Краткий обзор Gentee

Введение

Язык программирования Gentee можно отнести к разряду процедурных языков с некоторыми возможностями объектно-ориентированного программирования. Он не имеет сложных конструкций и прост в использовании, но в тоже время является мощным инструментом для решения самых различных задач. Синтаксис языка основан на синтаксисе языка программирования **C** и имеет много общего с другими C-подобными языками **C++**, **Java**, **C#**. У Gentee есть те же самые числовые типы **int**, **uint**, **byte**, **ubyte**, **long**, **double**, **float**, ... и такие же операции над ними **+**, **==**, **<**, **>**, **-**, **/**, **+=**, **++**, **--**, **/=**,... как в других подобных языках программирования. При написании программ вы можете использовать все основные конструкции, которые встречаются в других языках. Например, такие как: **while**, **if**, **for**, **with**, **foreach**, **switch**, **include** .

Единицей компиляции Gentee является описание-команда. Ниже приведены примеры описания глобальных переменных и макросов.

global

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

define

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

Если для обращения к переменной или вызова функции достаточно указать ее имя и параметры (в случае функции), то для подстановки макроса необходимо добавлять слева '\$'. Значения макросов подставляются на этапе компиляции.

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

Типы

Кроме базовых числовых и встроенных типов **buf**, **str**, **collection** можно описывать свои типы с помощью команды **type**.

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

Переменные любого типа не требуют дополнительной инициализации после своего описания, вы сразу можете обращаться к этой переменной. Обращение к полям типа происходит с помощью операции '!'. Типы могут наследоваться как в объектно-ориентированных языках, при этом обеспечивается полиморфизм операций. Если отсутствует метод или функция для какой-то переменной определенного типа, то будут искаться аналогичные методы для ее типов-родителей. Для любых типов можно определить и использовать операции используемые для числовых типов (**=**, **+=**, **==**, **!=**), а также цикл **foreach**.

```
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
}
```

```
}
```

Функции

Gentee имеет три вида команд для определения исполняемого кода: **func**, **method**, **operator**. Исполнение программы начинается с функции, имеющей атрибут **main**.

func - Обычная функция, которая отвечает за выполнение указанных в ней действий.

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

method - Функция, привязанная к определенному типу. Вызов метода подобен взятию поля у типа и осуществляется с помощью '.', за которой следует имя метода и параметры.

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

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

operator - Данная команда позволяет определять и использовать в дальнейшем для любых типов операторы присваивания, сравнения, арифметические операторы и т.д.

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

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

Gentee - язык строгой типизации. Это накладывает определенные ограничения при программировании, но с другой стороны значительно уменьшает вероятность

появления ошибок. Допускается присутствие нескольких функций или методов с одним и тем же именем, но они должны иметь хотя бы один различающийся параметр или разное количество параметров.

Строки

Gentee имеет широчайшие возможности по работе со строками. Строки определяются с помощью двойных кавычек и имеют управляющий символ '\'. Если строка начинается с '\$', то она не будет учитывать управляющий символ. Кроме подстановки специальных символов, управляющий символ дает возможность вставлять данные из файлов, вычислять и подставлять выражения внутри строки, а также подставлять макросы.

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

Часто бывает необходимо вывести большие объемы текста, причем часть текста должна формироваться динамически. В этом случае удобно использовать **text-функции**. Они могут осуществлять вывод в строку, которую вы указали при вызове, или на консоль.

```
text mytext( uint x )
Some text
x = \( x )
x * x = \( x*x )
\{ uint i
```

```

fornum i, 5
{
    @"x * \ ( i ) = \ ( x * i ) \ 1"
}
}
Some text
\!

```

Импорт функций и использование Gentee в других приложениях

Gentee сразу разрабатывался таким образом, чтобы можно было, с одной стороны, импортировать функции из DLL (или аналогичных модулей на других ОС) и, с другой стороны, чтобы можно было использовать компилятор Gentee из программ, написанных на других языках программирования.

Если вам необходимо импортировать функции из DLL, то достаточно указать имя DLL файла и описать импортируемые функции.

```

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

```

Если вы хотите компилировать файлы на языке Gentee и исполнять их из своего приложения, то вам достаточно взять файл gentee.dll и вызывать необходимые интерфейсные функции. Модуль gentee.dll можно использовать бесплатно, но необходимо соблюдать [лицензионное соглашение](#).

Заключение

Скажем несколько слов о работе компилятора. Исходные тексты компилятора на языке программирования C находятся в открытом доступе, так как Gentee является проектом с открытыми исходными текстами. Скорость компиляции очень высокая. В результате компиляции программы создается байт-код, который можно сохранить в файл или выполнить сразу. Сохраненный байт-код можно запускать без повторной компиляции, а можно использовать в качестве библиотечного модуля в других программах. Следует заметить, что имеется набор готовых библиотек, который постоянно пополняется и помогает создавать программы самой различной сложности. Кроме этого, возможно создание исполняемых (exe) файлов.

Мы показали только основные моменты, характерные для языка программирования Gentee. Вы всегда можете найти на этом сайте дополнительную информацию и обсудить любые вопросы с разработчиками и другими пользователями Gentee.

История создания Gentee

Алексей Кривоногов

Идея создания своего языка программирования появилась у меня в конце 90-х годов. Мне хотелось сделать программирование более простым и удобным занятием. Тогда же начались пробные эксперименты по созданию простых языков, но реальное воплощение эти занятия приобрели только в 2002 году. Я в то время работал над созданием инсталлятора, и как раз возникла потребность в простом скриптовом языке. К работе подключился мой брат, что позволило довольно быстро выпустить пробную версию языка. Я бы не стал называть ее прототипом Gentee, но мы приобрели неоценимый опыт и получили представление о том, каким должен быть наш будущий язык.

В 2003 году я приостановил работу над остальными проектами и всерьез занялся созданием Gentee. Мы с братом все же не могли тратить на Gentee все свое время, поэтому работа затянулась больше чем на год. Я бы сказал что самое трудное было не программирование компилятора. Самое трудное было принимать решение по синтаксису и возможностям языка. Gentee разрабатывался как процедурный язык программирования, мы сразу отказались от объектов и классов в их классическом понимании, правда следует заметить, что сейчас имеется и наследование типов и полиморфизм. За основу был взят C-подобный синтаксис как наиболее лаконичный и проверенный временем. Я смело могу назвать Gentee очень субъективным языком. Когда язык делают 1-2 человека, то он получается таким, каким они хотят его видеть.

