parameters

flags Flags.

G_CONSOLE	Console application.
G_SILENT	Don't display any service messages.
G_CHARPRN	Print Window s characters.
G_ASM	Run-time converting a bytecode to assembler.
G_TMPRAND	Random name of t temporary directory.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Gentee API

gentee_load

Load and launch the bytecode. This function loads the bytecode from the file or the memory and launch it if it is required. You can create the bytecode w ith gentee_compile function.

```
uint STDCALL gentee_load (
   pubyte bytecode,
   uint flag
)
```

Parameters

bytecode	The pointer to the bytecode or the filename of .ge file.
flag	Flags.

GLOAD_ARGS	Get command line arguments
GLOAD_FILE	Read file to load the bytecode. The bytecode is name of the loading file
GLOAD_RUN	Load <entry> functions and run <main> function.

Return value

The result of the executed bytecode if GLOAD_RUN w as defined.

Related links

- Gentee API

gentee_ptr

Get Gentee structures. This function returns pointers to global Gentee structures.

```
pvoid STDCALL gentee_ptr (
   uint par
)
```

Parameters

par The identifier of the parameter.

GPTR_GENTEE	Pointer to gentee structure. See gentee.
GPTR_VM	Pointer to vm structure
GPTR_COMPILE	Pointer to compile structure
GPTR_CALL	Pointer to gentee_call function

.

Return value

The pointer to according global Gentee structure.

Related links

- Gentee API

gentee_set

This function specifies some gentee parameters.

```
uint STDCALL gentee_set (
   uint state,
   pvoid val
)
```

Parameters

state The identifier of the parameter.

GSET_TEMPDIR	Specify the custom temporary directory
GSET_PRINT	Specify the custom print function
GSET_MESSAGE	Specify the custom message function
GSET_EXPORT	Specify the custom export function
GSET_ARGS	Specify the command-line arguments
GSET_FLAG	Specify flags
GSET_DEBUG	Specify the custom debug function
GSET_GETCH	Specify the custom getch function

val The new value of the parameter.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Gentee API

gentee

The main structure of gentee engine.

```
typedef struct
{
   uint flags;
   uint multib;
   uint tempid;
   str tempdir;
   uint tempfile;
   printfunc print;
   getchfunc getch;
   messagefunc message;
   exportfunc export;
   debugfunc debug;
   pubyte args;
} gentee, *pgentee;
```

Members

flags	Flags. **G_CONSOLE** — Console application. **G_SILENT** — Don't display any service messages. **G_CHARPRN** — Print Window s characters. **G_ASM** — Run-time converting a bytecode to assembler. **G_TMPRAND** — Random name of the temporary directory.
multib	1 if the current page is tw o-bytes code page
tempid	The indetifier of the temporary directory.
tempdir	The temporary directory
tempfile	The handle of the file for locking tempdir
print	The custom print function
getch	The custom getch and scan function
message	The custom message function
export	The custom export function
debug	The custom debug function
args	Command -line arguments. arg1 0 arg2 00

Related links

- Gentee API

compileinfo

The structure for the using in gentee_compile function.

```
typedef struct
{
   pubyte input;
   uint flag;
   pubyte libdirs;
   pubyte include;
   pubyte defargs;
   pubyte output;
   pvoid hthread;
   uint result;
   optimize opti;
} compileinfo, * pcompileinfo;
```

Members

input	The Gentee filename. You can specify the Gentee source if the flag CMPL_SRC is defined.
flag	Compile flags.

CMPL_SRC	Specify if compileinfo.input is Gentee source
CMPL_NORUN	Don't run anything after the compilation.
CMPL_GE	Create GE file
CMPL_LINE	Proceed #! at the first string
CMPL_DEBUG	Compilation w ith the debug information
CMPL_THREAD	Compilation in the thread
CMPL_NOWAIT	Do not w ait for the end of the compilation. Use w ith CMPL_THREAD only.
CMPL_OPTIMIZE	Optimize the output GE file.
CMPL_NOCLEAR	Do not clear existing objects in the virtual machine.
CMPL_ASM	Convert the bytecode to assembler code.

libdirs	Folders for searching files: name1 0 name2 0 ... 00. It may be NULL.
include	Include files: name1 0 name2 0 ... 00. These files w ill be compiled at the beginning of the compilation process. It may be NULL.
defargs	Define arguments: name1 0 name2 0 ... 00. You can specify additional macro definitions. For example, **MYMODE = 10**. In this case, you can use **$MYMODE** in the Gentee program. It may be NULL.
output	Ouput filename for GE. In default, .ge file is created in the same folder as .g main file. You can specify any path and name for the output bytecode file. You must specify CMPL_GE flag to create the bytecode file.
hthread	The result handle of the thread if you specified CMPL_THREAD | CMPL_NOWAIT.
result	Result of the program if it w as executed.
opti	Optimize structure. It is used if flag CMPL_OPTIMIZE is defined.

Related links

- Gentee API

optimize

The structure for the using in compileinfo structure.

```
typedef struct
{
   uint flag;
   pubyte nameson;
   pubyte avoidon;
} optimize, * poptimize;
```

Members

flag Flags of the optimization.

OPTI_DEFINE	Delete 'define' objects.
OPTI_NAME	Delete names of objects.
OPTI_AVOID	Delete not used objects.
OPTI_MAIN	Leave only one main function w ith OPTI_AVOID.

nameson Don't delete names w ith the follow ing w ildcards divided by 0 if OPTI_NAME specified

avoidon Don't delete objects w ith the follow ing w ildcards divided by 0 if OPTI_AVOID specified

Related links

- Gentee API

Hash

Hash (Associative array). Variables of the hash type allow you to w ork w ith associative arrays or hash tables. Each item in such an array corresponds to a unique key string. Items are addresses by specifying the corresponding key strings.

- Operators
- Methods
- Type

Operators

hash of type	Specifying the type of items.
* hash	Get the count of items.
hash[name]	Getting an item by a key string.
foreach var,hash	Foreach operator.

Methods

hash.clear	Clear a hash.
hash.create	Creating an item w ith this key.
hash.del	Delete an item w ith this key.
hash.find	Find an item w ith this key.
hash.ignorecase	Ignoring the letter case of keys.
hash.sethashsize	Set the size of a value table.

Type

hash	The main structure of the hash.

hash of type

Specifying the type of items. You can specify **of** type w hen you describe **hash** variable. In default, the type of the items is **uint**.

```
method hash.oftype (
   uint itype
)
```

Related links

- Hash

* hash

Get the count of items.

```
operator uint * (
   hash left
)
```

Return value

Count of hash items.

Related links

- Hash

hash[name]

Getting an item by a key string. In case there is no item, it w ill be created automatically.

```
method uint hash.index (
   str key
)
```

Return value

The **["key"]** item of the hash.

Related links

- Hash

foreach var,hash

- foreach variable,hash {...}
- foreach variable,hash.keys {...}

Foreach operator. You can use **foreach** operator to look over all items of the hash. **Variable** is a pointer to the hash item.

```
foreach variable,hash {...}
```

foreach var,hash.keys

You can use **foreach** operator to look over all keys of the hash.

```
foreach variable,hash.keys {...}
```

Related links

- Hash

hash.clear

Clear a hash. The method removes all items from the hash.

```
method hash.clear()
```

Related links

- Hash

hash.create

Creating an item w ith this key. If an item w ith this key already exists, it w ill be initiated again. Items are created automatically w hen they are addressed as array items for the first time - hashname["key string"].

```
method uint hash.create (
   str key
)
```

Parameters

key	Key value.

Return value

The pointer to the created item is returned.

Related links

- Hash

hash.del

Delete an item w ith this key.

```
method uint hash.del (
   str key
)
```

Parameters

key	Key value.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Hash

hash.find

Find an item w ith this key.

```
method uint hash.find (
   str key
)
```

Parameters

key	Key value.

Return value

Either the pointer to the found item is returned or 0 is returned if there is no item w ith this key.

Related links

- Hash

hash.ignorecase

Ignoring the letter case of keys. Work w ith the keys of this hash table w ithout taking into account the case of letters. The method must be called before any items are added.

```
method uint hash.ignorecase
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Hash

hash.sethashsize

Set the size of a value table. Set the size of the value table for searching for keys. The method must be called before any items are added. The parameter contains the pow er of tw o for calculating the size of the table since the number of items must be the pow er of tw o. By default, the size of a table is 4096 items.

```
method uint hash.sethashsize (
   uint power
)
```

Parameters

power	The pow er of tw o for calculating the size of the table.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Hash

hash

The main structure of the hash.

```
type hash
{
   arr hashes
   uint itype
   uint isize
   uint count
   uint igncase
   hkeys keys
}
```

Members

hashes	Array of hash values. Pointers to hashkey.
itype	The type of the items.
isize	The type of the item.
count	The count of items.
igncase	Equals 1 if the hash ignores case sensetive.
keys	The structure for looking over keys

Related links

- Hash

HTTP

HTTP protocol. You must call inet_init function before using this library. For using this library, it is required to specify the file http.g (from lib\http subfolder) w ith include command.

```
include : $"...\gentee\lib\http\http.g"
```

- Common internet functions
- URL strings

http_get	Getting data via the HTTP protocol.
http_getfile	Dow nloading a file via the HTTP protocol.
http_head	Getting a header via the HTTP protocol.
http_post	Sending data via the HTTP protocol.

Common internet functions

inet_close	Closing the library.
inet_error	Getting an error code.
inet_init	Library initialization.
inet_proxy	Using a proxy server.
inet_proxyenable	Enabling/disabling a proxy server.
inetnotify_func	Message handling function.

URL strings

str.iencoding	Recoding a string.
str.ihead	Getting a header.
str.ihttpinfo	Processing a header.
str.iurl	The method is used to parse a URL address.

http_get

Getting data via the HTTP protocol. The method sends a GET request to the specified URL and w rites data it receives to the databuf buffer.

```
func uint http_get (
   str url,
   buf databuf,
   uint notify,
   uint flag,
   str otherpars
)
```

Parameters

url	The URL address data is received from.
databuf	The buffer for getting data.
notify	The function for getting notifications. It can be 0.
flag	Flags.

$HTTPF_REDIRECT	If redirection is used, dow nload from the new address.
$HTTPF_STR	Add 0 to databuf after data is received. Use this flag if databuf is a string.
$HTTPF_CONTINUE	If the file already exists, resume dow nloading it. It is valid for http_getfile.
$HTTPF_SETTIME	Set the same time for the file as it is on the server. It is valid for http_getfile.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- HTTP

http_getfile

Dow nloading a file via the HTTP protocol. The method sends a GET request to the specified URL and w rites data it receives to the specified file.

```
func uint http_getfile (
   str url,
   str filename,
   uint notify,
   uint flag
)
```

Parameters

url	The URL address for dow nloading.
filename	The name of the file for w riting.
notify	The function for getting notifications. It can be 0.
flag	Flags.

$HTTPF_REDIRECT	If redirection is used, dow nload from the new address.
$HTTPF_STR	Add 0 to databuf after data is received. Use this flag if databuf is a string.
$HTTPF_CONTINUE	If the file already exists, resume dow nloading it. It is valid for http_getfile.
$HTTPF_SETTIME	Set the same time for the file as it is on the server. It is valid for http_getfile.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- HTTP

http_head

Getting a header via the HTTP protocol. The method sends a HEAD request to the specified URL address and partially parses the received data.

```
func uint http_head (
   str url,
   str head,
   httpinfo hi
)
```

Parameters

url	The URL address for getting the header.
head	The string for getting the text of the header.
hi	The variable of the httpinfo type for getting information about the header.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- HTTP

http_post

Sending data via the HTTP protocol. The method sends a POST request w ith the specified string to the specified URL address. It is used to fill out forms automatically.

```
func uint http_post (
   str url,
   str data,
   str result,
   uint notify,
   str otherpars
)
```

Parameters

url	The URL address w here the data w ill be sent.
data	The string w ith the data being sent. Before the data is sent, request strings w ith parameters should be recoded w ith the help of the str.iencoding method.
result	The string for getting a response from the server.
notify	The function for getting notifications. It can be 0.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- HTTP

INI File

INI files. This library allow s you to w ork w ith ini files. Variables of the ini type allow you to w ork w ith them. For using this library, it is required to specify the file ini.g (from lib\ini subfolder) w ith include command.

```
include : $"...\gentee\lib\ini\ini.g"
```

- Methods
- Functions

Methods

ini.delkey	Deleting a key.
ini.delsection	Deleting a section.
ini.getnum	Get the numerical value of an entry.
ini.getvalue	Get the value of an entry.
ini.keys	Get the list of entries in this section.
ini.read	Read data from a file.
ini.sections	Getting the list of sections.
ini.setnum	Write the numerical value of an entry.
ini.setvalue	Write the value of an entry.
ini.w rite	Save data into an ini file.

Functions

inigetval	Get the value of an entry from an ini file.
inisetval	Write the value of an entry into an ini file.

ini.delkey

Deleting a key.

```
method ini.delkey (
   str section,
   str key
)
```

Parameters

section	Section name.
key	The name of the entry being deleted.

Related links

- INI File

ini.delsection

Deleting a section.

```
method ini.delsection (
   str section
)
```

Parameters

section	The name of the section being deleted.

Related links

- INI File

ini.getnum

Get the numerical value of an entry.

```
method uint ini.getnum (
   str section,
   str key,
   uint defvalue
)
```

Parameters

section	Section name.
key	Key name.
defval	The value to be assigned if the entry is not found.

Return value

The numerical value of the key.

Related links

- INI File

ini.getvalue

Get the value of an entry.

```
method uint ini.getvalue (
   str section,
   str key,
   str value,
   str defvalue
)
```

Parameters

section	Section name.
key	Key name.
value	The string for getting the value.
defval	The value to be assigned if the entry is not found.

Return value

Returns 1 if the entry is found and 0 otherw ise.

Related links

- INI File

ini.keys

Get the list of entries in this section. All entries w ill be w ritten into an array of strings.

```
method arrstr ini.keys (
   str section,
   arrstr ret
)
```

Parameters

section	Section name.
ret	The array of strings the names of entries w ill be w ritten to.

Return value

Returns the parameter **ret**.

Related links

- INI File

ini.read

Read data from a file.

```
method ini.read (
   str filename
)
```

Parameters

filename	The name of the ini file.

Related links

- INI File

ini.sections

Getting the list of sections. All sections w ill be w ritten into an array of strings.

```
method arrstr ini.sections (
   arrstr ret
)
```

Parameters

ret	The array of strings the names of sections w ill be w ritten to.

Return value

Returns the parameter **ret**.

Related links

- INI File

ini.setnum

Write the numerical value of an entry.

```
method ini.setnum (
   str section,
   str key,
   uint value
)
```

Parameters

section	Section name.
key	Key name.
value	The value of the entry being w ritten.

Related links

- INI File

ini.setvalue

Write the value of an entry.

```
method ini.setvalue (
   str section,
   str key,
   str value
)
```

Parameters

section	Section name.
key	Key name.
value	The value of the entry being w ritten.

Related links

- INI File

ini.write

Save data into an ini file.

```
method uint ini.write (
   str filename
)
```

Parameters

filename	The name of the ini file.

Return value

Returns the size of the w ritten data.

Related links

- INI File

inigetval

Get the value of an entry from an ini file.

```
func str inigetval (
   str ininame,
   str section,
   str key,
   str value,
   str defval
)
```

Parameters

ininame	The name of the ini file.
section	Section name.
key	Key name.
value	The string for w riting the value.
defval	The value that w ill be inserted in case of an error or if there is not such an entry.

Return value

Returns the parameter **value**.

Related links

- INI File

inisetval

Write the value of an entry into an ini file.

```
func uint inisetval (
   str ininame,
   str section,
   str key,
   str value
)
```

Parameters

ininame	The name of the ini file.
section	Section name.
key	Key name.
value	The value of the entry being w ritten.

Return value

#lng\retf

Related links

- INI File

Keyboard

These functions are used to emulate the w ork of the keyboard. For using this library, it is required to specify the file keyboard.g (from lib\keyboard subfolder) w ith include command.

```
include : $"...\gentee\lib\keyboard\keyboard.g"
```

sendstr	Types a string on the keyboard.
sendvkey	Pressing a key.

sendstr

Types a string on the keyboard.

```
func uint sendstr (
   str input
)
```

Parameters

data	The string to be typed on the keyboard.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Keyboard

sendvkey

Pressing a key. Press a key alone or together w ith **Shift, Ctrl, Alt**.

```
func uint sendvkey (
   ushort vkey,
   uint flag
)
```

Parameters

vkey	Virtual key code.
flag	Flags for pressing additional keys.

$SVK_SHIFT	**Shift** is pressed.
$SVK_ALT	**Alt** is pressed.
$SVK_CONTROL	**Ctrl** is pressed.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Keyboard

Math

Mathematical functions.

abs	The absolute value for integers |x|.
acos	Calculating the arc cosine.
asin	Calculating the arc sine.
atan	Calculating the arc tangent.
ceil	Smallest double integer not less than given.
cos	Calculating the cosine.
exp	Exponential function.
fabs	The absolute value for double |x|.
floor	Largest double integer not greater than given.
ln	Natural logarithm.
log	Common logarithm.
modf	Splitting into whole and fractional parts.
pow	Raising to the power.
sin	Calculating the sine.
sqrt	Square root.
tan	Calculating the tangent.

abs

The absolute value for integers |x|.

```
func uint abs (
   int x
)
```

Parameters

x	An integer value.

Return value

The absolute value.

Related links

- Math

acos

Calculating the arc cosine.

```
func double acos (
   double x
)
```

Parameters

x	A value for calculating the arc cosine.

Return value

The arc cosine of x w ithin the range [0; PI].

Related links

- Math

asin

Calculating the arc sine.

```
func double asin (
   double x
)
```

Parameters

x	A value for calculating the arc sine.

Return value

The arc cosine of x w ithin the range [-PI/2 ; PI/2].

Related links

- Math

atan

Calculating the arc tangent.

```
func double atan (
   double x
)
```

Parameters

x	A value for calculating the arc tangent.

Return value

The arc tangent of x w ithin the range [-PI/2 ; PI/2].

Related links

- Math

ceil

Getting the smallest integer that is greater than or equal to x.

```
func double ceil (
   double x
)
```

Parameters

x	Floating-point value.

Return value

The closest least integer.

Related links

- Math

cos

Calculating the cosine.

```
func double cos (
   double x
)
```

Parameters

x	An angle in radians.

Return value

The cosine of x.

Related links

- Math

exp

Exponential function.

```
func double exp (
   double x
)
```

Parameters

x	A pow er for the number e.

Return value

The number e raised to the pow er of x.

Related links

- Math

fabs

The absolute value for double |x|.

```
func double fabs (
   double x
)
```

Parameters

x	Floating-point value.

Return value

The absolute value.

Related links

- Math

floor

Getting the largest integer that is less than or equal to x.

```
func double floor (
   double x
)
```

Parameters

x	Floating-point value.

Return value

The closest greatest integer.

Related links

- Math

ln

Natural logarithm.

```
func double ln (
   double x
)
```

Parameters

x	Floating-point value.

Return value

The natural logarithm ln(x).

Related links

- Math

log

Common logarithm.