Не могу сказать, что работа шла гладко. Над некоторыми проблемами приходилось ломать голову не один день. Бывало выбирался тупиковый путь и некоторые вещи переделывались, а от других возможностей приходилось отказываться. Все таки хотелось создать компактный и быстрый компилятор. Мы сразу планировали возможность использования Gentee из других приложений посредством небольшого DLL файла и не могли себе позволить сильно перегружать язык.

Первая публичная версия компилятора была выложена в Интернете **1 ноября 2004** года. Эту дату можно считать днем рождения языка. После этого регулярно выходили версии с новыми возможностями, была даже выпущена версия для Linux. К сожалению, Gentee не смог получить то распространение на которое мы надеялись. В июне 2006 года было решено сделать Gentee **проектом с открытыми исходными текстами**. Компилятор Gentee был бесплатным с самого начала, но мы решили не выкладывать имеющиеся на тот момент исходники, а переписать все заново. Язык к тому времени уже устоялся, но кое-что требовало доработки и переделки. Работы по переписыванию компилятора опять заняли больше года, так как постоянно прерывались и держались на одном нашем энтузиазме. Можно сказать, что в августе-сентябре 2007 года Gentee обрел свое второе рождение. Как он будет развиваться в дальнейшем - покажет время, но мне хотелось бы, чтобы наши усилия не пропали даром. Сейчас, когда открыты все исходные тексты компилятора и библиотек мне хотелось бы чтобы как можно больше человек включилось в процесс развития и улучшения Gentee.

Open Source лицензия (MIT лицензия)

Copyright (c) 2006-2009 The Gentee Group. Все права защищены.

1. Данная лицензия разрешает, безвозмездно, лицам, получившим копию данного программного обеспечения и сопутствующей документации (в дальнейшем именуемыми "Программное Обеспечение"), использовать Программное Обеспечение без ограничений, включая неограниченное право на использование, копирование, изменение, добавление, публикацию, распространение, sublicензирование и/или продажу копий Программного Обеспечения, также как и лицам, которым предоставляется данное Программное Обеспечение, при соблюдении следующих условий:

2. Вышеупомянутый копирайт и данные условия должны быть включены во все копии или значимые части данного Программного Обеспечения.

3. ДАННОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПРЕДОСТАВЛЯЕТСЯ «КАК ЕСТЬ», БЕЗ ЛЮБОГО ВИДА ГАРАНТИЙ, ЯВНО ВЫРАЖЕННЫХ ИЛИ ПОДРАЗУМЕВАЕМЫХ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ГАРАНТИЯМИ ТОВАРНОЙ ПРИГОДНОСТИ, СООТВЕТСТВИЯ ПО ЕГО КОНКРЕТНОМУ НАЗНАЧЕНИЮ И НЕНАРУШЕНИЯ ПРАВ. НИ В КАКОМ СЛУЧАЕ АВТОРЫ ИЛИ ПРАВООБЛАДАТЕЛИ НЕ НЕСУТ ОТВЕТСТВЕННОСТИ ПО ИСКАМ О ВОЗМЕЩЕНИИ УЩЕРБА, УБЫТКОВ ИЛИ ДРУГИХ ТРЕБОВАНИЙ ПО ДЕЙСТВУЮЩИМ КОНТРАКТАМ, ДЕЛИКТАМ ИЛИ ИНОМУ, ВОЗНИКШИМ ИЗ, ИМЕЮЩИМ ПРИЧИНОЙ ИЛИ СВЯЗАННЫМ С ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ ИЛИ ИСПОЛЬЗОВАНИЕМ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИЛИ ИНЫМИ ДЕЙСТВИЯМИ С ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ.

Связаться с Gentee Group можно по электронной почте info@gentee.com.

Более подробную информацию о Gentee Group и Gentee Open Source проекте можно найти на сайте www.gentee.ru.

Синтаксис языка

Вы открыли руководство по синтаксису и семантике языка программирования **Gentee**. Здесь описаны все синтаксические конструкции языка, а также рассмотрены возможности по программированию, которые предоставляет язык.

Это руководство не является учебником по программированию. Здесь описана официальная версия языка от разработчиков компилятора Gentee.

Язык программирования **Gentee** является процедурным языком высокого уровня. Синтаксис языка имеет много общего с синтаксисом C/C++, что помогает многим пользователям почти сразу начинать писать программы на Gentee. Подобно языкам **Java** или **C#** исходный текст программы компилируется в промежуточный код, который затем выполняется виртуальной машиной.

Содержание

- Базовые элементы языка
 - [Идентификаторы](#)
 - [Числа](#)
 - [Строки](#)
 - [Двоичные данные](#)
 - [Макросы](#)
 - [Коллекции](#)
- [Структура программы. Препроцессор](#)
 - [Комментарии. Замена символов](#)
 - [Команда define](#)
 - [Команда ifdef](#)
 - [Макровыражения](#)
 - [Команда include](#)
 - [Команда import](#)
 - [Команды public и private](#)
- [Типы и переменные](#)
 - [Команда type](#)
 - [Наследование типов](#)
 - [Системные методы для типов](#)
 - [Команда global](#)
 - [Локальные переменные](#)
- [Функции методы операции](#)
 - [Определение функции func](#)
 - [Определение метода method](#)
 - [Переопределение операций operator](#)
 - [Определение text функции](#)
 - [Свойства property](#)
 - [Команда extern](#)
 - [Подфункции subfunc](#)
 - [Возвращение переменных](#)
- [Конструкции языка](#)
 - [Конструкция условия if-elif-else](#)
 - [Конструкция выбора switch](#)
 - [Конструкции цикла while и do](#)
 - [Конструкции цикла for и fornum](#)
 - [Конструкция цикла foreach](#)
 - [Инструкции return, break, continue](#)
 - [Инструкции label, goto](#)
 - [Конструкция with](#)
- [Выражения и операторы](#)
 - [Арифметические операторы](#)
 - [Логические операторы](#)
 - [Операторы присваивания](#)
 - [Приведение типов](#)
 - [Поля и указатели](#)
 - [Вызов функций и методов](#)
 - [Условный оператор ?](#)
 - [Операция позднего связывания](#)

- [Таблица приоритетов операторов](#)
- Приложение
- [Язык Gentee в БНФ](#)

Идентификаторы

Идентификаторы - это имена используемые для названия переменных, типов, функций, методов и т.д. Идентификаторы могут состоять из букв, цифр и знака подчеркивания. Начинаться имя может только с буквы или знака подчеркивания. Можно использовать как буквы английского алфавита, так и символы с кодом от 0x80 до 0xFF. Тем не менее, мы рекомендуем использовать только буквы английского алфавита для того, чтобы имена правильно отображались на других компьютерах. Длина имени ограничена 255 символами. Примеры допустимых имен: **_my12**, **temp**, **MainFunction**. Регистр букв имеет значение. **MyFunc** и **myfunc** являются разными именами.