```
func double log (
   double x
)
```

Parameters

x	Floating-point value.

Return value

The common logarithm log10(x).

Related links

- Math

modf

Splitting into w hole and fractional parts.

```
func double modf (
   double x,
   uint y
)
```

Parameters

x	Floating-point value.
y	A pointer to double for getting the w hole part.

Return value

The fractional part of x.

Related links

- Math

pow

Raising to the pow er.

```
func double pow (
   double x,
   double y
)
```

Parameters

x	A base.
y	A pow er.

Return value

Raising x to the pow er of y.

Related links

- Math

sin

Calculating the sine.

```
func double sin (
   double x
)
```

Parameters

x	An angle in radians.

Return value

The sine of x.

Related links

- Math

sqrt

Square root.

```
func double sqrt (
   double x
)
```

Parameters

x	A positive floating-point value.

Return value

The square root of x.

Related links

- Math

tan

Calculating the tangent.

```
func double tan (
   double x
)
```

Parameters

x	An angle in radians.

Return value

The tangent of x.

Related links

- Math

Memory

Gentee has ow n memory manager. This overview describes the memory management provided by Gentee. You can allocate and use memory w ith these functions.

malloc	Allocate the memory.
mcmp	Comparison memory.
mcopy	Copying memory.
mfree	Memory deallocation.
mlen	Size till zero.
mmove	Move memory.
mzero	Filling memory w ith zeros.

malloc

Allocate the memory. The function allocates the memory of the specified size.

```
func uint malloc (
   uint size
)
```

Parameters

size	The size of memory space to be allocated.

Return value

The pointer to the allocated memory space or 0 in case of an error.

Related links

- Memory

mcmp

Comparison memory. The function compares tw o memory spaces.

```
func int mcmp (
   uint dest,
   uint src,
   uint len
)
```

Parameters

dest	The pointer to the first memory space.
src	The pointer to the second memory space.
len	The size being compared.

Return value

0	The spaces are equal.
<0	The first space is smaller.
>0	The second space is smaller.

Related links

- Memory

mcopy

Copying memory. The function copies data from one memory space into another.

```
func uint mcopy (
   uint dest,
   uint src,
   uint len
)
```

Parameters

dest	The pointer for the data being copied.
src	The pointer to the source of the data being copied.
len	The size of the data being copied.

Return value

The pointer to the copied data.

Related links

- Memory

mfree

Memory deallocation. The function deallocates memory.

```
func uint mfree (
   uint ptr
)
```

Parameters

ptr	The pointer to the memory space to be deallocated.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Memory

mlen

Size till zero. Determines the number of bytes till zero.

```
func uint mlen (
   uint data
)
```

Parameters

data	The pointer to a memory space.

Return value

The number of bytes till the zero character.

Related links

- Memory

mmove

Move memory. The function moves the specified space. The initial and final data may overlap.

```
func mmove (
   uint dest,
   uint src,
   uint len
)
```

Parameters

dest	The pointer for the data being copied.
src	The pointer to the source of the data being copied.
len	The size of the data being copied.

Related links

- Memory

mzero

Filling memory w ith zeros. The functions zeroes the memory space.

```
func uint mzero (
   uint dest,
   uint len
)
```

Parameters

dest	The pointer to a memory space.
len	The size of the data being zeroed.

Return value

The pointer to the zeroed data.

Related links

- Memory

ODBC (SQL)

Data Access (SQL queries) Using ODBC. This library is applied for running SQL queries on a database using ODBC. The queries w ith parameters are not supported by the current version. Read ODBC description for more details. For using this library, it is required to specify the file odbc.g (from lib\odbc subfolder) w ith include command.

```
include : $"...\gentee\lib\odbc\odbc.g"
```

- Methods
- SQL query methods
- Field methods

ODBC description	A brief description of ODBC library.

Methods

odbc.connect	Create a database connection.
odbc.disconnect	Disconnect from a database.
odbc.geterror	Get the last error message.
odbc.newquery	Create a new ODBC query.

SQL query methods

odbcquery.active	Checks w hether a result set exists after the SQL query execution.
odbcquery.close	Close a result set.
odbcquery.fieldbyname	Find a field based on a specified field name.
odbcquery.first	Move the cursor to the first record in the result set.
odbcquery.geterror	Get the last error message.
odbcquery.getrecordcount	Get the total number of records in a result set.
odbcquery.last	Move the cursor to the last record in the result set.
odbcquery.moveby	Move the cursor to a position relative to its current position.
odbcquery.next	Move the cursor to the next record in the result set.
odbcquery.prior	Move the cursor to the prior record in the result set.
odbcquery.run	SQL query execution.
odbcquery.settimeout	Set query timeout.

Field methods

odbcfield.getbuf	Gets the field's value as a value of the buf type (the binary data).
odbcfield.getdatetime	Gets the field's value as a value of the datetime type.
odbcfield.getdouble	Gets the field's value as a floating-point number.
odbcfield.getindex	Gets the field index number.
odbcfield.getint	Gets the field's value as an integer.
odbcfield.getlong	Get the field's value as a number of the long type.
odbcfield.getname	Gets a field's name.
odbcfield.getnumeric	Gets the field's value as a fixed point number.
odbcfield.getstr	Get the field's value as a string of the **str** type.
odbcfield.gettype	Gets a type of the field's value.
odbcfield.isnull	Determines if the field contains the NULL value.

ODBC description

A brief description of ODBC library. The object of the **odbc** type provides connection to a database. The objects of the **odbcquery** type are used to run SQL queries and move the cursor through a result set. This object has got the **arr fields[] of odbcfield** array that contains result set fields **odbcfield**; furthermore, the number of elements of the array equals the number of the fields.

The objects of the **odbcfield** type make it possible to get the required information of the field as w ell as the field's value (depending on the current position of the cursor in the result set).

The sequence of operations for w orking w ith the database:

- create an ODBC connection to the database using the odbc.connect method;
- create a new ODBC query using the odbc.new query method. Note that severalqueries are likely to be created for one connection;
- run a SQL query using the odbcquery.run method; the query may retrieve the result set (the **SELECT** command) or no data (the **INSERT** command, the **UPDATE** command etc.);
- move the cursor through the result set using the follow ing methods: odbcquery.first, odbcquery.next etc. if necessary. The access is gained to the fields through the fields array **odbcquery.fields[i]**, w here i - a field number begining from 0, or w ith the odbcquery.fieldbyname method;
- use the odbcfield.getstr method, the odbcfield.getint method etc.in order to get field values;
- run the next SQL query after processing if necessary;
- disconnect from the database using the ODBC method odbc.disconnect.

There are some peculiarities to keep in mind w hen w orking w ith ODBC drivers:
w hile running a SQL query w ith the help of multiple sequential statements of the "INSERT ..." type, only some of the query statements are being executed (there can be from 300 to 1000 statements used for the "SQL server" driver) and no error message is displayed. In this case, you had better divide such queries into several parts;
some types of drivers do not make it possible to calculate the total number of messages received by the SQL query.

Related links

- ODBC (SQL)

odbc.connect

- method uint odbc.connect(str connectstr)
- method uint odbc.connect(str dsn, str user, str psw)

Create a database connection. You can connect to a database using a string connection or a DSN name.

The method is called in order to connect to the database w ith the help of the string connection. Use The ODBC connection string for this purpose, that contains a driver type, a database name and some additional parameters. The example below show s a type of the string connected to the SQL server: **"Driver={SQL Server};Server=MSSQLSERVER; Database=mydatabase;Trusted_Connection=yes;"**

```
method uint odbc.connect (
   str connectstr
)
```

Parameters

connectstr	Connection string.

Return value

Returns 1 if the connection is successful; otherw ise, returns 0.

---

odbc.connect

This method is used to connect to the database through the previously defined connection (the DSN name).

```
method uint odbc.connect (
   str dsn,
   str user,
   str psw
)
```

Parameters

dsn	Name of a previously defined connection - DSN.
user	User name.
psw	User passw ord.

Return value

Returns 1 if the connection is successful; otherw ise, returns 0.

Related links

- ODBC (SQL)

odbc.disconnect

Disconnect from a database.

```
method odbc.disconnect()
```

Related links

- ODBC (SQL)

odbc.geterror

Get the last error message. Gets the message if the last error occured w hile connecting to the database.

```
method uint odbc.geterror (
   str state,
   str message
)
```

Parameters

state	This string w ill contain the current state.
message	This string w ill contain an error message.

Return value

Returns the last error code.

Related links

- ODBC (SQL)

odbc.newquery

Create a new ODBC query. Creates a new ODBC query for the particular ODBC connection. Several queries are likely to be created for one connection. Queries are created inside the ODBC object and deleted in case of its deletion.

```
method odbcquery odbc.newquery()
```

Return value

A new ODBC query.

Related links

- ODBC (SQL)

odbcquery.active

Checks w hether a result set exists after the SQL query execution. If the SQL query of the **"SELECT ..."** type has been executed successfully, this method returns nonzero.

```
method uint odbcquery.active()
```

Return value

Returns nonzero if a result set exists.

Related links

- ODBC (SQL)

odbcquery.close

Close a result set. Closes a result set. This method is used after the SQL query of the **SELECT...** type has been executed. While calling the odbcquery.run method, the given method is automatically called.

```
method odbcquery.close()
```

Related links

- ODBC (SQL)

odbcquery.fieldbyname

Find a field based on a specified field name.

```
method odbcfield odbcquery.fieldbyname (
   str name
)
```

Parameters

name	Field name.

Return value

Returns the field or zero if fields w ith the same name are not found.

Related links

- ODBC (SQL)

odbcquery.first

Move the cursor to the first record in the result set.

```
method uint odbcquery.first()
```

Return value

If the cursor has been moved, it returns nonzero.

Related links

- ODBC (SQL)

odbcquery.geterror

Get the last error message. Gets the message if the last error occured w hile running the SQL query.

```
method uint odbcquery.geterror (
   str state,
   str message
)
```

Parameters

state	This string w ill contain the current state.
message	This string w ill contain an error message.

Return value

Returns the last error code.

Related links

- ODBC (SQL)

odbcquery.getrecordcount

Get the total number of records in a result set. Gets the total number of records in a result set w hen the SQL query of the **"SELECT ..."** type has been executed.

```
method uint odbcquery.getrecordcount()
```

Return value

Returns the the total number of records; if the total number of records is not determined, it returns -1.

Related links

- ODBC (SQL)

odbcquery.last

Move the cursor to the last record in the result set.

```
method uint odbcquery.last()
```

Return value

If the cursor has been moved, it returns nonzero.

Related links

- ODBC (SQL)

odbcquery.moveby

Move the cursor to a position relative to its current position.

```
method uint odbcquery.moveby (
   int off
)
```

Parameters

off	Indicates the number of records to move the cursor. If the number is negative, the cursor is moved backw ard.

Return value

If the cursor has been moved, it returns nonzero.

Related links

- ODBC (SQL)

odbcquery.next

Move the cursor to the next record in the result set.

```
method uint odbcquery.next()
```

Return value

If the cursor has been moved, it returns nonzero; otherw ise, it returns zero. If the current record is the last, it returns zero.

Related links

- ODBC (SQL)

odbcquery.prior

Move the cursor to the prior record in the result set.

```
method uint odbcquery.prior()
```

Return value

If the cursor has been moved, it returns nonzero.

Related links

- ODBC (SQL)

odbcquery.run

SQL query execution.

```
method uint odbcquery.run (
   str sqlstr
)
```

Parameters

sqlstr	String that contains the SQL query.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- ODBC (SQL)

odbcquery.settimeout

Set query timeout. Sets the number of seconds to w ait for a SQL query execution.

```
method odbcquery.settimeout (
   uint timeout
)
```

Parameters

timeout	The number of seconds to w ait for a SQL query execution. If it is equal to 0, then there is no timeout.

Related links

- ODBC (SQL)

odbcfield.getbuf

Gets the field's value as a value of the buf type (the binary data). This method is applied for fields w ith binary data.

```
method buf odbcfield.getbuf (
   buf dest
)
```

Parameters

dest	Result **buf** object.

Return value

Returns the parameter **dest**.

Related links

- ODBC (SQL)

odbcfield.getdatetime

Gets the field's value as a value of the datetime type. This method is applied for fields that contain date and/or time.

```
method datetime odbcfield.getdatetime (
   datetime dt
)
```

Parameters

dt	Result datetime object.

Return value

Returns the parameter **dt**.

Related links

- ODBC (SQL)

odbcfield.getdouble

Gets the field's value as a floating-point number. This method is applied for fields that contain floating-point numbers.

```
method double odbcfield.getdouble()
```

Return value

Returns the field's value.

Related links

- ODBC (SQL)

odbcfield.getindex

Gets the field index number.

```
method uint odbcfield.getindex()
```

Return value

Field index number.

Related links

- ODBC (SQL)

odbcfield.getint

- method int odbcfield.getint()
- method uint odbcfield.getuint()

Gets the field's value as an integer. This method is applied for fields that contain integers (the storage size is up to 4 bytes).

```
method int odbcfield.getint()
```

Return value

Returns the field's value.

---

odbcfield.getuint

Gets the field's value as an unsigned integer. This method is applied for fields that contain integers (the storage size is up to 4 bytes).

```
method uint odbcfield.getuint()
```

Return value

Returns the field's value.

Related links

- ODBC (SQL)

odbcfield.getlong

- method long odbcfield.getlong()
- method ulong odbcfield.getulong()

Get the field's value as a number of the long type. This method is applied for fields that contain long integers (8 bytes).

```
method long odbcfield.getlong()
```

Return value

Returns the field's value.

---

odbcfield.getulong

Get the field's value as a number of the **ulong** type. This method is applied for fields that contain long integers (8 bytes).

```
method ulong odbcfield.getulong()
```

Return value

Returns the field's value.

Related links

- ODBC (SQL)

odbcfield.getname

Gets a field's name.

```
method str odbcfield.getname (
   str result
)
```

Parameters

result	Result string.

Return value

Returns the parameter **result**.

Related links

- ODBC (SQL)

odbcfield.getnumeric

Gets the field's value as a fixed point number. This method is applied for fields that contain fixed point numbers. The structure is applied for data of this type, as follow s:

```
type numeric {
   long val
   uint scale
}
```

The **val** field contains the integer representation of a number, and the **scale** field indicates how many times val is divided by 10 in order to get a real number (a precision number).

```
method numeric odbcfield.getnumeric (
   numeric num
)
```

Parameters

num	Result numeric structure.

Return value

Returns the parameter **num**.

Related links

- ODBC (SQL)

odbcfield.getstr

Get the field's value as a string of the **str** type. This method is applied for fields that contain a string, a date, time and numeric fields.

```
method str odbcfield.getstr (
   str dest
)
```

Parameters

dest	Result **str** object.

Return value

Returns the parameter **dest**.

Related links

- ODBC (SQL)

odbcfield.gettype

Gets a type of the field's value. Returns the identifier of one of the following types: **buf, str, int, long, numeric, double, datetime**.

```
method uint odbcfield.gettype()
```

Return value

Type identifier.

Related links

- ODBC (SQL)

odbcfield.isnull

Determines if the field contains the NULL value.

```
method uint odbcfield.isnull()
```

Return value

Returns nonzero, if the field contains the NULL value; otherw ise, it returns zero.

Related links

- ODBC (SQL)

Process

Process, shell, arguments and environment functions.

argc	Get the number of parameters.
argv	Get a parameter.
exit	Exit the current program.
getenv	Get an environment variable.
process	Starting a process.
setenv	Set a value of an environment variable.
shell	Launch or open a file in the associated application.

argc

Get the number of parameters. The function returns the count of parameters in the command line.

```
func uint argc()
```

Return value

The number of parameters passed in the command line.

Related links

- Process

argv

Get a parameter. The function returns the parameter of the command line.

```
func str argv (
   str ret,
   uint num
)
```

Parameters

ret	A variable to w rite the return value to.
num	The number of the parameter to be obtained beginning from 1.

Return value

Returns the parameter **ret**.

Related links

- Process

exit

Exit the current program.

```
func exit (
   uint code
)
```

Parameters

code	A return code or the results of the w ork of the program.

Related links

- Process

getenv

Get an environment variable.

```
func str getenv (
   str varname,
   str ret
)
```

Parameters

varname	Environment variable name.
ret	String for getting the value.

Return value

Returns the parameter **ret**.

Related links

- Process

process

Starting a process.

```
func uint process (
   str cmdline,
   str workdir,
   uint result,
   uint state
)
```

Parameters

cmdline	The command line.
workdir	The w orking directory. It can be 0->str.
result	The pointer to uint for getting the result. If **0**, the function w ill not w ait until the process finishes its w ork.

Return value

1 if the calling process w as successful; otherw ise **0**.

Related links

- Process

setenv

Set a value of an environment variable. The function adds new environment variable or modifies the value of the existing environment variable. New values w ill be valid only in the current process.

```
func uint setenv (
   str varname,
   str varvalue
)
```

Parameters

varname	Environment variable name.
varvalue	A new value of the environment variable.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Process

shell

Launch or open a file in the associated application.

```
func shell (
   str name
)
```

Parameters

name	Filename.

Related links

- Process

Registry

Working w ith the Registry. This library allow s you to w ork w ith the Window s Registry. For using this library, it is required to specify the file registry.g (from lib\registry subfolder) w ith include command.

```
include : $"...\gentee\lib\registry\registry.g"
```

Functions

regdelkey	Deleting a registry key.
regdelvalue	Deleting the value of a key.
reggetmultistr	Getting a string sequence.
reggetnum	Get the numerical value of a registry key.
regkeys	Getting the list of keys.
regsetmultistr	Writing a string sequence.
regsetnum	Write a number as a registry key value.
regvaltype	Get the type of a registry key value.
regvalues	Getting the list of values in a key.
regverify	Creating missing keys.

Methods

buf.regget	Getting a value.
buf.regset	Writing a value.
str.regget	Getting a value.
str.regset	Write a string as a registry key value.

regdelkey

Deleting a registry key.

```
func uint regdelkey (
   uint root,
   str subkey
)
```

Parameters

root A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey The name of the registry key being deleted.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Registry

regdelvalue

Deleting the value of a key.

```
func uint regdelvalue (
   uint root,
   str subkey,
   str value
)
```

Parameters

root A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey A name of the registry key.

value The name of the value being deleted.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Registry

reggetmultistr

Getting a string sequence. Get the value of a registry key of the $REG_MULTISZ type into a string array.

```
func arrstr reggetmultistr (
   uint root,
   str subkey,
   str valname,
   arrstr val
)
```

Parameters

root — A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey — A name of the registry key.

valname — A name of the specified key value.

val — The array strings are w ritten to.

Return value

Returns the parameter **val**.

Related links

- Registry

reggetnum

- func uint reggetnum(uint root, str subkey, str valname)
- func uint reggetnum(uint root, str subkey, str valname, uint defval)

Get the numerical value of a registry key.

```
func uint reggetnum (
   uint root,
   str subkey,
   str valname
)
```

Parameters

root A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey A name of the registry key.

valname A name of the specified key value.

Return value

A numerical value is returned.

---

reggetnum

Get the numerical value of a registry key.