В языке существуют зарезервированные имена, которые нельзя использовать в качестве идентификаторов. Это так называемые **ключевые слова**, которые служат для обозначения конструкций или объектов языка. Ниже приведен список ключевых или служебных слов.

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.

Числа

Язык Gentee имеет несколько числовых типов. Натуральные или целые числа могут быть представлены несколькими способами.

Десятичная запись

Наиболее распространенная форма представления.

Например: **65**, **-45367**, **0**

Шестнадцатеричная запись

В этом случае числа должны начинаться с **0X** или **0x**. Символы от **A** до **F** могут быть как в верхнем, так и в нижнем регистре.

Например: **0xBA23**, **0x1d2f**, **0XFFFFFF**

Двоичная запись

Числа в двоичной записи должны начинаться с **0b** или **0B** и состоять только из 0 или 1.

Например: **0b11001**, **0B1010110110**, **0b10101011000011**

Код символа

Вы можете указывать конкретный символ вместо соответствующего ему числа. Для этого заключите этот символ в одинарные кавычки.

Например: **'A'**, **(')**, **'k'**, **'2'**, **'='**

В Gentee имеется тип **long** и **ulong**, которые занимают по 8 байт. Для обозначения таких чисел добавляйте в конце **L** или **l**.

Например: **23l**, **0xfaafd45fff67ffl**, **-24363627252652L**

Действительные числа

Имеется два типа действительных чисел: **double** и **float**. Число с десятичной точкой или с мантисой имеет тип **double**. Для определения числа типа **float** необходимо добавить в конце **F** или **f**. Для записи числа с типом **double** без десятичной точки и мантисы нужно указывать в конце букву **D** или **d**.

Примеры чисел **double**: **123.122**, **-123.2e-2**, **789D**

Примеры чисел **float**: **12.75f**, **0.55F**, **-78F**

Строки

Строки в языке определяются с помощью пары двойных кавычек. Если подряд идут две строки, то они будут объединены в одну строку. По умолчанию, константные строки не могут быть определены в кодировке Unicode. В Gentee есть Unicode строки (`ustr`) и вы можете использовать кодировку **UTF-8** в константных строках для последующей конвертации в Unicode. Простые строковые переменные определяются указанием типа `str`.

```
"Это простая строка
из двух строчек."
```

Имеется служебный символ `\`, который позволяет выполнять различные действия или производить замены. Ниже приведен список команд со служебным символом.

`\` Вывод служебного символа.

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

`"` Вывод двойных кавычек.

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

`\n` Перевод строки, код 0x0A.

`\r` Возврат каретки, код 0x0D.

`\t` Горизонтальная табуляция, код 0x09.

`\` Конец строки - комбинация `\r\n`. Может быть полезным при выводе в текстовый файл.

`\0XX` Комбинация служебного символа с нулём, а затем число-байт в шестнадцатеричном виде позволяет вставить в строку любой символ с соответствующим кодом от 0 до 255.

`\#` Удаление предшествующих переводов строки или пробелов и знаков табуляции. Удаляются только или переводы строки или пробелы и знаки табуляции в зависимости от того, что стояло впереди.

`\0xd 0xa` Если строка просто заканчивается служебным символом, то последующие символы перевода строки будут удалены. Это удобно использовать для разбивки слишком длинной строки.

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

`!...*\` Комментарии. Вы можете вставлять любые комментарии внутрь строки.

`\$macro$` Вставка в строку значения макроса препроцессора. Последний знак `'$'` является необязательным если далее следует не буква и не цифра.

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

`( выражение )` Вставляется результат выражения. Внутри круглых скобок должно быть выражение любого типа, имеющего метод конвертации в строку.

`< имя файла >` Вставляется содержимое указанного файла. Внутри угловых скобок должно быть указано имя файла без учета служебного символа.

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

`[idname]` Если у вас длинная строка и вы хотите в каком-то фрагменте отключить служебный символ, то укажите в квадратных скобках любую комбинацию любых символов. Можно даже не указывать ни одного дополнительного символа. В дальнейшем для включения служебного символа укажите эту же комбинацию в квадратных скобках.

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

Кроме обычной строки есть еще так называемая **макрострока**. У нее перед кавычками стоит знак `'$'`. Эта строка отличается от обычной строки тем, что у нее нет служебного символа, но она подставляет макросы которые встречаются внутри. Этот тип строки очень удобен при указании путей файлов.

```
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
```

Двоичные данные

Двоичные данные определяются с помощью пары одинарных кавычек. Элементом двоичных данных могут быть числа в десятичном и шестнадцатеричном виде и строки. Числа могут разделяться пробелами, запятыми, переводами строк и точкой с запятой. Двоичным данным соответствует тип **buf**.

Для представления различных элементов используются комбинации со служебным символом '\':

... Комментарии. Вы можете вставлять любые комментарии внутрь двоичных данных.

\\$macro\$ Вставка в строку значения макроса препроцессора. Последний знак '\$' является необязательным.

\(выражение) Вставляется результат выражения. Внутри круглых скобок должно быть выражение любого типа, имеющего метод конвертации в двоичные данные.

\< имя файла > Вставляется содержимое указанного файла. Внутри угловых скобок должно быть указано имя файла без учета служебного символа.

"строка" Вставка макростроки в бинарные данные. Нулевой символ добавится, если вы заключите строку в круглые скобки \("string").

\h Переключение в режим чтения шестнадцатеричных чисел. Далее могут идти цифры 2, 4, 8, которые указывают на размер числа в байтах. Если размер не указан, то числа будут рассматриваться как байты. Режим чтения байт в шестнадцатеричном представлении является режимом по умолчанию.

\i Режим чтения чисел в десятичном виде. В этом режиме можно определять числа с плавающей точкой. После **i** также может быть указана размерность чисел 2, 4 или 8.

```
'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>'
```

Макросы

Макросы - это такие константы, которые подставляются на этапе компиляции. Макросы могут быть идентифкаторами, числами, строками, двоичными данными и коллекциями. Для замены макроса на его значение необходимо указать имя макроса заключенное между знаками '\$'. Если после макроса идет любой символ, который нельзя использовать в имени, то знак '\$' в конце имени ставить не обязательно. Макросы не являются переменными и им нельзя присваивать никакие значения. Макросы определяются с помощью команды [define](#). Макросы также можно использовать для условной компиляции в операторе [ifdef](#).

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

Имеется несколько predefined макросов, которые нельзя изменить.

Предопределенные макросы

\$_FILE	Полное имя текущего компилируемого файла.
\$_LINE	Текущая строка в файле.
\$_DATE	Текущая дата в формате DDMMYYYY.
\$_TIME	Текущее время в формате HHMMSS.
\$_WINDOWS	Равен 1 в операционной системе Windows.
\$_LINUX	Равен 1 в операционной системе Linux.

Смотрите также

- [Команда define](#)
- [Команда ifdef](#)
- [Макровыражения](#)

Коллекции

Коллекции позволяют хранить вместе данные различных типов. Кроме этого они могут служить для инициализации массивов и любых других структур. Также коллекции можно использовать для передачи неопределенного количества параметров разных типов в функции и методы. Коллекциям соответствует тип **collection**. Коллекции определяются с помощью фигурных скобок `{ ... }`. Внутри фигурных скобок можно указывать через запятую различные типы данных или другие коллекции.

Коллекциями, не содержащими выражений, можно инициализировать глобальные переменные.

global

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

Для инициализации переменной определенного типа с помощью коллекций необходимо предварительно описать оператор присваивания коллекции данному типу.

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
}
```

После этого можно инициализировать переменные простым присваиванием:

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

Используя параметр-коллекцию в функции, можно передавать неопределенное количество параметров разных типов.

func outvals(collection cl)

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

Вызов этой функции может быть следующим:

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

Структура программы. Препроцессор

Программа на языке Gentee может быть оформлена в виде одного или нескольких файлов. Основным элементом программы является команда. Команда начинается на новой строке, большинство команд содержит в себе блоки ограниченные фигурными скобками { }. Все команды можно разбить на четыре группы по их целевому назначению.

Команды препроцессора

Препроцессор отвечает за подстановку значений макросов, за замену служебных символов и за условную компиляцию. Препроцессор производит свои действия непосредственно во время компиляции текущего фрагмента исходного кода.

Команда define	Определение макросов.
Команда ifdef	Условная компиляция.

Команды исполняемого кода

Эти команды содержат операторы и отвечают за исполняемую часть программы.

Команда extern	Предопределение функций, методов, операторов.
func	Функция.
Определение метода method	Метод для типа.
Переопределение операций operator	Определение оператора для типа.
Свойства property	Функция-свойство.
text	text-функция для работы с текстом.

Определение типов и глобальных переменных

Команда type	Определение типа.
Команда global	Объявление глобальных переменных.

Прочие команды

Команда include	Подключение других файлов на языке Gentee.
Команда import	Подключение импортируемых функций из DLL.
Команды public и private	Определение области видимости.

Вот пример простейшей программы.

```
/* Example */

define
{
    NAME = "John"
}

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

Комментарии. Замена символов

В процессе работы компилятор удаляет все комментарии, производит подстановку значений макросов и заменяет символы форматирования.

`/*...*/` Комментарий может быть в любом месте программы. Он должен начинаться с `/*` и заканчиваться `*/`.

`//` Строчный комментарий. Исключается текст от текущей позиции до конца строки.

```
/*
```

```
  This is a comment.
```

```
*/
```

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

`;` Перенос строки является разделяющим символом между выражениями и операторами. Точка с запятой заменяется на перенос строки. Вы можете использовать этот символ, если хотите разместить несколько операторов в одной строке.

`:` Двоеточие заменяется на открывающую фигурную скобку, а в конце текущей строки вставляется закрывающая фигурная скобка.

```
// These examples are equal
```

```
if a == 10 : a = b + c; c = d + e
```

```
if a == 10
```

```
{
```

```
  a = b + c
```

```
  c = d + e
```

```
}
```

Команда define

Команда **define** предназначена для описания макросов. Макросу можно присвоить значение одного из следующих типов: число, строка, двоичные данные или имя идентификатора. Также макросу можно присвоить значение другого макроса. В дальнейшем, для подстановки его значения, имя макроса должно быть указано как **\$имямакроса** или **\$имямакроса\$**. Макрос может быть переопределен в другом **define**. Макросы описываются внутри фигурных скобок, на каждой строке может быть описан один макрос. Для инициализации макроса, после его имени ставится = и соответствующая константа или выражение. Мы рекомендуем использовать в именах макросов только заглавные буквы.

define

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

Атрибуты

Вы можете указать у **define** атрибуты **export** и **namedef**. Используйте атрибут **export** если вы распространяете модуль в виде байт-кода (.ge файл) и хотите чтобы эти макросы можно было использовать в других программах. Если у **define** определения указан атрибут **namedef**, то все его макросы можно использовать без указания символа '\$'.

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
}
```

Определение имени define

Вы можете указать имя у **define** определения. В этом случае обращение к макросам возможно как напрямую, так и с указанием define имени. Это сделано для избежания конфликтов между макросами из разных модулей. В этом случае обращение к макросу имеет следующий вид: **\$имяdefine.имямакроса**.

```
// file1.g
define myflag< export >
```

```
{
  FLAG1 = 0xFFFF0
  FLAG2 = 0xFFFF1
}
```

```
// file2.g
```

```
define flags
{
  FLAG1 = 0x0001
  FLAG2 = 0x0002
}
```

func uint my(uint param)

```
{
  if param & $myflag.FLAG1
  { ... }
  if param & $flags.FLAG1
  { ... }
}
```

Перечисление

В Gentee нет отдельной команды для определения перечислений. Вы можете использовать для этого команду **define**. Если макросы не присвоено никакое значение, то его значение становится на единицу больше значения предыдущего макроса. Если предыдущий макрос отсутствует или не целое число, то значение текущего макроса становится 0. В случае перечисления макросы могут разделяться пробелами.

define