```
func uint reggetnum (
   uint root,
   str subkey,
   str valname,
   uint defval
)
```

Parameters

root A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey A name of the registry key.

valname A name of the specified key value.

defval The default number in case there is no value.

Return value

A numerical value is returned.

Related links

- Registry

regkeys

Getting the list of keys.

```
func uint regkeys (
   uint root,
   str subkey,
   arrstr ret
)
```

Parameters

root A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey A name of the registry key.

ret The array the names of the keys w ill be w ritten to.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Registry

regsetmultistr

Writing a string sequence. Write an array of strings as a value of a registry key of the $REG_MULTISZ type. If there is no key, it w ill be created.

```
func uint regsetmultistr (
   uint root,
   str subkey,
   str valname,
   arrstr val,
   arrstr ret
)
```

Parameters

root	A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey	A name of the registry key.
valname	The name of the value being w ritten.
val	The arrays of strings being w ritten.
ret	The array of strings all the created keys w ill be w ritten to. It can be 0.

Return value

0	No data has been w ritten.
1	The value of the key w as created during the w riting process.
2	Data is w ritten into the existing value.

Related links

- Registry

regsetnum

Write a number as a registry key value. If there is no key, it w ill be created.

```
func uint regsetnum (
   uint root,
   str subkey,
   str valname,
   uint value,
   arrstr ret
)
```

Parameters

root	A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey	A name of the registry key.
valname	The name of the value being w ritten.
value	The number being w ritten.
ret	The array of strings all the created keys w ill be w ritten to. It can be 0.

Return value

0	No data has been w ritten.
1	The value of the key w as created during the w riting process.
2	Data is w ritten into the existing value.

Related links

- Registry

regvaltype

Get the type of a registry key value.

```
func uint regvaltype (
   uint root,
   str subkey,
   str valname
)
```

Parameters

root A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey A name of the registry key.

valname The name of the key value the type of w hich is being determined.

Return value

0 is returned if the type is not determined or there is no such value. Besides, the follow ing values are possible:

$REG_NONE	Unknow n.
$REG_SZ	String.
$REG_EXPAND_SZ	Expanded string. String w ith environment variables.
$REG_BINARY	Binary data.
$REG_DWORD	Number.
$REG_MULTI_SZ	String sequence.

Related links

- Registry

regvalues

Getting the list of values in a key.

```
func uint regvalues (
   uint root,
   str subkey,
   arrstr ret
)
```

Parameters

root A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey A name of the registry key.

ret The array the names of values in the keys w ill be w ritten to.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Registry

regverify

Creating missing keys. Check if there is a certain key in the registry and create it if it is not there.

```
func uint regverify (
   uint root,
   str subkey,
   arrstr ret
)
```

Parameters

root A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey The name of the registry key being checked.

ret The array of strings all the created keys w ill be w ritten to. It can be 0.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Registry

buf.regget

Getting a value. This method w rites the value of a registry key into a Buffer object.

```
method buf buf.regget (
   uint root,
   str subkey,
   str valname,
   uint regtype
)
```

Parameters

root	A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey	A name of the registry key.
valname	A name of the specified key value.
regtype	The pointer to uint the type of this value w ill be w ritten to. It can be 0.

Return value

Returns the object w hich method has been called.

Related links

- Registry

buf.regset

Writing a value. Write the data of an buf object as registry key value. If there is no key, it w ill be created.

```
method uint buf.regset (
   uint root,
   str subkey,
   str valname,
   uint regtype,
   arrstr ret
)
```

Parameters

root A root key.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

subkey A name of the registry key.

valname The name of the value being w ritten.

regtype Value type.

$REG_NONE	Unknow n.
$REG_SZ	String.
$REG_EXPAND_SZ	Expanded string. String w ith environment variables.
$REG_BINARY	Binary data.
$REG_DWORD	Number.
$REG_MULTI_SZ	String sequence.

ret The array of strings all the created keys w ill be w ritten to. It can be 0.

Return value

0	No data has been w ritten.
1	The value of the key w as created during the w riting process.
2	Data is w ritten into the existing value.

Related links

- Registry

str.regget

- method str str.regget(uint root, str subkey, str valname)
- method str str.regget(uint root, str subkey, str valname, str defval)

Getting a value. This method w rites the value of a registry key into a String object.

```
method str str.regget (
   uint root,
   str subkey,
   str valname
)
```

Parameters

root	A root key.
subkey	A name of the registry key.
valname	A name of the specified key value.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

Return value

Returns the object w hich method has been called.

str.regget

This method w rites the value of a registry key into a String object.

```
method str str.regget (
   uint root,
   str subkey,
   str valname,
   str defval
)
```

Parameters

root	A root key.
subkey	A name of the registry key.
valname	A name of the specified key value.
defval	The default string in case there is no value.

$HKEY_CLASSES_ROOT	Classes Root.
$HKEY_CURRENT_USER	Current user's settings.
$HKEY_LOCAL_MACHINE	Local machine settings.
$HKEY_USERS	All users' settings

Return value

Returns the object w hich method has been called.

Related links

- Registry

str.regset

- method uint str.regset(uint root, str subkey, str valname, arrstr ret)
- method uint str.regset(uint root, str subkey, str valname)

Write a string as a registry key value. If there is no key, it w ill be created.

```
method uint str.regset (
   uint root,
   str subkey,
   str valname,
   arrstr ret
)
```

Parameters

root	A root key.
	$HKEY_CLASSES_ROOT — Classes Root.
	$HKEY_CURRENT_USER — Current user's settings.
	$HKEY_LOCAL_MACHINE — Local machine settings.
	$HKEY_USERS — All users' settings
subkey	A name of the registry key.
valname	The name of the value being w ritten.
ret	The array of strings all the created keys w ill be w ritten to. It can be 0.

Return value

0	No data has been w ritten.
1	The value of the key w as created during the w riting process.
2	Data is w ritten into the existing value.

str.regset

Write a string as a registry key value. If there is no key, it w ill be created.

```
method uint str.regset (
   uint root,
   str subkey,
   str valname
)
```

Parameters

root	A root key.
	$HKEY_CLASSES_ROOT — Classes Root.
	$HKEY_CURRENT_USER — Current user's settings.
	$HKEY_LOCAL_MACHINE — Local machine settings.
	$HKEY_USERS — All users' settings
subkey	A name of the registry key.
valname	The name of the value being w ritten.

Return value

0	No data has been w ritten.
1	The value of the key w as created during the w riting process.
2	Data is w ritten into the existing value.

.

Related links

- Registry

Socket

Sockets and common internet functions. You must call inet_init function before using this library. For using this library, it is required to specify the file internet.g (from lib\socket subfolder) w ith include command.

```
include : $"...\gentee\lib\socket\internet.g"
```

- Common internet functions
- Socket methods
- URL strings
- Types

Common internet functions

inet_close	Closing the library.
inet_error	Getting an error code.
inet_init	Library initialization.
inet_proxy	Using a proxy server.
inet_proxyenable	Enabling/disabling a proxy server.
inetnotify_func	Message handling function.

Socket methods

socket.close	Closes a socket.
socket.connect	Opens a socket.
socket.isproxy	Connecting via a proxy or not.
socket.recv	The method gets a packet from the connected server.
socket.send	The method sends a request to the connected server.
socket.urlconnect	Creating and connecting a socket to a URL.

URL strings

str.iencoding	Recoding a string.
str.ihead	Getting a header.
str.ihttpinfo	Processing a header.
str.iurl	The method is used to parse a URL address.

Types

httpinfo	HTTP header data.
inetnotify	Type for handling messages.
socket	Socket structure.

inet_close

Closing the library. This function must be called after the w ork w ith the library is finished.

```
func uint inet_close()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Socket

inet_error

Getting an error code. The function returns the code of the last error. Codes greater than 10000 are codes of errors in the library **WinSock 2** (w s2_32.dll).

```
func uint inet_error()
```

Return value

The code of the last error.

$ERRINET_DLLVERSION	Unsupported version of w s2_32.dll.
$ERRINET_HTTPDATA	Not HTTP data is received.
$ERRINET_USERBREAK	The process is interrupted by the user.
$ERRINET_OPENFILE	Cannot open the file.
$ERRINET_WRITEFILE	Cannot w rite the file.
$ERRINET_READFILE	Cannot read the file.
$ERRFTP_RESPONSE	The w rong response of the server.
$ERRFTP_QUIT	The w rong QUIT response of the server.
$ERRFTP_BADUSER	The bad user name.
$ERRFTP_BADPSW	The w rong passw ord.
$ERRFTP_PORT	Error PORT.

Related links

- Socket

inet_init

Library initialization. This function must be called before w orking w ith the library.

```
func uint inet_init()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Socket

inet_proxy

Using a proxy server. The functions allow s you to specify a proxy server to be used for connecting to the Internet.

```
func uint inet_proxy (
   uint flag,
   str proxyname
)
```

Parameters

flag The flag specifying for w hich protocols the specified proxy should be used.

$PROXY_HTTP	Use a proxy server for the HTTP protocol.
$PROXY_FTP	Use a proxy server for the FTP protocol.
$PROXY_ALL	Use a proxy server for all protocols.
$PROXY_EXPLORER	Take the proxy server information from the Internet Explorer settings. In this case the proxyname parameter can be empty.

proxyname The name of the proxy server. It must contain a host name and a port number separated by a colon.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Socket

inet_proxyenable

Enabling/disabling a proxy server. The function allow s you to enable or disable the proxy server for various protocols. Initially, the proxy server must be specified using the inet_proxy function.

```
func uint inet_proxyenable (
   uint flag,
   uint enable
)
```

Parameters

flag The flag specifying for w hich protocols the proxy should be enabled or disabled.

$PROXY_HTTP	Use a proxy server for the HTTP protocol.
$PROXY_FTP	Use a proxy server for the FTP protocol.
$PROXY_ALL	Use a proxy server for all protocols.
$PROXY_EXPLORER	Take the proxy server information from the Internet Explorer settings. In this case the proxyname parameter can be empty.

enable Specify 1 to enable the proxy server or 0 to disable the proxy server.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Socket

inetnotify_func

Message handling function. When some functions are called, you can specify a function for handing incoming notifications. In particular, it allow s you to show the w orking process to the user. This handling function must have the follow ing parameters.

```
func uint inetnotify_func (
   uint code,
   inetnotify ni
)
```

Parameters

code Message code.

$NFYINET_ERROR	An error occurred. The code of the error can be got w ith the help of the inet_error function.
$NFYINET_CONNECT	Server connection.
$NFYINET_SEND	Sending a request.
$NFYINET_POST	Sending data.
$NFYINET_HEAD	Processing the header. ni.param points to httpinfo.
$NFYINET_REDIRECT	Request redirection. ni.sparam contains the new URL.
$NFYINET_GET	Data is received. ni.param contains the total size of all data.
$NFYINET_PUT	Data is sent. ni.param contains the total size of all data.
$NFYINET_END	The connection is terminated.
$NFYFTP_RESPONSE	Response of the FTP server. The field ni.head contains it.
$NFYFTP_SENDCMD	Sending a command to the FTP server. The field ni.head contains it.
$NFYFTP_NOTPASV	Passive mode w ith the FTP server is unavailable.

ni The variable of the inetnotify type w ith additional data.

Return value

The function must return 1 to continue w orking and 0 otherw ise.

Related links

- Socket

socket.close

Closes a socket.

```
method uint socket.close()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Socket

socket.connect

Opens a socket. The method creates a socket and establishes a connection to the **host** and **port** specified in the host and port fields of the socket structure.

```
method uint socket.connect()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Socket

socket.isproxy

Connecting via a proxy or not. This method can be used to determine if a socket is connected via a proxy server or not.

```
method uint socket.isproxy()
```

Return value

1 is returned if the socket is connected via a proxy server and 0 is returned otherw ise.

Related links

- Socket

socket.recv

The method gets a packet from the connected server.

```
method uint socket.recv (
   buf data
)
```

Parameters

data	The buffer for w riting data. The received packet w ill be added to the data already existing in the buffer.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Socket

socket.send

- method uint socket.send(str data)
- method uint socket.send(buf data)

The method sends a request to the connected server.

```
method uint socket.send (
   str data
)
```

Parameters

data	Request string.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

---

socket.send

The method sends a request data to the connected server.

```
method uint socket.send (
   buf data
)
```

Parameters

data	Request buffer.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Socket

socket.urlconnect

Creating and connecting a socket to a URL. The method is used to create and connect a socket to the specified Internet address. If a proxy server is enabled, the connection w ill be established via it.

```
method uint socket.urlconnect (
   str url,
   str host,
   str path
)
```

Parameters

url	The URL address for connecting.
host	The string for getting the host from the URL.
path	The string for getting the relative path from the URL.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Socket

str.iencoding

Recoding a string. The method recodes the specified string in order to send it using the POST method. Spaces are replaced w ith '+', special characters are replaced w ith their hexadecimal representations **%XX**. The result w ill be w ritten to the string for w hich the method w as called.

```
method str str.iencoding (
   str src
)
```

Parameters

src	The string for recoding.

Return value

Returns the object w hich method has been called.

Related links

- Socket

str.ihead

Getting a header. The method is used to get the message header. It w ill be w ritten to the string for w hich the method w as called. Besides, the header w ill be deleted from the data object.

```
method str str.ihead (
   buf data
)
```

Parameters

data	The buffer of the string containing the data being processed.

Return value

Returns the object w hich method has been called.

Related links

- Socket

str.ihttpinfo

Processing a header. The method processes a string as an HTTP header and w rites data it gets into the httpinfo structure.

```
{
```

Parameters

hi	The variable of the httpinfo type for getting the results.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Socket

str.iurl

The method is used to parse a URL address.

```
method uint str.iurl (
   str host,
   str port,
   str path
)
```

Parameters

host	The string for getting the host name.
port	The string for getting the port.
path	The string for getting the relative path.

Return value

1 is returned if the FTP protocol w as specified. Otherw ise, 0 is returned.

Related links

- Socket

httpinfo

HTTP header data. The structure is used to get data from an HTTP header. Depending on the header, some fields may be empty.

```
type httpinfo
{
   uint code
   datetime dt
   str size
   str location
}
```

Members

code	Message code.
dt	Last modified date.
size	File size.
location	New file location.

Related links

- Socket

inetnotify

Type for handling messages. This structure is passed to the message handling function as a parameter. Additional parameters take various values depending on the message code.

```
type inetnotify
{
   str url
   str head
   uint param
   str sparam
}
```

Members

url	The URL address being processed.
head	The header of the received packet.
param	Additional integer parameter.
sparam	Additional string parameter.

Related links

- Socket

socket

Socket structure.

```
type socket
{
   str host
   ushort port
   uint socket
   uint flag
}
```

Members

host	Host name.
port	Port number.
socket	Open socket identifier.
flag	Additional flags. **$SOCKF_PROXY** - The socket is opened via a proxy server.

Related links

- Socket

Stack

Stack. You can use variables of the **stack** type for w orking w ith stacks. The **stack** type is inherited from the **arr** type. So, you can also use methods of the arr type.

- Methods
- Type

Methods

stack.pop	Extracting an item.
stack.popval	Extracting an number.
stack.push	Add an item to a stack.
stack.top	Get the top item in a stack.

Type

stack	The main structure of the stack.

stack.pop

- method uint stack.pop
- method str stack.pop(str val)

Extracting an item. The method deletes the top item from a stack.

```
method uint stack.pop
```

Return value

The pointer to the next new top item.

---

stack.pop

The method extracts a string from a stack. The stack must be described as **stack of str**.

```
method str stack.pop (
   str val
)
```

Parameters

val	Result string.

Return value

Returns the parameter **val**.

Related links

- Stack

stack.popval

Extracting an number. The method extracts a number from a stack.

```
method uint stack.popval
```

Return value

The number extracted from the stack is returned.

Related links

- Stack

stack.push

- method uint stack.push
- method uint stack.push(uint val)
- method str stack.push(str val)

Add an item to a stack.

```
method uint stack.push
```

Return value

The pointer to the added item.

---

stack.push

The method adds a number to a stack.

```
method uint stack.push (
   uint val
)
```

Parameters

val	Pushing uint or int number.

Return value

The added value is returned.

---

stack.push

The method adds a string to a stack. The stack must be described as **stack of str**.

```
method str stack.push (
   str val
)
```

Parameters

val	Pushing string.

Return value

The added string is returned.

Related links

- Stack

stack.top

Get the top item in a stack.

```
method uint stack.top
```

Return value

The pointer to the top item.

Related links

- Stack

stack

The main structure of the stack.

```
type stack <inherit = arr>
{
}
```

Related links

- Stack

String

Strings. It is possible to use variables of the **str** type for w orking w ith strings. The **str** type is inherited from the **buf** type. So, you can also use methods of the buf type.

- Operators
- Methods
- Search methods

Operators

*** str**	Get the length of a string.
str + str	Putting tw o strings together and creating a resulting string.
str = str	Copy the string.
str += type	Appending types to the string.
str == str	Comparison operation.
str < str	Comparison operation.
str > str	Comparison operation.
str(type)	Converting types to str.
type(str)	Converting string to other types.

Methods

str.append	Data addition.
str.appendch	Adding a character to a string.
str.clear	Clearing a string.
str.copy...	Copying.
str.crc	Calculating the checksum.
str.del	Delete a substring.
str.dellast	Delete the last character.
str.eqlen...	Comparison.
str.fill...	Filling a string.
str.find...	Find the character in the string.
str.hex...	Converting an unsigned integer in the hexadecimal form.
str.insert	Insertion.
str.islast	Check the final character.
str.lines	Convert a multi-line string to an array of strings.
str.lower	Converting to low ercase.
str.out4	Output a 32-bit value.
str.print	Print a string into the console w indow .
str.printf	Write formatted data to a string.
str.read	Read a string from a file.
str.repeat	Repeating a string.
str.replace	Replacing in a string.

str.replacech	Replace a character.
str.setlen	Setting a new string size.
str.split	Splitting a string.
str.substr	Getting a substring.
str.trim ...	Trimming a string.
str.upper	Converting to uppercase.
str.write	Writing a string to a file.
str.writeappend	Appending string to a file.

Search methods

spattern	The pattern structure for the searching.
spattern.init	Creating data search pattern.
spattern.search	Search a pattern in another string.
str.search	Substring search.

* str

Get the length of a string.

```
operator uint * (
   str left
)
```

Return value

The length of the string.

Related links

- String

str + str

Putting tw o strings together and creating a resulting string.

```
operator str +<result> (
   str left,
   str right
)
```

Return value

The new result string.

Related links

- String

str = str

Copy the string.

```
operator str = (
   str left,
   str right
)
```

Return value

The result string.

Related links

- String

str += type

- operator str +=(str left, str right)
- operator str +=(str left, uint right)
- operator str +=(str left, int val)
- operator str +=(str left, float val)
- operator str +=(str left, long val)
- operator str +=(str left, ulong val)
- operator str +=(str left, double val)

Appending types to the string. Append **str** to **str** => **str += str**.

```
operator str += (
   str left,
   str right
)
```

Return value

The result string.

---

str += uint

Append **uint** to **str** => **str += uint**.

```
operator str += (
   str left,
   uint right
)
```

str += int

Append **int** to **str** => **str += int**.

```
operator str += (
   str left,
   int val
)
```

str += float

Append **float** to **str** => **str += float**.