```
{  
    VAL0 VAL1 VAL2 // VAL2 = 2  
  
    ID1 = 100  
    ID2 ID3 ID4  
    ID5 // ID5 = 104  
}
```

Выражения

Макросам могут быть присвоены не только числа, но и результаты выражений. Операндами выражений могут быть или константы или другие макросы. Вы можете посмотреть полный список возможных операций на странице [Макровыражения](#).

define

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

Смотрите также

- [Команда ifdef](#)
- [Макросы](#)
- [Макровыражения](#)

Команда `ifdef`

Команда условной компиляции `ifdef` позволяет включать или исключать некоторые части программы для компиляции в зависимости от каких-то условий. После ключевого слова `ifdef` должно следовать выражение-условие, а затем в фигурных скобках находится часть программы, которую нужно откомпилировать если условие выполняется (не равно 0). В качестве условия может использоваться выражение состоящее из макросов и констант (число, строка, набор данных). Посмотреть все возможные операции для выражений можно на странице [Макровыражения](#).

В примере ниже функция `myfunc` будет компилироваться если макрос `$MODE` является числом не равным нулю или не пустой строкой.

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

`ifdef` может использоваться не только на верхнем уровне вложенности, но и внутри любой другой команды и даже внутри выражений. Кроме этого допускается вложенность команд `ifdef` друг в друга.

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

`elif` и `else`

Если условие ложно и в этом случае необходимо откомпилировать другую часть программы, то используется дополнительная команда `else`. Если имеется более чем два варианта компиляции, то можно использовать команду `elif` с дополнительным условием. Может быть несколько команд `elif` подряд, а в конце команда `else`.

`define`

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

Смотрите также

- [Команда `define`](#)
- [Макросы](#)
- [Макровыражения](#)

Макровыражения

При определении макросов с помощью [Команда define](#) и в [Команда ifdef](#) можно использовать простейшие выражения с константами и макросами. Операнды должны иметь одинаковый тип, за исключением логических операций `&&` и `||`. Возможно использование круглых скобок для указания порядка вычисления выражения.

Операция	Типы операндов	Тип результата
Арифметические		
+	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
Битовые		
&	int uint long ulong	int uint long ulong
	int uint long ulong	int uint long ulong
^	int uint long ulong	int uint long ulong
Логические		
&&	int uint long ulong float double str(1 если размер >0) buf(1 если размер >0)	int uint
	int uint long ulong float double str(1 если размер >0) buf(1 если размер >0)	int uint
Операции сравнения		
==	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
Унарные операции		
+	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 если размер >0) buf(1 если размер >0)	int uint

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

Смотрите также

- [Команда define](#)
- [Макросы](#)
- [Команда ifdef](#)

Команда include

Команда **include** предназначена для подключения дополнительных файлов с исходным кодом на языке Gentee или уже откомпилированным байт-кодом. Вы можете подключать готовые библиотеки и использовать после этого их функции или собирать несколько ваших модулей в один проект. Если вы указываете файл с расширением **.ge**, который содержит откомпилированный байт-код, то он добавляется без дополнительной компиляции. В случае, если какой-то файл добавляется более одного раза, то компилятор проигнорирует повторные включения файла.

Подключаемые файлы перечисляются внутри фигурных скобок по одному на каждую строку или должны быть разделены запятой. Можно указывать как абсолютные, так и относительные пути к файлам. Имена файлов представляют собой [строки](#), поэтому необходимо удваивать символ '\' или ставить впереди кавычек '\$'.

include

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

Команда **include** может использоваться в любом месте программы и в любых Gentee файлах. Можно использовать **include** внутри команды **ifdef**.

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

Вы можете настроить профили компилятора таким образом, чтобы у вас все время включались какие-то определенные файлы и тогда их не надо определять с помощью **include**. Также в профиле вы можете перечислить директории для поиска файлов. В этом случае вам достаточно будет указывать только имена файлов в команде **include** и компилятор сам найдет их в этих директориях.

Команда import

Команда **import** позволяет подключить импортируемые функции из DLL. После ключевого слова **import** следует имя DLL-файла, из которого будут импортироваться функции, затем открывается блок с описаниями. В каждой строке блока должно быть описание импортируемой функции - имя типа возвращаемого значения (если есть), затем имя функции и в круглых скобках имена типов параметров через запятую. Можно определить новое имя функции, которое будет использоваться вместо импортируемого имени. Для этого после описания ставится **->** и новое имя функции. После импортирования, вызов DLL функции ничем не отличается от вызова функции написанной на языке 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
}
```

Если вы будете запускать Gentee программу из собственного EXE файла, то вы можете использовать функции из EXE модуля. Для этого, укажите имя DLL файла в виде пустой строки, а про передачу адресов импортируемых функций читайте в разделе **Настройка и запуск Gentee**.

Атрибуты

cdecl

Означает, что импортируются **__cdecl** функции. По умолчанию считается, что импортируемые функции являются **__stdcall** функциями.

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

link

В этом случае, dll файл будет включен в .ge, во время запуска запишется во временную директорию и оттуда уже будет загружен программой. По завершению работы он уничтожится. То есть, этот атрибут полезен, если вы не хотите распространять дополнительные dll файлы вместе с .ge, но не уверены, что они будут на машине пользователя. Желательно указывать полный путь к dll файлу.

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

exe

Используйте этот атрибут если вы знаете только путь DLL относительно вашего приложения. Вот пример, когда my.dll будет подключаться из поддиректории *Plugins* где запущена программа.

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

Команды public и private

По умолчанию, все функции, методы, типы и прочие элементы языка являются общедоступными после их описания. Если вы не хотите, чтобы какие-то элементы были доступны за пределами файла, в котором они описаны, то используйте команду **private**. Все элементы языка, которые идут после этой команды, будут видны только до конца компиляции текущего .g файла. После этого имена этих элементов будут удалены и к ним невозможно будет обратиться по имени. Команда **public** восстанавливает режим общего доступа для последующих элементов. Вы можете чередовать public и private как вам нужно в ваших исходных текстах. Действие этих команд распространяется на функции, методы, операторы, типы, глобальные переменные.

```
private
```

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

```
public
```

```
func str myfunc  
{  
    ...  
    mylocal()  
}
```

Типы и переменные

Gentee - язык строгой типизации, поэтому типы занимают очень важное место при программировании на Gentee. Все типы можно разбить на две три группы: **числовые** типы, **структурные** типы и тип **reserved**.

Числовые типы

Все числовые типы встроены в язык. **uint** является самым распространенным числовым типом. В языке Gentee нет указателей и логического типа, их функции выполняет тип **uint**. Типы **byte**, **ubyte**, **short**, **ushort** при выполнении арифметических операций рассматриваются как типы **int** или **uint**, в зависимости от знака. Если вы будете указывать их в качестве полей в структурных типах, то они будут занимать соответствующее количество байт.