```
operator str += (
   str left,
   float val
)
```

str += long

Append **long** to **str** => **str += long**.

```
operator str += (
   str left,
   long val
)
```

str += ulong

Append **ulong** to **str** => **str += ulong**.

```
operator str += (
   str left,
   ulong val
)
```

str += double

Append **double** to **str** => **str += double**.

```
operator str += (
   str left,
   double val
)
```

Related links

- String

str == str

- operator uint ==(str left, str right)
- operator uint !=(str left, str right)
- operator uint %==(str left, str right)
- operator uint %!=(str left, str right)

Comparison operation.

```
operator uint == (
   str left,
   str right
)
```

Return value

Returns **1** if the strings are equal. Otherw ise, it returns **0**.

str != str

Comparison operation.

```
operator uint != (
   str left,
   str right
)
```

Return value

Returns **0** if the strings are equal. Otherw ise, it returns **1**.

str %== str

Comparison operation w ith ignore case.

```
operator uint %== (
   str left,
   str right
)
```

Return value

Returns **1** if the strings are equal w ith ignore case. Otherw ise, it returns **0**.

str %!= str

Comparison operation w ith ignore case.

```
operator uint %!= (
   str left,
   str right
)
```

Return value

Returns **0** if the strings are equal w ith ignore case. Otherw ise, it returns **1**.

Related links

- String

str < str

- operator uint <(str left, str right)
- operator uint <=(str left, str right)
- operator uint %<(str left, str right)
- operator uint %<=(str left, str right)

Comparison operation.

```
operator uint < (
   str left,
   str right
)
```

Return value

Returns **1** if the first string is less than the second one. Otherw ise, it returns **0**.

---

str <= str

Comparison operation.

```
operator uint <= (
   str left,
   str right
)
```

Return value

Returns **1** if the first string is less or equal the second one. Otherw ise, it returns **0**.

---

str %< str

Comparison operation w ith ignore case.

```
operator uint %< (
   str left,
   str right
)
```

Return value

Returns **1** if the first string is less than the second one w ith ignore case. Otherw ise, it returns **0**.

---

str %<= str

Comparison operation w ith ignore case.

```
operator uint %<= (
   str left,
   str right
)
```

Return value

Returns **1** if the first string is less or equal the second one w ith ignore case. Otherw ise, it returns **0**.

Related links

- String

str > str

- operator uint >(str left, str right)
- operator uint >=(str left, str right)
- operator uint %>(str left, str right)
- operator uint %>=(str left, str right)

Comparison operation.

```
operator uint > (
   str left,
   str right
)
```

Return value

Returns **1** if the first string is greater than the second one. Otherw ise, it returns **0**.

---

str >= str

Comparison operation.

```
operator uint >= (
   str left,
   str right
)
```

Return value

Returns **1** if the first string is greater or equal the second one. Otherw ise, it returns **0**.

---

str %> str

Comparison operation w ith ignore case.

```
operator uint %> (
   str left,
   str right
)
```

Return value

Returns **1** if the first string is greater than the second one w ith ignore case. Otherw ise, it returns **0**.

---

str %>= str

Comparison operation w ith ignore case.

```
operator uint %>= (
   str left,
   str right
)
```

Return value

Returns **1** if the first string is greater or equal the second one w ith ignore case. Otherw ise, it returns **0**.

Related links

- String

str(type)

- method str int.str < result >
- method str uint.str < result >
- method str float.str <result>
- method str long.str <result>
- method str ulong.str<result>
- method str double.str <result>

Converting types to str. Convert **int** to **str** => **str(int)**.

```
method str int.str < result >
```

Return value

The result string.

---

str(uint)

Convert **uint** to **str** => **str(uint)**.

```
method str uint.str < result >
```

---

str(float)

Convert **float** to **str** => **str(float)**.

```
method str float.str <result>
```

---

str(long)

Convert **long** to **str** => **str(long)**.

```
method str long.str <result>
```

---

str(ulong)

Convert **ulong** to **str** => **str(ulong)**.

```
method str ulong.str<result>
```

---

str(double)

Convert **double** to **str** => **str(double)**.

```
method str double.str <result>
```

Related links

- String

type(str)

- method int str.int
- method uint str.uint
- method float str.float
- method long str.long
- method double str.double

Converting string to other types. Convert **str** to **int** => **int(str)**.

```
method int str.int
```

Return value

The result value of the according type.

---

uint(str)

Convert **str** to **uint** => **uint(str)**.

```
method uint str.uint
```

---

float(str)

Convert **str** to **float** => **float(str)**.

```
method float str.float
```

---

long(str)

Convert **str** to **long** => **long(str)**.

```
method long str.long
```

---

double(str)

Convert **str** to **double** => **double(str)**.

```
method double str.double
```

Related links

- String

str.append

Data addition. Add data to a string.

```
method str str.append (
   uint src,
   uint size
)
```

Parameters

src	The pointer to the data to be added.
size	The size of the data being added.

Return value

Returns the object w hich method has been called.

Related links

- String

str.appendch

Adding a character to a string.

```
method str str.appendch (
   uint ch
)
```

Parameters

ch	The character to be added.

Return value

Returns the object w hich method has been called.

Related links

- String

str.clear

Clearing a string.

```
method str str.clear()
```

Return value

Returns the object w hich method has been called.

Related links

- String

str.copy...

- method str str.copy(uint ptr)
- method str str.load(uint ptr, uint len)

Copying. The method copies data into a string.

```
method str str.copy (
   uint ptr
)
```

Parameters

ptr	The pointer to the data being copied. All data to the zero character w ill be copied.

Return value

Returns the object w hich method has been called.

---

str.load

The method copies data into a string.

```
method str str.load (
   uint ptr,
   uint len
)
```

Parameters

src	The pointer to the data being copied. If data does not end in a zero, it w ill be added automatically.
size	The size of the data being copied.

Return value

Returns the object w hich method has been called.

Related links

- String

str.crc

Calculating the checksum. The method calculates the checksum of a string.

```
method uint str.crc()
```

Return value

The string checksum is returned.

Related links

- String

str.del

Delete a substring.

```
method str str.del (
   uint off,
   uint len
)
```

Parameters

off	The offset of the substring being deleted.
len	The size of the substring being deleted.

Return value

Returns the object w hich method has been called.

Related links

- String

str.dellast

Delete the last character. The method deletes the last character if it is equal the specified parameter.

```
method str str.dellast (
   uint ch
)
```

Parameters

ch	A character to be checked.

Return value

Returns the object w hich method has been called.

Related links

- String

str.eqlen...

- method uint str.eqlen(uint ptr)
- method uint str.eqlenign(uint ptr)
- method uint str.eqlen(str src)
- method uint str.eqlenign(str src)

Comparison. Compare a string w ith the specified data. The comparison is carried out only at the length of the string the method is called for.

```
method uint str.eqlen (
   uint ptr
)
```

Parameters

ptr	The pointer to the data to be compared.

Return value

Returns 1 if there is an equality and 0 otherw ise.

---

str.eqlenign

Compare a string w ith the specified data. The comparison is carried out only at the length of the string the method is called for.

```
method uint str.eqlenign (
   uint ptr
)
```

Parameters

ptr	The pointer to the data to be compared. The comparison is case-insensitive.

---

str.eqlen

Compare a string w ith the specified string. The comparison is carried out only at the length of the string the method is called for.

```
method uint str.eqlen (
   str src
)
```

Parameters

src	The string to be compared.

---

str.eqlenign

Compare a string w ith the specified string. The comparison is carried out only at the length of the string the method is called for.

```
method uint str.eqlenign (
   str src
)
```

Parameters

src	The string to be compared. The comparison is case-insensitive.

Related links

- String

str.fill...

- method str str.fill(str val, uint count, uint flag)
- method str str.fillspacel(uint len)
- method str str.fillspacer(uint len)

Filling a string. Fill a string to the left or to the right.

```
method str str.fill (
   str val,
   uint count,
   uint flag
)
```

Parameters

val	The string that w ill be filled.
count	The number of additions.
flag	Flags.

$FILL_LEFT	Filling on the left side.
$FILL_LEN	The count parameter contains the final string size.
$FILL_CUT	Cut if longer than the final size. Used together w ith FILL_LEN.

Return value

Returns the object w hich method has been called.

str.fillspacel

Fill a string w ith spaces to the left.

```
method str str.fillspacel (
   uint len
)
```

Parameters

len	Final string size.

str.fillspacer

Fill a string w ith spaces to the right.

```
method str str.fillspacer (
   uint len
)
```

Parameters

len	Final string size.

Related links

- String

str.find...

- method uint str.findch(uint offset, uint symbol, uint fromend)
- method uint str.findch(uint symbol)
- method uint str.findchr(uint symbol)
- method uint str.findchfrom(uint symbol, uint offset)
- method uint str.findchnum(uint symbol, uint i)

Find the character in the string.

```
method uint str.findch (
   uint offset,
   uint symbol,
   uint fromend
)
```

Parameters

offset	The offset to start searching from.
symbol	Search character.
fromend	If it equals 1, the search w ill be carried out from the end of the string.

Return value

The offset of the character if it is found. If the character is not found, the length of the string is returned.

str.findch

Find the character from the beginning of the string.

```
method uint str.findch (
   uint symbol
)
```

Parameters

symbol	Search character.

str.findchr

Find the character from the end of the string.

```
method uint str.findchr (
   uint symbol
)
```

Parameters

symbol	Search character.

str.findchfrom

Find the character from the specified offset in the string.

```
method uint str.findchfrom (
   uint symbol,
   uint offset
)
```

Parameters

symbol	Search character.
offset	The offset to start searching from.

str.findchnum

Find the **#glt(i)** character in the string.

```
method uint str.findchnum (
   uint symbol,
   uint i
)
```

Parameters

symbol	Search character.
i	The number of the character starting from 1.

Related links

- String

str.hex...

- method str str.hexl(uint val)
- method str str.hexu(uint val)
- func str hex2strl<result>(uint val)
- func str hex2stru<result>(uint val)

Converting an unsigned integer in the hexadecimal form. Low er characters.

```
method str str.hexl (
   uint val
)
```

Parameters

val	The unsigned integer value to be converted into the string.

Return value

Returns the object w hich method has been called.

---

str.hexu

Converting an unsigned integer in the hexadecimal form. (upper characters).

```
method str str.hexu (
   uint val
)
```

Parameters

val	The unsigned integer value to be converted into the string.

Return value

Returns the object w hich method has been called.

---

hex2strl

Converting an unsigned integer in the hexadecimal form. (low er characters).

```
func str hex2strl<result> (
   uint val
)
```

Parameters

val	The unsigned integer value to be converted into the string.

Return value

The new result string.

---

hex2stru

Converting an unsigned integer in the hexadecimal form. (upper characters).

```
func str hex2stru<result> (
   uint val
)
```

Parameters

val	The unsigned integer value to be converted into the string.

Return value

The new result string.

Related links

- String

str.insert

Insertion. The method inserts one string into another.

```
method str str.insert (
   uint offset,
   str value
)
```

Parameters

offset	The offset w here string w ill be inserted.
value	The string being inserted.

Return value

Returns the object w hich method has been called.

Related links

- String

str.islast

Check the final character.

```
method uint str.islast (
   uint symbol
)
```

Parameters

symbol	The character being checked.

Return value

Returns 1 if the last character in the string coincides w ith the specified one and 0 otherw ise.

Related links

- String

str.lines

- method arrstr str.lines(arrstr ret, uint trim, arr offset)
- method arrstr str.lines(arrstr ret, uint trim)
- method arrstr str.lines<result>(uint trim)

Convert a multi-line string to an array of strings.

```
method arrstr str.lines (
   arrstr ret,
   uint trim,
   arr offset
)
```

Parameters

ret	The result array of strings.
trim	Specify 1 if you w ant to trim all characters less or equal space in lines.
offset	The array for getting offsets of lines in the string. It can be 0->>arr.

Return value

The result array of strings.

---

str.lines

Convert a multi-line string to an array of strings.

```
method arrstr str.lines (
   arrstr ret,
   uint trim
)
```

Parameters

ret	The result array of strings.
trim	Specify 1 if you w ant to trim all characters less or equal space in lines.

---

str.lines

Convert a multi-line string to an array of strings.

```
method arrstr str.lines<result> (
   uint trim
)
```

Parameters

trim	Specify 1 if you w ant to trim all characters less or equal space in lines.

Return value

The new result array of strings.

Related links

- String

str.lower

- method str str.low er()
- method str str.low er(uint off, uint size)

Converting to low ercase. The method converts characters in a string to low ercase.

```
method str str.lower()
```

Return value

Returns the object w hich method has been called.

---

str.lower

Convert a substring in the specified string to low ercase.

```
method str str.lower (
   uint off,
   uint size
)
```

Parameters

off	Substring offset.
size	Substring size.

Related links

- String

str.out4

- method str str.out4(str format, uint val)
- method str str.out8(str format, ulong val)

Output a 32-bit value. The value is appended at the end of the string.

```
method str str.out4 (
   str format,
   uint val
)
```

Parameters

format	The format of the output. It is the same as in the function 'printf' in C programming language.
val	32-bit value to be appended.

Return value

Returns the object w hich method has been called.

---

str.out8

Output a 64-bit value. The value is appended at the end of the string.

```
method str str.out8 (
   str format,
   ulong val
)
```

Parameters

format	The format of the output. It is the same as in the function 'printf' in C programming language.
val	64-bit value to be appended.

Return value

Returns the object w hich method has been called.

Related links

- String

str.print

- method str.print()
- func print(str output)

Print a string into the console w indow .

```
method str.print()
```

print

Print a string into the console w indow .

```
func print (
   str output
)
```

Parameters

output	The output string.

Related links

- String

str.printf

Write formatted data to a string. The method formats and stores a series of characters and values in string. Each argument is converted and output according to the corresponding C/C++ format specification (printf) in format parameter.

```
method str str.printf (
   str format,
   collection clt
)
```

Parameters

format	The format of the output.
clt	Optional arguments.

Return value

Returns the object w hich method has been called.

Related links

- String

str.read

Read a string from a file.

```
method uint str.read (
   str filename
)
```

Parameters

filename	Filename.

Return value

The size of the read data.

Related links

- String

str.repeat

Repeating a string. Repeat a string the specified number of times.

```
method str str.repeat (
   uint count
)
```

Parameters

count	The number of repeatitions. The result w ill be w ritten into this very string.

Return value

Returns the object w hich method has been called.

Related links

- String

str.replace

- method str str.replace(uint offset, uint size, str value)
- method str str.replace(arrstr aold, arrstr anew , uint flags)
- method str str.replace(str sold, str snew , uint flags)

Replacing in a string. The method replaces data in a string.

```
method str str.replace (
   uint offset,
   uint size,
   str value
)
```

Parameters

offset	The offset of the data being replaced.
size	The size of the data being replaced.
value	The string being inserted.

Return value

Returns the object w hich method has been called.

---

str.replace

The method looks for strings from one array and replace to strings of another array.

```
method str str.replace (
   arrstr aold,
   arrstr anew,
   uint flags
)
```

Parameters

aold	The strings to be replaced.
anew	The new strings.
flags	Flags.

$QS_IGNCASE	Case-insensitive search.
$QS_WORD	Search the w hole w ord only.
$QS_BEGINWORD	Search w ords w hich start w ith the specified pattern.

Return value

Returns the object w hich method has been called.

---

str.replace

The method replaces one string to another string in the source string.

```
method str str.replace (
   str sold,
   str snew,
   uint flags
)
```

Parameters

sold	The string to be replaced.
snew	The new string.
flags	Flags.

$QS_IGNCASE	Case-insensitive search.
$QS_WORD	Search the w hole w ord only.
$QS_BEGINWORD	Search w ords w hich start w ith the specified pattern.

Return value

Returns the object w hich method has been called.

Related links

- String

str.replacech

Replace a character. The method copies a source string w ith the replacing a character to a string.

```
method str str.replacech (
   str src,
   uint from,
   str to
)
```

Parameters

src	Initial string.
from	A character to be replaced.
to	A string for replacing.

Return value

Returns the object w hich method has been called.

Related links

- String

str.setlen

- method str str.setlen(uint len)
- method str str.setlenptr()

Setting a new string size. The method does not reserve space. You cannot specify the size of a string greater than the reserved space you have. Mostly, this function is used for specifying the size of a string after external functions w rite data to it.

```
method str str.setlen (
   uint len
)
```

Parameters

len	New string size.

Return value

Returns the object w hich method has been called.

---

str.setlenptr

Recalculate the size of a string to the zero character. The function can be used to determine the size of a string after other functions w rite data into it.

```
method str str.setlenptr()
```

Related links

- String

str.split

- method arrstr str.split(arrstr ret, uint symbol, uint flag)
- method arrstr str.split <result> (uint symbol, uint flag)
- method uint str.split(uint symbol, str left, str right)

Splitting a string. The method splits a string into substrings taking into account the specified separator.

```
method arrstr str.split (
   arrstr ret,
   uint symbol,
   uint flag
)
```

Parameters

ret	The result array of strings.
symbol	Separator.
flag	Flags.

$SPLIT_EMPTY	Take into account empty substrings.
$SPLIT_NOSYS	Delete characters <= space on the left and on the right.
$SPLIT_FIRST	Split till the first separator.
$SPLIT_QUOTE	Take into account that elements can be enclosed by single or double quotation marks.
$SPLIT_APPEND	Adding strings. Otherw ise, the array is cleared before loading.

Return value

The result array of strings.

The method splits a string into the new result array of strings.

```
method arrstr str.split <result>  (
   uint symbol,
   uint flag
)
```

Parameters

symbol	Separator.
flag	Flags.

$SPLIT_EMPTY	Take into account empty substrings.
$SPLIT_NOSYS	Delete characters <= space on the left and on the right.
$SPLIT_FIRST	Split till the first separator.
$SPLIT_QUOTE	Take into account that elements can be enclosed by single or double quotation marks.
$SPLIT_APPEND	Adding strings. Otherw ise, the array is cleared before loading.

Return value

The new result array of strings.

---

str.split

The method looks for the first *symbol* and splits a string into tw o parts.

```
method uint str.split (
   uint symbol,
   str left,
   str right
)
```

Parameters

symbol	Separator.
left	The substring left on the *symbol*.
right	The substring right on the *symbol*.

Return value

Returns 1 if the separator has been found. Otherw ise, return 0.

Related links

- String

str.substr

- method str str.substr(str src, uint off, uint len)
- method str str.substr(uint off, uint len)

Getting a substring.

```
method str str.substr (
   str src,
   uint off,
   uint len
)
```

Parameters

src	Initial string.
off	Substring offset.
len	Substring size.

Return value

Returns the object w hich method has been called.

---

str.substr

Get a substring. The result substring w ill be w ritten over the existing string.

```
method str str.substr (
   uint off,
   uint len
)
```

Parameters

off	Substring offset.
len	Substring size.

Related links

- String

str.trim...

- method str str.trimsys()
- method str str.trimrsys()
- method str str.trim(uint symbol, uint flag)
- method str str.trimrspace()
- method str str.trimspace()

Trimming a string. Deleting spaces and special characters on both sides.

```
method str str.trimsys()
```

Return value

Returns the object w hich method has been called.

---

str.trimrsys

Deleting spaces and special characters on the right.

```
method str str.trimrsys()
```

---

str.trim

Delete the specified character on either sides of a string.

```
method str str.trim (
   uint symbol,
   uint flag
)
```

Parameters

symbol	The character being deleted.
flag	Flags.

$TRIM_LEFT	Trim the left side.
$TRIM_RIGHT	Trim the right side.
$TRIM_ONE	Delete only one character.
$TRIM_PAIR	If the character being deleted is a bracket, look the closing bracket on the right
$TRIM_SYS	Delete characters less or equal space.

---

str.trimrspace

Deleting spaces on the right.

```
method str str.trimrspace()
```

---

str.trimspace

Deleting spaces on the both sides.

```
method str str.trimspace()
```

Related links

- String

str.upper

- method str str.upper()
- method str str.upper(uint off, uint size)

Converting to uppercase. The method converts characters in a string to uppercase.

```
method str str.upper()
```

Return value

Returns the object w hich method has been called.

---

str.upper

Convert a substring in the specified string to uppercase.

```
method str str.upper (
   uint off,
   uint size
)
```

Parameters

off	Substring offset.
size	Substring size.

Related links

- String

str.write

Writing a string to a file.

```
method uint str.write (
   str filename
)
```

Parameters

filename	The name of the file for w riting. If the file already exists, it w ill be overw ritten.

Return value

The size of the w ritten data.

Related links

- String

str.writeappend

Appending string to a file. The method appends a string to the specified file.

```
method uint str.writeappend (
   str filename
)
```

Parameters

filename	Filename.

Return value

The size of the w ritten data.

Related links

- String

spattern

The pattern structure for the searching. The spattern type is used to search through the string for another string. Don't change the fields of the spattern strcuture. The spattern variable must be initialized w ith spattern.init method.

```
type spattern
{
   uint pattern
   uint size
   reserved shift[1024]
   uint flag
}
```

Members

pattern	Hidden data.
size	The size of the pattern.
shift[1024]	Hidden data.
flag	Search flags.

Related links

- String

spattern.init

- method spattern spattern.init(buf pattern, uint flag)
- method spattern spattern.init(str pattern, uint flag)

Creating data search pattern. Before search start-up, call this method in order to initialize the search pattern. Then do a search of the specified pattern w ith spattern.search.

```
method spattern spattern.init (
   buf pattern,
   uint flag
)
```

Parameters

pattern	Search string (pattern).
flag	Search flags.

$QS_IGNCASE	Case-insensitive search.
$QS_WORD	Search the w hole w ord only.
$QS_BEGINWORD	Search w ords w hich start w ith the specified pattern.

Return value

Returns the object w hich method has been called.

spattern.init

Creating data search pattern.

```
method spattern spattern.init (
   str pattern,
   uint flag
)
```

Parameters

pattern	Search string (pattern).
flag	Search flags.

$QS_IGNCASE	Case-insensitive search.
$QS_WORD	Search the w hole w ord only.
$QS_BEGINWORD	Search w ords w hich start w ith the specified pattern.

Related links

- String

spattern.search

- method uint spattern.search(buf src, uint offset)
- method uint spattern.search(uint ptr, uint size)
- method uint spattern.search(str src, uint offset)

Search a pattern in another string. Before search start-up, call the spattern.init method in order to initialize the search pattern.

```
method uint spattern.search (
   buf src,
   uint offset
)
```

Parameters

src	String w here the specified string w ill be searched (search pattern).
offset	Offset w here the search must be started or proceeded.

Return value

The offset of the found fragment. If the offset is equal to string size,no fragment is found.

spattern.search

Search a pattern in a memory data.

```
method uint spattern.search (
   uint ptr,
   uint size
)
```

Parameters

ptr	The pointer to the memory data w here the pattern w ill be searched.
size	The size of the memory data.

Return value

The offset of the found fragment. If the offset is equal to string size,no fragment is found.

spattern.search

Search a pattern in another string.

```
method uint spattern.search (
   str src,
   uint offset
)
```

Parameters

src	String w here the specified string w ill be searched (search pattern).
offset	Offset w here the search must be started or proceeded.

Return value

The offset of the found fragment. If the offset is equal to string size,no fragment is found.

Related links

- String

str.search

Substring search. The method determines if the string has been found inside another string or not.

```
method uint str.search (
   str pattern,
   uint flag
)
```

Parameters

pattern	Search string (pattern).
flag	Search flags.

$QS_IGNCASE	Case-insensitive search.
$QS_WORD	Search the w hole w ord only.
$QS_BEGINWORD	Search w ords w hich start w ith the specified pattern.

Return value

The method returns 1 if the substring is found, otherw ise the return value is zero.

Related links

- String

String - Filename

Filename strings. Methods for w orking w ith file names.

str.faddname	Adding a name.
str.fappendslash	Adding a slash.
str.fdelslash	Deleting the final slash.
str.ffullname	Getting the full name.
str.fgetdir	Getting the directory name.
str.fgetdrive	Getting the name of a disk.
str.fgetext	Get the extension.
str.fgetparts	Getting name components.
str.fnameext	Getting the name of a file.
str.fsetext	Modifying the extension.
str.fsetname	Modifying the name of the file.
str.fsetparts	Compounding or modifying the name.
str.fsplit	Getting the directory and name of a file.
str.fwildcard	Wildcard check.

str.faddname

Adding a name. Add a file name or a directory to a path.

```
method str str.faddname (
   str name
)
```

Parameters

name	The name being added. It w ill be added after a slash.

Return value

Returns the object w hich method has been called.

Related links

- String - Filename

str.fappendslash

Adding a slash. Add '\' to the end of a string if it is not there.

```
method str str.fappendslash()
```

Return value

Returns the object w hich method has been called.

Related links

- String - Filename

str.fdelslash

Deleting the final slash. Delete the final '\' if it is there.

```
method str str.fdelslash()
```

Return value

Returns the object w hich method has been called.

Related links

- String - Filename

str.ffullname

Getting the full name. The method gets the full path and name of a file.

```
method str str.ffullname (
   str name
)
```

Parameters

name	Initial filename.

Return value

Returns the object w hich method has been called.

Related links

- String - Filename

str.fgetdir

Getting the directory name. The method removes the final name of a file or directory.

```
method str str.fgetdir (
   str name
)
```

Parameters

name	Initial filename.

Return value

Returns the object w hich method has been called.

Related links

- String - Filename

str.fgetdrive

Getting the name of a disk. Get the netw ork name (\\computer\share\) or the name of a disk (c:\).

```
method str str.fgetdrive (
   str name
)
```

Parameters

name	Initial filename.

Return value

Returns the object w hich method has been called.

Related links

- String - Filename

str.fgetext

Get the extension. The method w rites the file extension into the result string.

```
method str str.fgetext< result >
```

Return value

The result string w ith the extension.

Related links

- String - Filename

str.fgetparts

Getting name components. Get the directory, name and extensions of a file.

```
method str.fgetparts (
   str dir,
   str fname,
   str ext
)
```

Parameters

dir	The string for getting the directory. It can be 0->str.
fname	The string for getting the file name. It can be 0->str.
ext	The string for getting the file extension. It can be 0->str.

Related links

- String - Filename

str.fnameext

Getting the name of a file. Get the name of the filename or directory from the full path.

```
method str str.fnameext (
   str name
)
```

Parameters

name	Initial filename.

Related links

- String - Filename

str.fsetext

- method str str.fsetext(str name, str ext)
- method str str.fsetext(str ext)

Modifying the extension. The method gets the file name w ith a new extension.

```
method str str.fsetext (
   str name,
   str ext
)
```

Parameters

name	Initial file name.
ext	File extension.

Return value

Returns the object w hich method has been called.

---

str.fsetext

Modifying the extension in the filename.

```
method str str.fsetext (
   str ext
)
```

Parameters

ext	File extension.

Related links

- String - Filename

str.fsetname

Modifying the name of the file. The method modifies the current filename.

```
method str str.fsetname (
   str filename
)
```

Parameters

filename	A new filename.

Return value

Returns the object w hich method has been called.

Related links

- String - Filename

str.fsetparts

Compounding or modifying the name. Compound the name of a file out of the path, name and extension. This function can be also used to modify the path, name or extension of a file. In this case if some component equals 0->str, it is left unmodified.

```
method str str.fsetparts (
   str dir,
   str fname,
   str ext
)
```

Parameters

dir	Directory.
fname	Filename.
ext	File extension.

Return value

Returns the object w hich method has been called.

Related links

- String - Filename

str.fsplit

Getting the directory and name of a file. The method splits the full path into the name of the final file or directory and the rest of the path.

```
method str.fsplit (
   str dir,
   str name
)
```

Parameters

dir	The string for getting the directory.
name	The string for getting the name of a file or directory.

Related links

- String - Filename

str.fwildcard

Wildcard check. Check if a string coincides w ith the specified mask.

```
method uint str.fwildcard (
   str wildcard
)
```

Parameters

wildcard	The mask being checked. It can contain '?' (one character) and '*' (any number of characters).

Return value

Returns 1 if the string coincides w ith the mask.

Related links

- String - Filename

String - Unicode

Unicode strings. It is possible to use variables of the **ustr** type for w orking w ith Unicode strings. The **ustr** type is inherited from the **buf** type. So, you can also use methods of the buf type.

- Operators
- Methods

Operators

*** ustr**	Get the length of a unicode string.
ustr[i]	Getting ushort character **[i]** of the Unicode string.
ustr + ustr	Add tw o strings.
ustr = type	Assign types to unicode string.
str = ustr	Copy a unicode string to a string.
ustr += type	Appending types to the unicode string.
str == ustr	Comparison operation.
ustr < ustr	Comparison operation.
ustr > ustr	Comparison operation.
ustr(str)	Converting a string to a unicode string **ustr(str)**.
str(ustr)	Converting a unicode string to a string **str(ustr)**.

Methods

ustr.clear	Clearing a unicode string.
ustr.copy	Copying.
ustr.del	Delete a substring.
ustr.findch	Find the character in the unicode string.
ustr.fromutf8	Convert a UTF-8 string to a unicode string.
ustr.insert	Insertion.
ustr.lines	Convert a multi-line unicode string to an array of unicode strings.
ustr.read	Read a unicode string from a file.
ustr.replace	Replacing in a unicode string.
ustr.reserve	Memory reservation.
ustr.setlen	Setting a new size of the unicode string.
ustr.split	Splitting a unicode string.
ustr.substr	Getting a unicode substring.
ustr.toutf8	Convert a unicode string to UTF-8 string.
ustr.trim...	Trimming a unicode string.
ustr.w rite	Writing a unicode string to a file.

* ustr

Get the length of a unicode string.

```
operator uint * (
   ustr left
)
```

Return value

The length of the unicode string.

Related links

- String - Unicode

ustr[i]

Getting ushort character **[i]** of the Unicode string.

```
method uint ustr.index (
   uint id
)
```

Return value

The **[i]** ushort character of the Unicode string.

Related links

- String - Unicode

ustr + ustr

- operator ustr +<result> (ustr left, ustr right)
- operator ustr +<result>(ustr left, str right)

Add tw o strings. Putting tw o unicode strings together and creating a resulting unicode string.

```
operator ustr +<result>  (
   ustr left,
   ustr right
)
```

Return value

The new result unicode string.

ustr + str

Add a unicode string and a string.

```
operator ustr +<result> (
   ustr left,
   str right
)
```

Return value

The new result unicode string.

Related links

- String - Unicode

ustr = type

- operator ustr =(ustr left, str right)
- operator ustr =(ustr left, ustr right)

Assign types to unicode string. Copy a string to the unicode string **ustr = str**.

```
operator ustr = (
   ustr left,
   str right
)
```

Return value

The result unicode string.

---

ustr = ustr

Copy a unicode string to another unicode string.

```
operator ustr = (
   ustr left,
   ustr right
)
```

Related links

- String - Unicode

str = ustr

Copy a unicode string to a string.

```
operator str = (
   str left,
   ustr right
)
```

Return value

The result string.

Related links

- String - Unicode

ustr += type

- operator ustr +=(ustr left, ustr right)
- operator ustr +=(ustr left, str right)

Appending types to the unicode string. Append **ustr** to **ustr** => **ustr += ustr**.

```
operator ustr += (
   ustr left,
   ustr right
)
```

Return value

The result unicode string.

---

ustr += str

Append **str** to **ustr** => **ustr += str**.

```
operator ustr += (
   ustr left,
   str right
)
```

Related links

- String - Unicode

str == ustr

- operator uint ==(str left, ustr right)
- operator uint ==(ustr left, str right)

Comparison operation.

```
operator uint == (
   str left,
   ustr right
)
```

Return value

Returns **1** if the strings are equal. Otherw ise, it returns **0**.

---

ustr == str

Comparison operation.

```
operator uint == (
   ustr left,
   str right
)
```

Return value

Returns **1** if the strings are equal. Otherw ise, it returns **0**.

Related links

- String - Unicode

ustr < ustr

- operator uint <(ustr left, ustr right)
- operator uint <=(ustr left, ustr right)

Comparison operation.

```
operator uint < (
   ustr left,
   ustr right
)
```

Return value

Returns **1** if the first string is less than the second one. Otherw ise, it returns **0**.

---

ustr <= ustr

Comparison operation.

```
operator uint <= (
   ustr left,
   ustr right
)
```

Return value

Returns **1** if the first string is less or equal the second one. Otherw ise, it returns **0**.

Related links

- String - Unicode

ustr > ustr

- operator uint >(ustr left, ustr right)
- operator uint >=(ustr left, ustr right)

Comparison operation.

```
operator uint > (
   ustr left,
   ustr right
)
```

Return value

Returns **1** if the first string is greater than the second one. Otherw ise, it returns **0**.

---

ustr >= ustr

Comparison operation.

```
operator uint >= (
   ustr left,
   ustr right
)
```

Return value

Returns **1** if the first string is greater or equal the second one. Otherw ise, it returns **0**.

Related links

- String - Unicode

ustr(str)

Converting a string to a unicode string **ustr(str)**.

```
method ustr str.ustr<result> (
)
```

Return value

The result unicode string.

Related links

- String - Unicode

str(ustr)

Converting a unicode string to a string **str(ustr)**.

```
method str ustr.str<result> (
)
```

Return value

The result string.

Related links

- String - Unicode

ustr.clear

Clearing a unicode string.

```
method ustr ustr.clear
```

Return value

Returns the object w hich method has been called.

Related links

- String - Unicode

ustr.copy

- method ustr ustr.copy(uint ptr, uint size)
- method ustr ustr.copy(uint ptr)

Copying. The method copies the specified size of the data into a unicode string.

```
method ustr ustr.copy (
   uint ptr,
   uint size
)
```

Parameters

ptr	The pointer to the data being copied. If data does not end in a zero, it w ill be added automatically.
size	The size of the data being copied.

Return value

Returns the object w hich method has been called.

---

ustr.copy

The method copies data into a unicode string.

```
method ustr ustr.copy( uint ptr )
```

Parameters

ptr	The pointer to the data being copied. All data to the zero ushort w ill be copied.

Return value

Returns the object w hich method has been called.

Related links

- String - Unicode

ustr.del

Delete a substring.

```
method ustr ustr.del (
   uint off,
   uint len
)
```

Parameters

off	The offset of the substring being deleted.
len	The size of the substring being deleted.

Return value

Returns the object w hich method has been called.

Related links

- String - Unicode

ustr.findch

- method uint ustr.findch(uint off, ushort symbol)
- method uint ustr.findch(ushort symbol)

Find the character in the unicode string.

```
method uint ustr.findch (
   uint off,
   ushort symbol
)
```

Parameters

off	The offset to start searching from.
symbol	Search character.

Return value

The offset of the character if it is found. If the character is not found, the length of the string is returned.

---

ustr.findch

Find the character in the unicode string from the beginning of the string.

```
method uint ustr.findch (
   ushort symbol
)
```

Parameters

symbol	Search character.

Related links

- String - Unicode

ustr.fromutf8

Convert a UTF-8 string to a unicode string.

```
method ustr ustr.fromutf8 (
   str src
)
```

Parameters

src	Source UTF-8 string.

Return value

Returns the object w hich method has been called..

Related links

- String - Unicode

ustr.insert

Insertion. The method inserts one unicode string into another.

```
method ustr ustr.insert (
   uint offset,
   ustr value
)
```

Parameters

offset	The offset w here string w ill be inserted.
value	The unicode string being inserted.

Return value

Returns the object w hich method has been called.

Related links

- String - Unicode

ustr.lines

- method arrustr ustr.lines(arrustr ret, uint flag)
- method arrustr ustr.lines<result>(uint trim)
- method arrustr ustr.lines(arrustr ret)
- method arrustr ustr.lines<result>()

Convert a multi-line unicode string to an array of unicode strings.

```
method arrustr ustr.lines (
   arrustr ret,
   uint flag
)
```

Parameters

ret	The result array of unicode strings.
flag	Flags.

$SPLIT_EMPTY	Take into account empty substrings.
$SPLIT_NOSYS	Delete characters <= space on the left and on the right.
$SPLIT_FIRST	Split till the first separator.
$SPLIT_QUOTE	Take into account that elements can be enclosed by single or double quotation marks.
$SPLIT_APPEND	Adding strings. Otherw ise, the array is cleared before loading.

Return value

The result array of unicode strings.

---

ustr.lines

Convert a multi-line unicode string to an array of unicode strings.

```
method arrustr ustr.lines<result> (
   uint trim
)
```

Parameters

trim	Specify 1 if you w ant to trim all characters less or equal space in lines.

Return value

The new result array of unicode strings.

---

ustr.lines

Convert a multi-line unicode string to an array of unicode strings.

```
method arrustr ustr.lines (
   arrustr ret
)
```

Parameters

ret	The result array of strings.

---

ustr.lines

Convert a multi-line unicode string to an array of unicode strings.

```
method arrustr ustr.lines<result>()
```

Return value

The new result array of unicode strings.

Related links

- String - Unicode

ustr.read

Read a unicode string from a file.

```
method uint ustr.read (
   str filename
)
```

Parameters

filename	Filename.

Return value

The size of the read data.

Related links

- String - Unicode

ustr.replace

Replacing in a unicode string. The method replaces data in a unicode string.

```
method ustr ustr.replace (
   uint offset,
   uint size,
   ustr value
)
```

Parameters

offset	The offset of the data being replaced.
size	The size of the data being replaced.
value	The unicode string being inserted.

Return value

Returns the object w hich method has been called.

Related links

- String - Unicode

ustr.reserve

Memory reservation. The method increases the size of the memory allocated for the unicode string.

```
method ustr.reserve (
   uint len
)
```

Parameters

len	The summary requested length of th eunicode string. If it is less than the current size, nothing happens. If the size is increased, the current string data is saved.

Return value

Returns the object w hich method has been called.

Related links

- String - Unicode

ustr.setlen

- method ustr ustr.setlen(uint len)
- method ustr ustr.setlenptr

Setting a new size of the unicode string. The method does not reserve space. You cannot specify the size of a string greater than the reserved space you have. Mostly, this function is used for specifying the size of a string after external functions w rite data to it.

```
method ustr ustr.setlen (
   uint len
)
```

Parameters

len	New string size.

Return value

Returns the object w hich method has been called.

---

ustr.setlenptr

Recalculate the size of a unicode string to the zero character. The function can be used to determine the size of a string after other functions w rite data into it.