Имя типа	Размер	Минимум	Максимум	Примечание
Целочисленные типы				
byte	1(4)	-128	+127	знаковый
ubyte	1(4)	0	+255	беззнаковый
short	2(4)	-32768	+32767	знаковый
ushort	2(4)	0	+65535	беззнаковый
int	4	-2147483648	+2147483647	знаковый
uint	4	0	+4294967295	беззнаковый
long	8	-2 ⁶³	+2 ⁶³ - 1	знаковый
ulong	8	0	+2 ⁶⁴ - 1	беззнаковый
Числа с плавающей точкой				
float	4	(+ или -)10E-37	(+ или -)10E38	;
double	8	(+ или -)10E-307	(+ или -)10E308	

Структурные типы

Структурные типы описываются командой **type**. При этом типы строка (**str**), буфер (**buf**), коллекция (**collection**) встроены в язык. Много типов определено в стандартной и прочих библиотеках (массивы, хэш-таблицы и т.д.).

Тип reserved

Тип **reserved** является особым типом и не принадлежит ни к числовым, ни к структурным типам. Он представляет собой массив байтов и описывается и используется как массив. Его особенность в том, что резервируется область памяти в том месте, где он описан. Например, можно указать поле в структуре *reserved field[50]*. Это означает, что в структуре будет зарезервирована область размером 50 байт. Если мы опишем это же самое внутри функции, то мы отведем 50 байт в стеке для данной локальной переменной. Резервируемый размер ограничен 65535 байтами. Для указания размера нельзя использовать выражение, в квадратных скобках должно стоять число.

Команда type

Структурные типы описываются с помощью команды **type**. После команды следуют имя типа, атрибуты, а затем описание полей в фигурных скобках. В каждой строке описания может быть определено одно или несколько полей одного типа. В начале идет имя типа, потом имена полей через пробел или запятую. Поле может иметь как числовой тип, так и описанный ранее структурный тип. В памяти поля структуры располагаются в таком порядке, как они были описаны. Если поле является структурой, то эта структура полностью вставляется в данный тип. При описании полей, могут быть определены размерности (в квадратных скобках через запятую) и тип элемента, который указывается после ключевого слова **of**. Для получения или присваивания значения поля у переменной необходимо указать через точку его имя.

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

Атрибуты

index

Типы могут содержать в себе другие элементы, например массив строк. Вы можете по умолчанию указать элементы какого типа содержит объект данного типа. Для этого присвойте этому атрибуту соответствующий тип. Если элементы по умолчанию имеют этот же самый тип (например дерево), то укажите **index = this**.

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

inherit

Можно осуществлять наследование типов. Для этого необходимо использовать атрибут **inherit = имятипа**. Подробнее смотрите в [Наследование типов](#).

protected

Имеется возможность запрещать обращение к полям типа из других модулей. Для этого необходимо указать атрибут **protected**. В этом случае все поля типа будут доступны только до конца компиляции текущего файла. После этого обращение к полям данного типа будет невозможно.

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

Дополнительные возможности

Для любого структурного типа вы можете определить методы, которые позволят вам

- Производить дополнительные действия при инициализации и уничтожении переменной
- Указывать **of** при описании переменных данного типа
- Использовать квадратные скобки при обращении к отдельным элементам
- Использовать **foreach** для перебора элементов данного типа.

Эти методы описаны на странице [Системные методы для типов](#).

Смотрите также

- [Наследование типов](#)
- [Системные методы для типов](#)

Наследование типов

Gentee позволяет наследовать структурные типы. Для этого необходимо при описании типа указать атрибут **inherit** с указанием родительского типа.

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

Если новый тип не имеет дополнительных полей, то указываются пустые фигурные скобки или двоеточие.

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

Нельзя наследовать базовые числовые типы и тип *reserved*. При наследовании типов у вас сохраняется возможность обращаться к полям родительских типов.

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

Также при наследовании сохраняется возможность вызывать методы или функции в всех родительских типах. Компилятор при вызове метода, функции или оператора сам находит наиболее подходящую функцию или метод. Например, пусть имеются следующие функции

```
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")
}
```

Пусть

```
my mm
```

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

тогда первый вызов *print* выведет **MYTYPE PARAMETER = 20**, а второй *print* выведет **MY = 0**, а не **MYTYPE = 0**. Аналогичная ситуация с методами и операторами. Если необходимо вызвать именно родительский метод или функцию, то нужно привести переменную к родительскому типу. **print(mm->mytype)** выведет **MYTYPE = 0**. Таким образом **Gentee** предоставляет вам такие основные возможности объектно-ориентированного программирования как **наследование** и **полиморфизм**.

Смотрите также

- [Команда type](#)

Системные методы для типов

Для каждого типа можно определить методы, которые позволят упростить работу с переменными данного типа и расширит его возможности. Рассмотрим абстрактный тип.

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

Инициализация

В Gentee происходит автоматическая инициализация переменных и полей любого типа. Если вы хотите произвести дополнительные действия при инициализации переменной типа, то определите метод **init**. Следует заметить, что все числовые поля инициализируются нулями, поля других типов также инициализируются в соответствии с описаниями тех типов. Например, если поле имеет тип **str**, то оно сразу будет инициализировано пустой строкой.

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

Уничтожение

Если вы перед уничтожением переменной данного типа хотите произвести дополнительные действия, то укажите их в методе **delete**.

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

Использование оператора of

Предположим, переменная данного типа может содержать элементы другого типа. В этом случае вы должны иметь возможность указать это при описании переменной. Например, **test mytest of double**. Для того чтобы компилятор мог понять оператор **of** необходимо определить метод **oftype**. У него должен быть параметр, в котором будет передан тип элементов.

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

Указание размера и размерности

Предположим, что при описании переменной вы сразу хотите создать несколько элементов, а также указать размерность данной переменной. Например, **test mytest[10,20] of double**. Для этого вы должны описать по одному методу **array** для каждой возможной размерности.

```
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
}
```

Обращение по индексу

Если вы хотите получить *i*-й элемент переменной данного типа с помощью квадратных скобок, то вам следует описать по одному методу **index** для каждой размерности. В качестве индекса можно указывать не только числа, но любые другие типы. Для этого достаточно определить метод **index** с параметром соответствующего типа. Обратите внимание, что метод **index** должен **возвращать указатель** на найденный элемент!

```

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" ])" )

```

Использование оператора foreach

В языке Gentee имеется оператор **foreach**, который перебирает все элементы у переменной определенного типа. Если вы хотите использовать этот оператор для вашего типа, то вы должны определить методы **eof**, **first**, **next** с параметром **fordata**. В поле **icur** у **fordata** хранится индекс текущего элемента при переборе. Вы должны обнулить его в методе **first** и увеличивать в методе **next**.

```

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
}

```

Переопределение операторов

Вы можете использовать всевозможные операции **=**, **+**, *****, **==**, **!=**, ***** и т.д. для переменных любого типа. Для этого необходимо описать соответствующие команды **operator**. Подробнее это рассматривается на странице [Переопределение операций operator](#).

```

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 }
```

Смотрите также

- [Команда type](#)
- [Переопределение операций operator](#)
- [Конструкция цикла foreach](#)
- [Определение метода method](#)

Команда `global`

Глобальные переменные объявляются командой `global`. После команды идут фигурные скобки внутри которых перечисляются переменные. Вы должны указать в начале тип переменной, а затем имя переменной. Переменные одного типа можно перечислять через запятую или пробел в одной строке. Например

`global`

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

Если тип переменной поддерживает использование `of` и квадратных скобок, то вы можете указать эти дополнительные параметры при описании глобальной переменной. Кроме этого, числовые переменные, а также строки `str` и двоичные данные `buf` могут быть инициализированы в момент описания с помощью операции присваивания '='. При инициализации переменных можно использовать [макровыражения](#). По умолчанию, переменная инициализируется нулями или вызовом соответствующей функции инициализации.