```
method ustr ustr.setlenptr
```

Related links

- String - Unicode

ustr.split

- method arrustr ustr.split(arrustr ret, ushort symbol, uint flag)
- method arrustr ustr.split <result> (uint symbol, uint flag)

Splitting a unicode string. The method splits a string into substrings taking into account the specified separator.

```
method arrustr ustr.split (
   arrustr ret,
   ushort symbol,
   uint flag
)
```

Parameters

ret	The result array of unicode strings.
symbol	Separator.
flag	Flags.

$SPLIT_EMPTY	Take into account empty substrings.
$SPLIT_NOSYS	Delete characters <= space on the left and on the right.
$SPLIT_FIRST	Split till the first separator.
$SPLIT_QUOTE	Take into account that elements can be enclosed by single or double quotation marks.
$SPLIT_APPEND	Adding strings. Otherw ise, the array is cleared before loading.

Return value

The result array of strings.

The method splits a unicode string into the new result array of unicode strings.

```
method arrustr ustr.split <result>  (
   uint symbol,
   uint flag
)
```

Parameters

symbol	Separator.
flag	Flags.

$SPLIT_EMPTY	Take into account empty substrings.
$SPLIT_NOSYS	Delete characters <= space on the left and on the right.
$SPLIT_FIRST	Split till the first separator.
$SPLIT_QUOTE	Take into account that elements can be enclosed by single or double quotation marks.
$SPLIT_APPEND	Adding strings. Otherw ise, the array is cleared before loading.

Return value

The new result array of unicode strings.

Related links

- String - Unicode

ustr.substr

Getting a unicode substring.

```
method ustr ustr.substr (
   ustr src,
   uint start,
   uint len
)
```

Parameters

src	Initial unicode string.
start	Substring offset.
len	Substring size.

Return value

Returns the object w hich method has been called.

Related links

- String - Unicode

ustr.toutf8

Convert a unicode string to UTF-8 string.

```
method str ustr.toutf8 (
   str dest
)
```

Parameters

dest	Destination string.

Return value

The dest parameter.

Related links

- String - Unicode

ustr.trim...

- method ustr ustr.trim(uint symbol, uint flag)
- method ustr ustr.trimrspace()
- method ustr ustr.trimspace()

Trimming a unicode string.

```
method ustr ustr.trim (
   uint symbol,
   uint flag
)
```

Parameters

symbol	The character being deleted.
flag	Flags.

$TRIM_LEFT	Trim the left side.
$TRIM_RIGHT	Trim the right side.
$TRIM_ONE	Delete only one character.
$TRIM_PAIR	If the character being deleted is a bracket, look the closing bracket on the right
$TRIM_SYS	Delete characters less or equal space.

Return value

Returns the object w hich method has been called.

ustr.trimrspace

Deleting spaces on the right.

```
method ustr ustr.trimrspace()
```

ustr.trimspace

Deleting spaces on the both sides.

```
method ustr ustr.trimspace()
```

Related links

- String - Unicode

ustr.write

Writing a unicode string to a file.

```
method uint ustr.write (
   str filename
)
```

Parameters

filename	The name of the file for w riting. If the file already exists, it w ill be overw ritten.

Return value

The size of the w ritten data.

Related links

- String - Unicode

System

System functions.

- Callback and search features
- Type functions

max	Determining the largest of tw o numbers.
min	Determining the smallest of tw o numbers.

Callback and search features

callback	Create a callback function.
freecallback	Free a created callback function.
getid	Getting the code of an object by its name.

Type functions

destroy	Destroying an object.
new	Creating an object.
sizeof	Get the size of the type.
type_delete	Delete the object as located by the pointer.
type_hasdelete	Whether an object should be deleted.
type_hasinit	Whether an object should be initialized.
type_init	Initiate the object as located by the pointer.

max

- func uint max(uint left, uint right)
- func uint max(int left, int right)

Determining the largest of tw o numbers.

```
func uint max (
   uint left,
   uint right
)
```

Parameters

left	The first compared number of the uint type.
right	The second compared number of the uint type.

Return value

The largest of tw o numbers.

---

max

Determining the largest of tw o int numbers.

```
func uint max (
   int left,
   int right
)
```

Parameters

left	The first compared number of the int type.
right	The second compared number of the int type.

Return value

The largest of tw o int numbers.

Related links

- System

min

- func uint min(uint left, uint right)
- func uint min(int left, int right)

Determining the smallest of tw o numbers.

```
func uint min (
   uint left,
   uint right
)
```

Parameters

left	The first compared number of the uint type.
right	The second compared number of the uint type.

Return value

The smallest of tw o numbers.

---

min

Determining the smallest of tw o int numbers.

```
func uint min (
   int left,
   int right
)
```

Parameters

left	The first compared number of the int type.
right	The second compared number of the int type.

Return value

The smallest of tw o int numbers.

Related links

- System

callback

Create a callback function. This function allow s you to use gentee functions as callback functions. For example, gentee function can be specified as a message handler for w indow s.

```
func uint callback (
   uint idfunc,
   uint parsize
)
```

Parameters

idfunc	Identifier (address) of gentee function that w ill be callback function.
parsize	The summary size of parameters (number of uint values). One parameter uint = 1 (uint = 1). uint + uint = 2, uint + long = 3.

Return value

You can use the return value as the callback address. You have to free it w ith freecallback function w hen you don't need this callback function.

Related links

- System

freecallback

Free a created callback function.

```
func freecallback (
   uint pmem
)
```

Parameters

pmem	The pointer that w as returned by callback function.

Related links

- System

getid

Getting the code of an object by its name. The function returns the code of an object (function, method, operator, type) by its name and parameters.

```
func uint getid (
   str name,
   uint flags,
   collection idparams
)
```

Parameters

name	The name of an object (function, method, operator).	
flags	Flags.	
	$GETID_METHOD	Search method. Specify the main type of the method as the first parameter in the collection.
	$GETID_OPERATOR	Search operator. You can specify the operator in name as is. For example, **+=**.
	$GETID_OFTYPE	Specify this flag if you w ant to describe parameters w ith types of items (of type). In this case, collection must contains pairs - idtype and idoftype.
idparams	The types of the required parameters.	

Return value

The code (identifier) of the found object. The function returns **0** if the such object w as not found.

Related links

- System

destroy

Destroying an object. Destroying an object created by the function new .

```
func destroy (
   uint obj
)
```

Parameters

obj	The pointer to the object to be destroyed.

Related links

- System

new

- func uint new (uint objtype)
- func uint new (uint objtype, uint oftype, uint count)

Creating an object. The function creates an object of the specified type.

```
func uint new (
   uint objtype
)
```

Parameters

objtype	The identifier or the name of a type.

Return value

The pointer to the created object.

---

new

The function creates an object w ith specifing the count and the type of its items.

```
func uint new (
   uint objtype,
   uint oftype,
   uint count
)
```

Parameters

objtype	The identifier or the name of a type.
oftype	The type of object's items.
count	The initial count of object's items.

Return value

The pointer to the created object.

Related links

- System

sizeof

Get the size of the type.

```
func uint sizeof (
   uint idtype
)
```

Parameters

idtype	Identifier or the name of the type. The compiler changes the name of the type to its identifier.

Return value

The type size in bytes.

Related links

- System

type_delete

Delete the object as located by the pointer. Gentee deletes objects automaticaly. Use this function only if you allocated the memory for the variable.

```
func type_delete (
   pubyte ptr,
   uint idtype
)
```

Parameters

ptr	The pointer to the memory space w here the object being deleted is located.
idtype	The type of the object.

Related links

- System

type_hasdelete

Whether an object should be deleted. Specifies the necessity to call the function type_delete for deleting an object of this type.

```
func uint type_hasdelete (
   uint idtype
)
```

Parameters

idtype	The type of an object.

Return value

1 is returned if it is necessary to call type_delete, **0** is returned otherw ise.

Related links

- System

type_hasinit

Whether an object should be initialized. Specifies the necessity to call the function type_init for initiating an object of this type.

```
func uint type_hasinit (
   uint idtype
)
```

Parameters

idtype	The type of an object.

Return value

1 is returned if it is necessary to call type_init, **0** is returned otherw ise.

Related links

- System

type_init

Initiate the object as located by the pointer. Gentee initializes objects automaticaly. Use this function only if you allocated the memory for the variable.

```
func uint type_init (
   pubyte ptr,
   uint idtype
)
```

Parameters

ptr	The pointer to the memory space w here the object being created is located.
idtype	The type of the object.

Return value

The pointer to the object is returned.

Related links

- System

Thread

This library allow s you to create threads and w ork w ith them. The methods described above are applied to variables of the **thread** type. For using this library, it is required to specify the file thread.g (from lib\thread subfolder) w ith include command.

```
include : $"...\gentee\lib\thread\thread.g"
```

- Methods
- Functions

Methods

thread.create	Create a thread.
thread.getexitcode	Get the thread exit code.
thread.isactive	Checking if a thread is active.
thread.resume	Resuming a thread.
thread.suspend	Stop a thread.
thread.terminate	Terminating a thread.
thread.wait	Waiting till a thread is exited.

Functions

exitthread	Exiting the current thread.
sleep	Pause the current thread for the specified time.

thread.create

Create a thread.

```
method uint thread.create (
   uint idfunc,
   uint param
)
```

Parameters

idfunc	The pointer to the function that w ill be called as a new thread. The function must have one parameter. You can get the pointer using the operator &.
param	Additional parameter.

Return value

The handle of the created thread is returned. It returns 0 in case of an error.

Related links

- Thread

thread.getexitcode

Get the thread exit code.

```
method uint thread.getexitcode (
   uint result
)
```

Parameters

result	The pointer to a variable of the uint type the thread exit code w ill be w ritten to. If the thread is still active, the value $STILL_ACTIVE w ill be w ritten.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Thread

thread.isactive

Checking if a thread is active.

```
method uint thread.isactive()
```

Return value

Returns 1 if the thread is active and 0 otherw ise.

Related links

- Thread

thread.resume

Resuming a thread. Resume a thread paused w ith the thread.suspend method.

```
method uint thread.resume()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Thread

thread.suspend

Stop a thread.

```
method uint thread.suspend()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Thread

thread.terminate

Terminating a thread.

```
method uint thread.terminate (
   uint code
)
```

Parameters

code	Thread termination code.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Thread

thread.wait

Waiting till a thread is exited.

```
method uint thread.wait()
```

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Thread

exitthread

Exiting the current thread.

```
func exitthread (
   uint code
)
```

Parameters

code	Thread exit code.

Related links

- Thread

sleep

Pause the current thread for the specified time.

```
func sleep (
   uint msec
)
```

Parameters

msec	The time for pausing the thread in milliseconds.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Thread

Tree

Tree object. The each node of tree object can have a lot of childs. It is required to include tree.g.

```
include : $"...\gentee\lib\tree\tree.g"
```

- Operators
- Methods
- Treeitem methods

Operators

tree of type	Specifying the type of items.
* tree	Get the count of items in a tree.
* treeitem	Get the count of childs in the tree item.
foreach var,treeitem	Foreach operator.

Methods

tree.clear	Delete all items in the tree.
tree.del	Deleting an item.
tree.leaf	Adding a "leaf".
tree.node	Adding a "node".
tree.root	Get the root item of a tree.

Treeitem methods

treeitem.changenode	Change the parent node of an item.
treeitem.child	Get the first child of an item.
treeitem.data	Get the pointer to the data stored in an object.
treeitem.getnext	Getting the next item to the current tree item.
treeitem.getprev	Getting the previous item to the current tree item.
treeitem.isleaf	Check if it is a leaf.
treeitem.isnode	Check if it is a node.
treeitem.isroot	Check if it is a root item.
treeitem.lastchild	Get the last child item of the tree item.
treeitem.move	Move an item.
treeitem.parent	Get the parent of an item.

tree of type

Specifying the type of items. You can specify **of** type w hen you describe **tree** variable. In default, the type of the items is **uint**.

```
method tree.oftype (
   uint itype
)
```

Related links

- Tree

* tree

Get the count of items in a tree.

```
operator uint * (
   tree itree
)
```

Return value

The count of childs in the tree.

Related links

- Tree

* treeitem

Get the count of childs in the tree item.

```
operator uint * (
   treeitem treei
)
```

Return value

The count of childs in the tree item.

Related links

- Tree

foreach var,treeitem

Foreach operator. You can use **foreach** operator to look over all items of the treeitem. **Variable** is a pointer to the child tree item.

```
foreach variable,treeitem {...}
```

Related links

- Tree

tree.clear

Delete all items in the tree.

```
method tree tree.clear (
)
```

Return value

Returns the object w hich method has been called.

Related links

- Tree

tree.del

- method tree.del(treeitem item, uint funcdel)
- method tree.del(treeitem item)

Deleting an item. Delete an item together w ith all its child items.

```
method tree.del (
   treeitem item,
   uint funcdel
)
```

Parameters

item	The item being deleted.
funcdel	The custom function that w ill be called before deleting the each item. It can be 0.

tree.del

Delete an item together w ith all its child items.

```
method tree.del (
   treeitem item
)
```

Parameters

item	The item being deleted.

Related links

- Tree

tree.leaf

- method treeitem tree.leaf(treeitem parent, treeitem after)
- method treeitem tree.leaf(treeitem parent)

Adding a "leaf". Add a "leaf" to the specified node. You can not add items to a "leaf".

```
method treeitem tree.leaf (
   treeitem parent,
   treeitem after
)
```

Parameters

parent	Parent node. If it is 0->treeitem then the item w ill be added to the root.
after	Insert an item after this tree item. If it is 0->treeitem then the item w ill be the first child.

Return value

The added item or 0 in case of an error.

---

tree.leaf

Add a "leaf" to the specified node. An item w ill be the last child item.

```
method treeitem tree.leaf (
   treeitem parent
)
```

Parameters

parent	Parent node. If it is 0->treeitem then the item w ill be added to the root.

Return value

The added item or 0 in case of an error.

Related links

- Tree

tree.node

- method treeitem tree.node(treeitem parent, treeitem after)
- method treeitem tree.node(treeitem parent)

Adding a "node". Add a "node" to the specified node. You can add items to a "node".

```
method treeitem tree.node (
   treeitem parent,
   treeitem after
)
```

Parameters

parent	Parent node. If it is 0->treeitem then the item w ill be added to the root.
after	Insert an item after this tree item. If it is 0->treeitem then the item w ill be the first child.

Return value

The added item or 0 in case of an error.

---

tree.node

Add a "node" to the specified node. An item w ill be the last child item.

```
method treeitem tree.node (
   treeitem parent
)
```

Parameters

parent	Parent node. If it is 0->treeitem then the item w ill be added to the root.

Return value

The added item or 0 in case of an error.

Related links

- Tree

tree.root

- method treeitem tree.root
- method treeitem treeitem.getroot()

Get the root item of a tree.

```
method treeitem tree.root
```

Return value

Returns the root item of the tree.

---

treeitem.getroot

Get the root item of a tree.

```
method treeitem treeitem.getroot()
```

Return value

Returns the root item of the tree.

Related links

- Tree

treeitem.changenode

Change the parent node of an item.

```
method uint treeitem.changenode (
   treeitem treei
)
```

Parameters

treei	New parent node.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Tree

treeitem.child

Get the first child of an item.

```
method treeitem treeitem.child()
```

Return value

Returns the first child item or 0 if there is none.

Related links

- Tree

treeitem.data

Get the pointer to the data stored in an object.

```
method uint treeitem.data()
```

Return value

Returns the pointer to the data.

Related links

- Tree

treeitem.getnext

Getting the next item to the current tree item.

```
method treeitem treeitem.getnext()
```

Return value

Returns the next item.

Related links

- Tree

treeitem.getprev

Getting the previous item to the current tree item.

```
method treeitem treeitem.getprev()
```

Return value

Returns the previous item.

Related links

- Tree

treeitem.isleaf

Check if it is a leaf. The method checks if an item is a "leaf" (if it cannot have child items).

```
method uint treeitem.isleaf
```

Return value

Returns 1 if this item is a tree "leaf" and 0 otherw ise.

Related links

- Tree

treeitem.isnode

Check if it is a node. The method checks is an item can have child items.

```
method uint treeitem.isnode
```

Return value

Returns 1 if this item is a tree "node" and 0 otherw ise.

Related links

- Tree

treeitem.isroot

Check if it is a root item. The method checks if an item is a root one.

```
method uint treeitem.isroot
```

Return value

Returns 1 if this item is a root one and 0 otherw ise.

Related links

- Tree

treeitem.lastchild

Get the last child item of the tree item.

```
method treeitem treeitem.lastchild()
```

Return value

Returns the last child item or 0 if there is none.

Related links

- Tree

treeitem.move

- method uint treeitem.move(treeitem after)
- method uint treeitem.move(treeitem target, uint flag)

Move an item.

```
method uint treeitem.move (
    treeitem after
)
```

Parameters

after	The node to insert the item after. Specify 0 if it should be made the first item.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

---

treeitem.move

Move an item.

```
method uint treeitem.move (
    treeitem target,
    uint flag
)
```

Parameters

target	The node to insert the item after or before depending on the flag.
flag	Move flag.

$TREE_FIRST	The first child item of the same parent.
$TREE_LAST	The last child item of the same parent.
$TREE_AFTER	After this item.
$TREE_BEFORE	Before this item.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- Tree

treeitem.parent

Get the parent of an item.

```
method treeitem treeitem.parent()
```

Return value

Returns the parent of this item.

Related links

- Tree

XML

XML file processing. This library is used for XML file processing and XML tree building. Neither a multibyte-character set nor a document type description **<!DOCTYPE .....>** are handled in the current version. For using this library, it is required to specify the file xml.g (from lib\xml subfolder) w ith include command.

```
include : $"...\gentee\lib\xml\xml.g"
```

- Operators
- Methods
- Methods of XML tree items

XML description	A brief description of XML library.

Operators

foreach var,xmlitem	Foreach operator.

Methods

xml.addentity	Adds an entity description.
xml.getroot	Gets the root item of the XML document tree.
xml.procfile	Process an XML file.
xml.procstr	Processes a string contained the XML document.

Methods of XML tree items

xmlitem.chtag	Gets a tag item w ith the help of a "path".
xmlitem.findtag	Search for a tag item by the name.
xmlitem.getattrib	Gets a tag item attribute value.
xmlitem.getchild	Gets the first child item of the current item.
xmlitem.getchildtag	Gets the first child tag item.
xmlitem.getchildtext	Gets the first child text item.
xmlitem.getname	Gets the name of the XML item.
xmlitem.getnext	Gets the next item.
xmlitem.getnexttag	Gets the next tag item.
xmlitem.getnexttext	Gets the next text item.
xmlitem.getparent	Gets the parent item of the current item.
xmlitem.gettext	Gets a text of the current item in the XML tree.
xmlitem.isemptytag	Determines if the item is an empty tag item.
xmlitem.ispitag	Checks if the item is a tag processing instruction.
xmlitem.istag	Determines if the current item is a tag item.
xmlitem.istext	Determines if the current item is a text item.

XML description

A brief description of XML library. Variables of either the **xml** and the **xmlitem** type (an XML tree item) are used for processing XML documents. An XML tree item can be of two types: a **text item** and a **tag item**. There are several types of tag items:

- tag item that contains other items **<tag ...>.....</tag>**;
- tag item that contains no other items **<tag .../>**;
- tag item of processing instruction **<?tag ...?>**.

A tag item may contain attributes.

The sequence of operations for processing an XML document:

- process a document (build an XML tree) with the help of the xml.procfile method or the xml.procstr method;
- add entity definitions, using the xml.addentity method if necessary;
- search for the required items in the XML tree using the following methods: xml.getroot, xmlitem.chtag, xmlitem.findtag, xmlitem.getnext, etc.;
- use the **foreach** statement in order to process similar elements if necessary;
- gain access to tag attributes with the help of the xmlitem.getattrib method and get a text using the xmlitem.gettext method.

Related links

- XML

foreach var,xmlitem

Foreach operator. Looking through all items w ith the help of the **foreach** operator. Defining an optional variable of the **xmltags** type is required. The foreach statement is used for variables of the **xmlitem** type and goes through all child tag items of the current tag.