Вы можете обращаться к глобальной переменной с момента ее объявления в последующих функциях и методах.

`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" }
}
```

Смотрите также

- [Макровыражения](#)
- [Системные методы для типов](#)

Локальные переменные

Локальные переменные служат для временного хранения промежуточных результатов во время выполнения функции или метода. Локальная переменная может быть объявлена в любом месте тела функции, в том числе и во вложенных блоках ограниченных фигурными скобками. Объявление начинается с новой строки, указывается имя типа, затем через запятую или пробел следуют имена переменных данного типа.

Если тип переменной поддерживает использование **of** и квадратных скобок, то вы можете указать эти дополнительные параметры при описании локальной переменной. Кроме этого, числовые переменные, а также строки **str** и двоичные данные **buf** могут быть инициализированы в момент описания с помощью операции присваивания '='. При инициализации переменных можно использовать любые выражения. По умолчанию, переменная инициализируется нулями или вызовом соответствующей функции инициализации.

```
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" }
    ...
}
```

Область видимости

Область видимости локальной переменной ограничена от места объявления до конца блока, в котором она объявлена, при этом она видна во вложенных блоках. Глобальные и локальные переменные могут быть переопределены, т.е. внутри нового блока может быть объявлена переменная с именем уже существующей переменной. Новая переменная может иметь другой тип. До конца текущего блока будет доступна только последняя переменная, а объявленная раньше становится невидимой. После завершения блока происходит возврат к старой переменной. Все объекты описанные как локальные переменные автоматически создаются при входе в блок и уничтожаются при выходе. Можно создавать объекты с помощью служебной функции **new**. В этом случае программисту надо следить за удалением объектов с помощью функции **destroy**. Так как, при выходе из функции происходит уничтожение локальных переменных, то вы можете возвращать только числовые локальные переменные.

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

Смотрите также

- [Возвращение переменных](#)
- [Системные методы для типов](#)

Определение функции func

Функция состоит из двух частей: описание и тело функции. Описание функции начинается с ключевого слова **func**, затем идут возвращаемый тип, имя функции, атрибуты функции в угловых скобках и описание параметров функции в круглых скобках. Обязательным является только имя функции. Если не указано имя возвращаемого типа, то функция ничего не возвращает.

Тело функции или метода - это все, что заключено в фигурные скобки идущие после описания функции. Тело функции может содержать [подфункции](#), выражения, конструкции и описания [локальных переменных](#).

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

Атрибуты

entry

Этот атрибут указывается у функций, которые должны быть запущены автоматически до вызова главной функции.

main

Этот атрибут указывается у главной функции с которой начнется выполнение программы. Если функций с таким атрибутом несколько, то вызовется последняя функция с атрибутом **main**. Главная функция запускается после вызова всех **entry** функций. Функции имеющие атрибут **main** или **entry** не должны иметь параметров.

```
func uint myprog<main>
```

```
{
    print("Hello, World!")
    getch()
}
```

result

Gentee не позволяет возвращать структурный тип из функции, если он принадлежит (описан внутри) этой функции.

Этот атрибут позволяет обойти это ограничение. Подробнее о его использовании написано на странице [Возвращение переменных](#).

alias

Если вам необходимо где-то получить и передать идентификатор функции, метода или оператора, то вы можете воспользоваться этим атрибутом. Так как функции и методы могут иметь одни и те же имена, но разные параметры, то нахождение необходимой функции может привести к некоторым трудностям. Вы можете присвоить атрибуту **alias** имя и использовать этой имя в качестве переменной-идентификатора функции.

```
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()
}
```

Параметры

Параметры функции описываются такой последовательностью: имя типа, имена параметров с данным типом, далее запятая или пробел и опять имя типа и параметры. Если функция не имеет параметров, то в описании указываются пустые скобки или скобки вообще не ставятся. Можно определять функции с одним и тем же именем, но с разными параметрами. В этом случае при вызове функции компилятор по типам указанных параметров определит подходящую функцию.

При описании параметров можно использовать квадратные скобки для указания размерности и оператор **of**. При описании таких параметров не нужно указывать в квадратных скобках конкретное количество элементов.

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

Обращение к параметрам ничем не отличается от обращения к локальным переменным. Все **числовые типы** передаются в функцию или метод **по значению**. То есть вы можете без всяких последствий изменять значение параметра. Все **структурные типы** передаются **по ссылке**. В этом случае, все сделанные вами изменения будут происходить с оригинальной переменной, которую вы передали в качестве параметра.

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

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

Смотрите также

- [Возвращение переменных](#)
- [Локальные переменные](#)
- [Подфункции subfunc](#)

Определение метода `method`

Для любых типов языка можно определять **методы**. Метод - это функция привязанная к объекту некоторого типа и обычно она осуществляет какие-либо действия с этим объектом. Определение метода состоит из ключевого слова **method**, имени возвращаемого типа (если он есть), типа объекта и через точку имени метода. Далее идут параметры и тело метода как у функции. Описание параметров метода аналогично описанию параметров функции. Вызов метода аналогичен получению значения поля с указанием параметров в круглых скобках: **объект.имяметода(параметры)**.

Внутри метода автоматически создаётся параметр **this**. Этот параметр содержит объект для которого вызывается данный метод. Тип параметра **this** совпадает с типом объекта.

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

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

У метода можно указывать атрибуты **result** и **alias** как у [функций](#). Методы отвечают за инициализацию объектов, их уничтожение, взятие индексов и конвертацию типов, а также для других целей. Более подробно об этом можно прочитать на странице [Системные методы для типов](#).

Конвертация типов

С помощью методов можно описывать конвертацию типов. Для этого в качестве исходного типа указывается тип объекта, а конечный тип, в который следует сконвертировать объект, представлен именем метода. Если конечный тип является структурой, то необходимо использовать атрибут **result**.