```
xmltags xtags
xmlitem curtag
...
foreach xmlitem cur, curtag.tags( xtags )
{
   ...
}
```

```
foreach variable,xmlitem.tags( xmltags ) {...}
```

Related links

- XML

xml.addentity

Adds an entity description. The entity must have been described before the gettext method is called. Below you can see the list of entities described by default:
&amp; - **&**;
&quot; - **"**;
&apos; - **'**;
&gt; - **>**;
&lt; - **<**;

```
method xml.addentity (
   str key,
   str value
)
```

Parameters

key	Key (an entity name - **&entity_name;**).
value	Entity value is a string that w ill be pasted into the text.

Related links

- XML

xml.getroot

Gets the root item of the XML document tree. Actually, a root item contains all items of an XML document tree only.

```
method xmlitem xml.getroot()
```

Return value

Returns a root item.

Related links

- XML

xml.procfile

Process an XML file. Reads the XML file, the name of w hich is specified as a parameter, and process it.

```
method uint xml.procfile (
   str filename
)
```

Parameters

filename	Name of the file processed.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- XML

xml.procstr

Processes a string contained the XML document.

```
method uint xml.procstr (
   str src
)
```

Parameters

src	XML data string.

Return value

If the function succeeds, the return value is **1**. If the function fails, the return value is **0**.

Related links

- XML

xmlitem.chtag

Gets a tag item w ith the help of a "path". Searches through the XML tree for a tag item w ith the help of the specified "path". A "path" consists of tag names separated by the '/' character, if the first character in a path is the '/' character, the item search begins from the tree root; otherw ise - from the current item.

```
method xmlitem xmlitem.chtag (
   str path
)
```

Parameters

path	Path of the item.

Return value

Returns the item obtained or zero, if no item has been found.

Related links

- XML

xmlitem.findtag

Search for a tag item by the name. Searches through the XML tree for a tag item w ith the specified name. The item is searched recursively through all child items.

```
method xmlitem xmlitem.findtag (
   str name
)
```

Parameters

name	Name of the required tag.

Return value

Returns the item obtained or zero, if no item has been found.

Related links

- XML

xmlitem.getattrib

Gets a tag item attribute value.

```
method str xmlitem.getattrib (
   str name,
   str result
)
```

Parameters

name	Attribute name.
result	Result string.

Return value

Returns the string that contains the attribute value. If no attribute has been found, it returns an empty string.

Related links

- XML

xmlitem.getchild

Gets the first child item of the current item.

```
method xmlitem xmlitem.getchild()
```

Return value

Returns the child item or zero, if the item does not contain any child items.

Related links

- XML

xmlitem.getchildtag

Gets the first child tag item. This method is similar to the xmlitem.getchild method; how ever, if the child item is not a tag item, in this case, the tag item that comes first is searched through the child items.

```
method xmlitem xmlitem.getchildtag()
```

Return value

Returns the child tag item or zero, if the item does not contain any child tag items.

Related links

- XML

xmlitem.getchildtext

Gets the first child text item. This method is similar to the xmlitem.getchild method; how ever, if the child item is not a text item, in this case, the text item that comes first is searched through the child items.

```
method xmlitem xmlitem.getchildtext()
```

Return value

Returns the child text item or zero, if the item does not contain any child text items.

Related links

- XML

xmlitem.getname

Gets the name of the XML item.

```
method str xmlitem.getname (
   str res
)
```

Parameters

res	Result string.

Return value

Returns the parameter **res**.

Related links

- XML

xmlitem.getnext

Gets the next item. How ever, the next item must be searched through the items w ith the same parent item.

```
method xmlitem xmlitem.getnext()
```

Return value

Returns the next item or zero, if the item is the last item.

Related links

- XML

xmlitem.getnexttag

Gets the next tag item. This method is similar to the xmlitem.getnext method, but if the next item is not a tag item, this operation repeats.

```
method xmlitem xmlitem.getnexttag
```

Return value

Returns the next tag item or zero, if the item is the last item.

Related links

- XML

xmlitem.getnexttext

Gets the next text item. This method is similar to the xmlitem.getnext method, but if the next item is not a text item, this operation repeats.

```
method xmlitem xmlitem.getnexttext()
```

Return value

Returns the next text item or zero, if the item is the last item.

Related links

- XML

xmlitem.getparent

Gets the parent item of the current item.

```
method xmlitem xmlitem.getparent()
```

Return value

Returns the parent item or zero, if the current item is the root item.

Related links

- XML

xmlitem.gettext

Gets a text of the current item in the XML tree. This method is applied either to a text item or a tag item, in the latter case, the text is obtained from the child text item.

```
method str xmlitem.gettext (
   str result
)
```

Parameters

result	Result string.

Return value

Returns the string that contains the text of the item. If no text has been found, it returns an empty string.

Related links

- XML

xmlitem.isemptytag

Determines if the item is an empty tag item. Determines if the current item is a tag item, that contains no child items **<tag .../>**;.

```
method uint xmlitem.isemptytag()
```

Return value

Returns nonzero if the item is a tag item, that contains no child items; otherw ise, it returns zero.

Related links

- XML

xmlitem.ispitag

Checks if the item is a tag processing instruction. Determines if the current item is a tag of processing instruction **<?tag ...?>**.

```
method uint xmlitem.ispitag()
```

Return value

Returns nonzero if the item is a tag of processing instruction, otherw ise, it returns zero.

Related links

- XML

xmlitem.istag

Determines if the current item is a tag item.

```
method uint xmlitem.istag()
```

Return value

Returns nonzero if the item is a tag item; otherw ise, it returns zero.

Related links

- XML

xmlitem.istext

Determines if the current item is a text item.

```
method uint xmlitem.istext()
```

Return value

Returns nonzero if the item is a text item; otherw ise, it returns zero.

Related links

- XML

Samples

Welcome to Gentee Programming Language! This tutorial w ill help you master our programming language by using easy-to-understand examples.

You should have some familiarity w ith computers. But it is not essential to have extensive experience in programming or to know any other computer languages. In fact, these lessons w ere w ritten under the assumption that you have little or no programming skills.

The tutorial begins w ith basic concepts and then builds on them w ith more complex lessons. If a lesson is too simple for you, skip over it! If you're a novice, take things step-by-step.

This tutorial does not concentrate primarily on the syntax and semantics of Gentee Programming Language. (Information about this can be found in the documentation.) Instead, the tutorial concentrates on the development of language skills by developing softw are solutions to practical computing problems. Furthermore, each lesson includes a self-study exercise for independent w ork. We feel that programming skills are best acquired through practice. Therefore, w e recommend that you complete the exercises.

Each lesson features source code that w ill help you understand how to develop programming solutions. If developing solutions seems daunting, carefully examine the source code for hints about how the program w orks. These source programs are located in the **Samples** subdirectory of the Gentee distribution kit.

hello	A simple program outputs a string to a console.
square	Ñalculating the area and the perimeter of a rectangle and of a circle.
easymath	Finding the greatest common divisor, factorial and the Fibonacci numbers.
primenumber	Calculate primes using "The Sieve of Eratosthenes".
fileattrib	Set or remove the attributes of the files a read-only file.
runini	Using INI files
easyhtml	Display a color palette as HTML, w hich is frequently used for creating an HTML page.
calendar	Create a month calendar, selected by a user, in HTML format.
samefiles	Find all files w hich have the same contents either in the required folder or in a drive.

If you encounter difficulty, send us an e-mail and w e w ill try to help you troubleshoot the problem. If your problem provides a useful lesson for others, w e w ill seek your permission to disassemble the program and release it to others to provide them w ith better understanding.

hello

There is a tradition in programming tutorials to show the code that prints "Hello, World!" on the computer screen. We w ill adhere to that tradition by show ing you the Gentee code that produces "Hello World:"

Example 1

```
func hello <main>
{
   print( "Hello, World!" )
   getch()
}
```

To understand how the code displays the results printed on your screen, you need to understand certain concepts.

The "Hello World" printed on the screen is called a "function." A function is a type of procedure or routine that performs a specific task. Some programming languages make a distinction betw een a function, w hich returns a value, and a procedure, w hich performs a specific task but does not return a value In the case of Gentee, functions are denoted by the operation set that performs any task.

Functions can be called from other ones. Functions are described by the keyw ord **func** in Gentee. The function w ith the name **hello** and the attribute **main** have just been mentioned, the attribute means that this function w ill be run after loading the program.

```
print( "Hello, World!" )
```

print is a function call outputting the specified string.

A string is a series of characters manipulated as a group. A character string is often specified by enclosing the characters in quotes. For example, WASHINGTON w ould be a name, but "WASHINGTON" w ould be character strings. In Gentee, strings are enclosed in double quotes. In other w ords, the quote marks help you define a string.

After w e designate a character string, w e call another function **getch** that results in a keystroke delay.

More information about this coding can be found in the documentation. In the lessons you can also find information about other functions and methods.

Now , let's talk about strings. There is a command character **'\'**, that performs some actions depending on the follow ing characters. This example demonstrates some of them:

\n represents a new line.
\\ represents the symbol: backslash '\'.

In addition to this, Gentee saves line feeds w ithin the string. A line feed is a code that moves the cursor on a display screen dow n one line. In the example below , the follow ing strings are equivalents.

```
"Hello, World!
Hello, World!"
"Hello, World!\nHello, World!"
```

Exercise 2

Make a program "Hello, World!" that prompts the user to press any key.

square

You w ill get acquainted w ith numbers in this lesson. We w ill first try to make a program to calculate the area of a rectangle and of a circle. We w ill use numbers w ith double precision (double type). Double precision refers to a type of floating-point number that has more precision, or more digits to the right of the decimal point, than a single-precision number.

To begin w ith, create a framew ork of the function.

```
func main<main>
{
   while 1
   {
      print("Enter the number of the action:
1. Calculate the area of a rectangle
2. Calculate the area of a circle
3. Exit\n")
      switch getch()
      {
         case '1'
         {
            print("Specify the width of the rectangle: ")
            print("Specify the height of the rectangle: ")
         }
         case '2'
         {
            print("Specify the radius of the circle: ")
         }
         case '3', 27 : break
         default : print("You have entered the wrong value!\n\n")
      }
   }
}
```

There are tw o new statements here: **w hile** and **sw itch**.

The **w hile** statement repeats the execution of a code, w hile the conditional expression is nonzero. In this case, the condition equals 1, that means an endless loop and the command **break** ,as defined below , causes an exit from the loop.

A loop is one of the three basic logic structures in computer programming. The other tw o logic structures are selection and sequence.

The **sw itch** operator evaluates an expression and looks for the value through the values. **case**. While the program is w aiting for the keystroke, a user thinks of further actions. Now let's take a look at the follow ing line:

```
case '3', 27 : break
```

Notice that the possible values separated by commas are enumerated in **case**. 27 determines the key code **Esc**. As for the symbol ':', it is denoted by the follow ing line enclosed in braces. In other w ords, this fragment is equivalent to the follow ing one:

```
case '3', 27 { break }
```

The use of braces is often required by Gentee, a usage of the symbol ':' helps you escape piling characters in simple tasks.

To perform calculations w e use a string type of a variable for the return values and a double type of tw o variables in order to store values. You can start by appending:

```
str      input
double   width height
```

Variables of the same type are separated by a comma or a single space.

Now you can perform calculations and get answ ers. So, to calculate the area of a rectangle w e could construct code like this:

```
print("Specify the width of the rectangle: ")
width = double( conread( input ))
print("Specify the height of the rectangle: ")
height = double( conread( input ))
print("The area of the rectangle: \( width * height )\n\n")
```

The **conread** function reads data input by a user. The **\(...)** operation w ithin the string evaluates the expression enclosed in brackets and inserts data into the string.

To calculate the area of a circle, w e can create another example similar to the code, above:

```
print("Specify the radius of the circle: ")
width = double( conread( input ))
print("The area of the circle: \( 3.1415 * width * width )\n\n")
```

Exercise 2

Write a program that calculates the perimeter of a rectangle and of a circle. Use a separate function for the perimeter of each shape.

easymath

Example 1

Now w e w ill analyze an example that finds the greatest common divisor of tw o numbers (GCD).

We take advantage of Euclid's Algorithm for the task solution. It w orks like this:

GCD(x, y) = x if y equals 0 and
GCD(x, y) = GCD(y, x MOD y) if y is nonzero.

x MOD y is the remainder of values.
In other w ords, dividing tw o numbers w e compute the remainder of the values but if the remainder is nonzero, the second number and the remainder of the values must be considered, etc.

The follow ing fragment provides an obvious example of recursion, that's used to have a function call itself from w ithin itself.
This function is w ritten like this:

```
func uint gcd( uint first second )
{
   if !second : return first
   return gcd( second, first % second )
}
```

% is used to divide tw o numbers and returns the remainder.
uint is a type designating a positive integer.
if is a conditional statement show n in the follow ing example:

```
if condition {
}
elif condition {
}
else {
}
```

Note: An infinite number of **elif** blocks can be used. If the condition is TRUE, statements in braces follow ing this condition w ill be executed.

Finally, it is time to w rite the main function that can receive the data from the user and call **gcd**. The function contains the follow ing loop:

```
while 1
{
   first = uint( congetstr( "Enter the first number ( enter 0 to exit ): ",
input ))
   if !first : break
   second = uint( congetstr( "Enter the second number: ", input ))
   print("GCD = \( gcd( first, second ))\n\n")
}
```

congetstr is a function provided by standard libraries, it outputs a text to the screen and receives the data from the user.

Example 2

Calculate a factorial n! for n from 1 to 12. A factorial determines the product of numbers up to the given number inclusive.

The follow ing program demonstrates its task solution:

```
func uint factorial( uint n )
{
   if n == 1 : return 1
   return n * factorial( n - 1 )
}

func main<main>
{
   uint    n

   print("This program calculates n! ( 1 * 2 *...* n ) for n from 1 to 12\n\n"
)
   fornum n = 1, 13
   {
      print("\(n)! = \(factorial( n ))\n")
```

```
    }
    getch()
}
```

The **fornum** loop is executed w hile the counter variable "n" is considered less than the value of the second expression. The loop counter increases by increments of 1 at each step. **fornum** is a special case of the **for** operator - more on that later.

```
for counter = expression,expression,change of the value of counter
{
}
```

Exercise 3

Now w e need to calculate the Fibonacci numbers.

These are a series of w hole numbers in w hich each number is the sum of the tw o preceding numbers. Beginning w ith 0 and 1, the sequence of Fibonacci numbers w ould be 0,1,1, 2, 3, 5, 8, 13, 21, 34, etc. using the formula: n = n(-1) + n(-2), w here the n(-1) means "the last number before n in the series" and n(-2) refers to "the second last one before n in the series."

We w ill calculate until the last number exceeds 2000000000. Use a recursive function.
X0 = 1
X1 = 1
...
Xn = Xn-1 + Xn-2

Exercise 4

Perform the previous task w ithout the use of recursion.

primenumber

Example 1

Calculate primes using "The Sieve of Eratosthenes".

Now w e w ill explain the task.

Primes are products of tw o numbers. In other w ords, a prime is divisible only by itself or 1. So, calculating primes by means of "The Sieve of Eratosthenes" is done like this:

We start w ith a list of candidates containing numbers from 2 to the definite number. 2 is a prime. We remove all even numbers from the list. Now , w e w ill take 3 and remove all the numbers that are products of it. After this, w e find the next number from the candidate list. This number is 5, then w e remove the fifth numbers from the list, etc. Therefore, w hen the candidate list is empty, the result list w ill contain all the primes.

Let's break the problem into tw o steps. The first step takes advantage of the algorithm, and the second step outputs the results to the screen.

```
str   input
uint  high i j

print("This program uses \"The Sieve of Eratosthenes\" for finding prime 
numbers.\n\n")
high = uint( congetstr("Enter the high limit number ( < 100000 ): ", input ))
if high > 100000 : high = 100000

arr  sieve[ high + 1 ] of byte

fornum i = 2, high/2 + 1
{
   if !sieve[ i ]
   {
      j = i + i
      while j <= high
      {
         sieve[ j ] = 1
         j += i
      }
   }
}
```

To begin w ith, a user should enter the number w hich is defined as the final candidate in the list. We w ant all primes below 100000 in order to not use a lot of resources.

```
arr  sieve[ high + 1 ] of byte
```

sieve is a description of an array of bytes. And array is a series of objects, all of w hich are the same size and type. Each object in an array is called an array element. For example, you could have an array of integers or an array of characters or an array of anything that has a defined data type. The important characteristics of an array are that (1) Each element has the same data type (although they may have different values). (2) The entire array is stored contiguously in memory.

The first element in the array is the 0th element, therefore let's set 1 to this number. Actually, arrays and variables are zero-based. So, if an element of the array equals 0, then this based number is not removed. If w e remove it, w e'll set 1 to this number.

For the **fornum** loop w e use only half of the numbers. Why do w e do this? Think it over. Then w e apply the algorithm. As you can see, it takes up several strings. Let us jump into an example that illustrates how it w orks:

```
j += i
```

This is an extension of the *j* variable to *i*. Similar operations are applied for the multiplication, the division, the subtraction.

The numbers has already been removed; let's now jump to the second step.
It is certainly possible that the numbers are output to the screen, nevertheless let's save a file.

```
j = 0
input.setlen( 0 )

fornum i = 2, high + 1
{
   if !sieve[ i ]
   {
      input.out4( "%8u", i )
      if ++j == 10
      {
         j = 0
         input += "\l"
      }
   }
}

input.write( "prime.txt" )
shell( "prime.txt" )
```

To find out more about the **out4** method, read the documentation. In this case each number is extended to eight symbols by spaces. Furthermore, after outputting each tenth number w e start a new line. The *j* variable performs it.

A combination of carriage return **'\r'** and new line **'\n'** is used for line feed in text files. In Gentee there's only **'\l'** command w hich executes it.

The **w rite** method w rites the string to the file, and the function **shell** opens this specified file in the appropriate application.

fileattrib

This lesson focuses on files.

Example 1

Set or remove the attributes of the files **a read-only file**. This program accepts command line parameters. The files can be stored in templates, using the follow ing operators: '*' and '?'. The '*' operator defines any sequence of characters, '?' represents a single character.
Thus,
*c:\temp\*.** - all files in the folder c:\temp
c:\temp\*.exe - all files w ith the extension exe in the folder c:\temp
c:\temp\a.** - all files beginning w ith 'a' in the folder c:\temp
c:\temp\ab?.** - all files beginning w ith 'ab' and one other character in the folder c:\temp

So, w e start w ith command line. It is fairly easy, because of tw o functions: **argc** returns the number of arguments, **argv** returns the required parameter. The first parameter must be the w ord **on** or **off** for setting or removing the attribute, the second parameter must be the template for file processing. So, w e can do it like this:

```
if argc() > 1
{
   if argv( temp, 1 ) %== "on" : mode = 1
   elif argv( temp, 1 ) %== "off" : mode = 2
   argv( path, 2 )
}
```

The '%==' operator produces a line-by-line comparison ignoring the characters' case. Here, you can w rite **ON** as w ell as **Off**.

If the parameters have not been indicated by the time the program starts or you typed ones that are not valid, give a chance to input necessary information on the console.

```
if !mode
{
   mode = conrequest( "Choose an action (press a number key):
1. Turn on readonly attribute
2. Turn off readonly attribute
3. Exit\n", "1|2|3" ) + 1

   if mode == 3 : return

   congetstr( "Specify a filename or a wildcard: ", path )
}
```

Here the user has to type: **1** to set the attribute, **2** to remove it and **3** to exit the program. The conrequest function w aits for the keystroke, then returns the number of the selected variant from 0.
For example,

```
conrequest("Press #'Y#' or #'N#'", "Yy|Nn" )
```

OK. Now w e proceed to the task solution. The ffind structure searches for the specified filename. Let's describe and initialize the variable fd of type ffind.

```
fd.init( path, $FIND_FILE | $FIND_RECURSE )
```

$FIND_FILE points to the search of specified filenames.
$FIND_RECURSE indicates the search of specified filenames in all subdirectories.

For instance,

```
fd.init( "c:\\temp.txt", $FIND_FILE | $FIND_RECURSE )
```

w ith the specified flag $FIND_RECURSE w ill search for the filename: temp.txt on the entire C: drive.

The **foreach** operator is used for file searching:

```
foreach cur,fd
{
   attrib = getfileattrib( cur.fullname )
   if mode == 1 : attrib |= $FILE_ATTRIBUTE_READONLY
   else : attrib &= ~$FILE_ATTRIBUTE_READONLY
   setfileattrib( cur.fullname, attrib )
   print( "\(cur.fullname)\n" )
}
```

finfo is a type that stores information about files. More information about this can be found in Help.
cur is a variable of the specified type w hich contains the stored information about any file that has been found.

Now , I w ould like to say a few w ords about loop content. We obtain the current file attributes

```
attrib = getfileattrib( cur.fullname )
```

According to conditions w e set or remove the attribute of the file **a read-only file**. Other attributes are saved.

```
if mode == 1 : attrib |= $FILE_ATTRIBUTE_READONLY
else : attrib &= ~$FILE_ATTRIBUTE_READONLY
```

We w rite the modified attributes of the file.

```
setfileattrib( cur.fullname, attrib )
```

runini

Now we try to automate the EXE files created using the ge2exe program, which is integrated into the compiler. This lesson describes the procedure mentioned above, making it easier for users to create the EXE files.

Example 1

. Let INI-file be a file, which contains the information about programs in the Gentee language. We have to help users choose a program from the given list, compile it and create the EXE file, if necessary.

We start with the description of the INI-file format. Each section denotes a program and consists of the following fields:
Name is the name of a program.
Src is the .g file of a program.
Exe - to create the EXE file or not (If the field contains a 1 or a 0).
Run - to run the program after successful compilation or not.
Output - If you want to change the last file name or store it in the other directory, you should enter the specified file name and its path here.

Note that the **Src** is a required field.

```
[ID2]
Name = Square
Src  = ..\square\square.1.g
Exe  = 0
```

The INI-file can be changed; it`s up to you. Moreover, you can add elements. Take a look at the **runini.ini** file used as an example in the **samples\runini** subdirectory.

The **ini.g** library is required to deal with the INI-file. So, let`s include the library by using the **include** command. To illustrate this, assume that these examples are located in the subdirectory **samples**, so we use the relative path. If you would like to carry this example to another directory, you should enter the absolute path.

```
include : $"..\..\lib\ini\ini.g"
```

The string with the initial dollar sign '$' does not contain any command characters, however it may contain macros. It is interesting to note that the use of such strings makes it easier to define the path to the files, because there is no need to double the '\' sign.

Let's write two auxiliary functions.

```
func uint  openini( ini retini )
```

The **openini** function reads data from the runini.ini - file; but if the file is not available, the error message is displayed. If you want to get the error code, take a look at the source program.

```
func uint  idaction( ini retini, str section )
```

This function is considered to be significant. It calls the program that can compile and create the exe-file. The first argument is the file object ini, the second one is the name of the section that should be launched.

The following statements read the field values.

```
retini.getvalue( section, "Src", src, "" )
if !*src
{
   congetch("ID '\(section)' is not valid. Press any key...\n")
   return 0
}
run = retini.getnum( section, "Run", 1 )
exe = retini.getnum( section, "Exe", 0 )

retini.getvalue( section, "Output", outname, "" )
```

Note that the last argument of the **getvalue** and the **getnum** functions defines the value, if this field isn't defined in the INI-file.

Using the options from the INI-file, the following code generates command lines in order to start up the compiler and ge2exe. The **process** function makes the programs start up. The "." directive, as the second argument of the **process** function, indicates that gentee.exe and ge2exe.exe will use the current directory as their working directory.

```
if exe
{
   process( "..\\..\\exe\\gentee.exe -p samples \(src)", ".", &result )
   src.fsetext( src, "ge" )
   process( "..\\..\\exe\\ge2exe.exe \(src)", ".", &result )
   deletefile( src )
   src.fsetext( src, "exe" )
   if run : process( src, ".", &result )
}
else : shell( src )
```

Let us jump into an example that illustrates the function body, w hich displays a list of possible programs and receives the program name chosen by a user:

```
ini      tini
arrstr   sections
str      name src section

openini( tini )

tini.sections( sections )
while 1
{
   print( "-----------\n" )
   foreach  cur, sections
   {
      tini.getvalue( cur, "Src", src, "" )
      if !*src : continue

      tini.getvalue( cur, "Name", name, src )
      print( "\(cur)".fillspacer( 20 ) + name + "\n" )
   }
   print( "-----------\n" )
   congetstr("Enter ID name (enter 0 to exit): ", section )
   if section[0] == '0' : break

   idaction( tini, section )
}
```

First, w e read the INI-file and get the section list contained in a string array. After the program list is displayed in a w indow , a user should choose a program name. Then, w e call the **idaction** function w ith the required program name.

Here, the follow ing string is described in detail.

```
print( "\(cur)".fillspacer( 20 ) + name + "\n" )
```

The **fillspacer** method appends a specified number of space characters onto the end of the string. As you can see, w e call the method on the string enclosed in double quote marks. Note, that in Gentee a string enclosed in double quote marks is the same object as a variable of type str. Furthermore, w e can call methods on functions and other methods, w hich return strings.
For example, the expression given below appends ten space characters onto the end of the string, thus increasing the string's length to 30 characters.

```
"ID: \(cur)".fillspacer( 20 ).fillspacel( 30 )
```

Exercise 2

Write a program using runini.1.g, that gets a program-section name from a command line and runs it. If no command-line argument is specified, this program must w ork like the program discussed above.

Tip: my program consists of 14 lines. For more details about this, read runini.2.g.

easyhtml

This lesson presents one type of function, the **text** function. As its name suggests, this type of function deals w ith texts. Unlike other functions, a text w ith built-in code serves as the basis for text functions.

Example 1

Display a color palette, w hich is frequently used for creating an HTML page. Save data as an HTML file.

First, determine the number of colors displayed in one line using the define command.

```
define {
   lcount = 12
}
```

Note that such constant quantities are called macros. The dollar sign '$' is used before the name in order to run them.
Let's tackle the last point first.

```
func color< main >
{
   str out

   out @ colorhtm()
   out.write( "color.htm" )
   shell( "color.htm" )
}
```

```
out @ colorhtm()
```

As you can see, the result of the **colorhtm** text function is output to the out string. Using the follow ing commands, w e save the obtained string to the file w hich is opened in the brow ser w indow . In our example, ellipses are substituted for the title and the end of the html file.

```
text  colorhtm
...
\{
   int vrgb i j k
   uint cur

   subfunc outitem
   {
      str  rgb

      rgb.out4( "%06X", vrgb )
      @ item( rgb )
      if ++cur == $lcount
      {
         @"</TR><TR>"
         cur = 0
      }
   }
   for i = 0xFF, i >= 0, i -= 0x33
   {
      for j = 0xFF, j >= 0, j -= 0x33
      {
         for k = 0xFF, k >= 0, k -= 0x33
         {
            vrgb = ( i << 16 ) + ( j << 8 ) + k
            outitem()
         }
      }
   }
   for vrgb = 0xFFFFFF, vrgb >= 0, vrgb -= 0x111111 : outitem()
   for vrgb = 0xFF0000, vrgb > 0, vrgb -= 0x110000 : outitem()
   for vrgb = 0x00FF00, vrgb > 0, vrgb -= 0x001100 : outitem()
   for vrgb = 0x0000FF, vrgb > 0, vrgb -= 0x000011 : outitem()
}
...
\!
```

The **\{...}** command is used to insert the code into a text. **outitem** minorant function is defined like thisas:

```
rgb.out4( "%06X", vrgb )
@ item( rgb )
```

Here, using the local variable named **vrgb**, the string is created that contains the hexadecimal representation, then another text function **item** is called for outputting the cell w ith the indicated color. The unary operator **@** is used to output into the current string or, if there is no string to the console. Then, w e determine the total number of cells in the row and add a new row to the table w here approriate.

We use three embedded color cycles for searching possible values. Red, green or blue color components are affected by color cycling. Then these color components are arranged in the vrgb variable and w e call the minorant function described above.

The next four color cycles display additional palette entries for gray, red, green and blue colors.
The **\!** command indicates the termination of a text function. By default, a text function w orks until the end of the file.

Let's take an example - a text function of cell entries.
We w ant a function that w orks like this:

```
text  item( str rgb )
<TD ALIGN=CENTER><TABLE BGCOLOR=#\(rgb) WIDTH=60><TR><TD>  </TD></TR></TABLE>
<FONT FACE="Courier">\(rgb)</FONT>
</TD>
\!
```

As you can see, this is an HTML text that outputs the **rgb** color parameter. It is used as the background color of a table cell and for the display of its value output under the table cell.

Exercise 2

Create a HTML file that contains the multiplication table.

calendar

This lesson provides you w ith a little more practice w ith the text function.

Example 1

Create a month calendar, selected by a user, in HTML format.

We're betting that this example w ill make a lot more sense to you. So, how w ould w e do this? Let's use the main function from the previous example and modify it; that is, a user enters the year required for creating a calendar.

```
congetstr( "Enter a year: ", year )
out @ calendar( uint( year ))
out.write( "calendar.htm" )
shell( "calendar.htm" )
```

Now , w e describe a variable of **datetime** type in the **calendar** text function and set January 1 of the specified year into this variable. Then w e output the title of the HTML file and start creating the calendar.

```
text  calendar( uint year )
\{ datetime  stime
   stime.setdate( 1, 1, year )
}<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Calendar for year \(stime.year)</TITLE>
<STYLE TYPE="text/css">
<!--
BODY {background: #FFF; font-family: Verdana;}
H1 {text-align: center; margin: 5px;}
TABLE {border: 0; border-spacing: 7px;}
TD {padding: 3px; border: 1px solid; text-align: center;}
#copy {font-size: smaller; text-align: center;}
-->
</STYLE>
</HEAD>
<BODY><H1>\(stime.year)</H1>
<TABLE ALIGN=CENTER>
```

Notice that this calendar contains three columns and four row s. The first day of the w eek is stored w ith the help of the **firstday** variable for the customer. **dayofweek** returns the day number for the current date value. The **nameofmonth** function returns the name of the month in a user language.

```
firstday = firstdayofweek()
dayofweek = stime.dayofweek
fornum i = 0, 4
{
   @"\l<TR>"
   fornum j = 1, 4
   {
      month = i * 3 + j
      @"\l<TD>\(nameofmonth( stemp,  month ))
<PRE>"
      ...
   }
}
```

Now that w e have defined it, w e can use **abbrnameofday** function to obtain the abbreviated day name. It is essential to add missing space characters, because the calendar includes characters w hich have the identical w idth. So, let's use each day name w ith four characters.

```
fornum k = firstday, firstday + 7
{
   @"  \( abbrnameofday( stemp, k ).setlen( 2 ))"
}
@"  \l"
@"    ".repeat( ( 7 + dayofweek - firstday ) % 7 )
```

If **dayofweek** function has the value 0, any Sunday is highlighted using red color. Our attention is turned to line feeds added after the last day of the w eek. The number of strings output is stored in the **lines** variable in the follow ing w ay:

```
uint day = 1
uint lines
while day <= daysinmonth( year, month )
{
   if !dayofweek : @"<FONT COLOR=red>"
   @str( day++ ).fillspacel( 4 )
   if !dayofweek : @"</FONT>"

   dayofweek = ( dayofweek + 1 ) % 7
   if dayofweek == firstday
   {
      @"   \l"
      lines++
   }
}
```

Finally, the space characters are inserted into the last string and the missing row s are output in order to create the months w ith the identical height.

```
@"    ".repeat( ( 7 + firstday - dayofweek ) % 7 )

while lines++ < 7 :  @"   \l"
@"</PRE>"
```

Frankly speaking, this task is difficult to comprehend because it provides a w ealth of HTML texts and extra formatting. On the other hand, w e finally succeed in w riting this program.

samefiles

This lesson focuses on file processing. Let's find the duplicate files on your computer. Moreover, w e try to make this task enjoyable and useful, i.e. w e w ill find the duplicate files using their contents, but their names are not required. Well, I suppose you w ill be astonished by the results, w hen these operations have been completed.

Example 1

Find all files w hich have the same contents either in the required folder or in a drive.

Actually, task performance takes much time. Let's think of the algorithm that w ill make this task easy to perform. If the files differ in size, they are not duplicate. So, from this assumption, first w e can get names and sizes of all compared files, w hich w ill be sorted by size, after that it w ill be possible to compare them by size.

Let's declare the structure for data storage w ith help of the **type** command. We w ill store only a file name instead of its w hole path in order to save memory. The index of the parent directory in the directory array w ill be stored in the field ow ner instead of full name.

```
type  finf
{
   str   name
   uint  size
   uint  owner
}
```

We need the follow ing global variables:

```
global
{
   arr dirs  of finf
   arr files of finf
   arr sizes of uint
   str output
}
```

dirs - processed directory array.
files - files array.
sizes - array that contains files indices w ill be sorted.
output - string for result output.

The functions w ritten below are responsible for appending directories and files to the appropriate arrays.

```
func uint newdir( str name, uint owner )
func uint newfile( str name, uint size owner )
```

To find out more about these functions, read the source code. Functions append an element to the array and fill the element fields.

The **scanfolder** function is used to find all directories and files by the specified path. If the directory has been found, the element w ill be appended to the **dirs** array; then this element is considered to be parent and the **scanfolder** function calls itself. If the file has been found, the element w ill be appended to the files array. To make this task easier, w e don't take files w hich have size more than 4GB, condition *!cur.sizehi*serves this purpose particularly.

```
func scanfolder( str wildcard, uint owner )
{
   ...
      if cur.attrib & $FILE_ATTRIBUTE_DIRECTORY
      {
         scanfolder( cur.fullname + "\\*.*", newdir( cur.name, owner ))
      }
   ...
}
```

The **scaninit** function prepares the first call **scanfolder** using the starting path. Modifying these functions, you can use various masks and specify size limits of the compared files for file searching.

```
func scaninit( str folder )
{
   str wildcard

   folder.fdelslash()
   @"Scanning \( folder )\n"
   scanfolder( (wildcard = folder ).faddname( "*.*" ), newdir( folder, 0 ))
}
```

After file scanning w e w ill sort obtained data. So, instead of sorting the files array, w e offer you the better w ay for process speed-up: to create a new array that contains indices of the files and sort it. Actually, w hile sorting elements get moved faster, thus elements of small size w ill be a better choice.

```
func int sortsize( uint left right )
{
   return int( files[ left->uint ].size ) - int( files[ right->uint ].size )
}

func sortfiles
{
   uint i

   @"Sorting...\n"
   sizes.expand( *files )
   fornum i, *sizes : sizes[ i ] = i

   sizes.sort( &sortsize )
}
```

The **sortfiles** function fills the sizes array w ith indices of the files. First, an index equals to the sequence number. Then, the sizes array w ill be sorted w ith help of the **sortsize** function. Such parameters as *left* and *right* are pointers to data. If elements of the array w ere structures, they w ould be used as objects; how ever, the element of the *sizes* array is uint, so w e use the **->uint** operation. This expression: **files[index].size** returns the size of a specified file. The function returns a positive number if the size of the left file is greater than the size of the right one, or a negative number if the size of the left file is less than the size of the right one, or zero if the sizes of both files are equal.

The **getdir** and **getfile** functions retrieve the full path of a file using the value of the ow ner field. **getdir** passes recursively through the first parent directory and makes up the full path as it comes back.

```
func str getdir( uint id, str ret )
func str getfile( uint id, str ret )
```

Let's jump to discussion of the main comparison function. In the loop w e look through all sorted files, w here the file of the least size is the first one.

```
func compare
{
   ...
   fornum i, *sizes - 1
   {
      id = sizes[ i ]

      if !*files[ id ].name : continue
```

The files that are considered to be duplicate are ignored in this example. Names of duplicate files w ill be nulled.

```
found = 0
      next = sizes[ j = i + 1 ]

      while files[ id ].size == files[ next ].size
      {
```

In the given loop the current file is compared w ith the files of the same size that come next. Furthermore, w e miss the obtained duplicate files. Comparison is made using the **isequalfiles** function from the standard library. In case of duplicate files, w e output a message to the string *output*.

```
if *files[ next ].name &&
             isequalfiles( getfile( id, idname ), getfile( next, nextname ))
         {
            if !found
            {
               output @ "\lSize: \(files[ id ].size) ========\l\( idname )\l"
            }
            count++
            ( output @ nextname ) @"\l"

            found = 1
```

```
                files[ next ].name.clear()
            }
            if ++j == *sizes : break
            next = sizes[ j ]
        }
```

This fragment outputs partial results. **i & 0x3F** defines the output of the result after each 64th file.

```
if i && !( i & 0x3F )
        {
            @ "\rApproved files: \(i) Found the same files: \(count)"
        }
    }
    ...
}
```

Using these functions

```
func init
func search
func main<main
```

is not a difficult task. There can be a great number of files and directories, so w e reserve a place for some elements in the **init** function in advance. Moreover, w e append one empty parent element to the **dirs** array in order to start directory numbering at 1. We consider that the ow ner field equals to zero either if the directory is root. In other w ords, the directory has no zero index.

Exercise 2

Write a program for searching duplicate files on all local hard drives.