```
// 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 )
}
```

Смотрите также

- [Определение функции `func`](#)
- [Системные методы для типов](#)

Переопределение операций operator

Язык **Gentee** позволяет для объектов вводить операции с использованием уже существующих операторов (=, ==, +=, +, *, <, == и т.д.). При этом приоритет операторов остаётся неизменным. Обработка операций осуществляется с помощью специальных функций-операторов начинающихся с ключевого слова **operator**. Далее идет тип результата операции, символьное представление оператора и один или два параметра, в зависимости от того бинарная или унарная операция. Тип параметров совпадает с типом операндов и в параметрах будут значения операндов. Если операция бинарная, то первый параметр представляет левый операнд, а второй правый. Операнды могут быть разного типа. Если результатом является новый объект (например при сложении), то необходимо использовать атрибут **result**. Так же вы можете использовать атрибут **alias**, если это необходимо.

Если Вы хотите определить операторы сравнения для своего типа, то вам достаточно определить операторы ==, < и >. Операторы !=, >=, <= не требуют определения и автоматически приводятся компилятором к ==, < и >.

```
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 )
}
```

Смотрите также

- [Определение функции func](#)

Определение text функции

Специально для работы с текстовыми данными имеется команда **text**. Она позволяет генерировать текст любой сложности и объема.

Описание начинается с ключевого слова **text**, затем идут атрибуты в угловых скобках и описание параметров в круглых скобках. Обязательным является только имя text функции. Text функция не может возвращать значение. Описание атрибутов и параметров идентично [описанию функций](#). Выводимый текст начинается с новой строки после ее описания и продолжается до конца файла или до комбинации **!**.

Главным отличием **text функций** от функций является то, что здесь не строки встраиваются в исходный код, а исходный код встраивается в текст. Задачей text функции является вывод текста на консоль или в строку. Тип действия определяется при вызове text функции.

Вывод на консоль

Вывод на консоль осуществляется с помощью унарной операции **@**.

@name textfunc(параметры)

Вывод в строку

Вывод в строку осуществляется с помощью бинарной операции **@** где слева указана строка для вывода. Результат text функции будет дописываться в строку.

stemp @ name textfunc(параметры)

Дополнительные возможности

Внутри текста действительны все команды со служебным символом что и в [строке](#). Кроме этого имеются следующие дополнительные команды.

! Конец text функции. По умолчанию, text функция идет до конца файла.

\@name(...) Вызов другой text функции. При вызове сохраняется тип текущего вывода (консоль или строка).

{...} Вставка блока кода. Внутри фигурных скобок вы можете размещать исходный код как в теле функции. Этот блок кода соответствует блоку кода первого уровня у функции и в нем можно описывать подфункции. Чтобы вывести строку в текущий вывод text функции из блока кода необходимо использовать операцию **@"string"**

```
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()
}
```

Текущий вывод

Можно получить текущую строку вывода с помощью использования **this**. Если **this** равно нулю, то значит текущий вывод осуществляется на консоль.

Смотрите также

- [Определение функции func](#)
- [Строки](#)

Свойства property

Gentee позволяет обращаться и устанавливать значения полей структурных типов через свойства **property**. С помощью свойств вы можете скрывать прямое обращение к полям и производить дополнительные вычисления при обращении или изменении значения поля с помощью операции присваивания. Имя свойства не должно совпадать с именем поля, так как прямое обращение к полю имеет более высокий приоритет, и вместо свойства будет получено или установлено значение поля.

Свойство **get**, которое возвращает значение не должно иметь параметров.

```
type mytype
{
    str val
}

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

Свойство **set**, которое устанавливает значение должно иметь один параметр. Свойство **set** также может возвращать значение.

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

Вызов свойств происходит при указании имени свойства как поля у переменной. Свойство **set** вызывается при указании его слева от операции присваивания, в остальных случаях вызывается свойство **get**.

```
func myfunc
{
    mytype myt

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

Смотрите также

- [Определение функции func](#)

Команда extern

Вы не можете в **Gentee** вызвать какую-либо функцию, если вы ее еще не определили. Команда **extern** позволяет предварительно описать функцию, метод, свойство или оператор. Эта команда позволяет вам вызывать функцию до ее реального определения. Например это может быть рекурсивный вызов одной функции из другой. В дальнейшем указанные функции могут быть описаны даже в другом файле.

После ключевого слова **extern** открывается блок с описаниями функций внутри фигурных скобок. В каждой строке блока должно быть описание функции, метода, оператора или свойства без тела.

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
}
```

Смотрите также

- [Определение функции func](#)

Подфункции `subfunc`

В теле функции могут быть определены подфункции с помощью конструкции `subfunc`. Подфункцию можно определить только на первом уровне вложенности тела функции. Подфункцию можно вызывать только внутри тела функции или из других подфункций данной функции. Внутри подфункции нельзя определить ещё одну подфункцию и рекурсивно вызывать саму себя, так как локальные переменные подфункции являются статическими. Подфункция может перекрывать имена других функций. Вызов подфункции ничем не отличается от вызова обычной функции, также аналогично описание подфункции и описание её параметров, за исключением того, что отсутствуют атрибуты. В подфункции можно обращаться к локальным переменным функции.

Подфункции очень полезны когда вам надо выполнить внутри функции один и тот же код, но вы не хотите оформлять его в виде отдельной функции.

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

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

Смотрите также

- [Определение функции `func`](#)

Возвращение переменных

Gentee не позволяет возвращать локальные переменные из функций и методов если они **не являются числовыми** типами. Все структурные локальные переменные уничтожаются при выходе из функции. Например, вызов следующей функции приведет к ошибке.

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

В таких ситуациях можно использовать атрибут **result**. Он дает возможность возвращать результирующее значение структурного типа из функций или методов. В некоторых случаях это позволяет избежать определение и передачу лишних локальных переменных. При наличии этого атрибута Вы можете работать с переменной с именем *result*, которая будет возвращена при выходе из функции. У функции с атрибутом **result** или не должно быть команды **return** или может быть пустой **return**.

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

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

Следует заметить, что в действительности при вызове такой функции или метода происходит создание временной переменной в вызывающем блоке. Эта переменная передается в функцию и является там переменной с именем **result**.

Смотрите также

- [Определение функции func](#)

Конструкции языка

Внутри тела функции (метода, оператора, свойства), могут быть специальные конструкции, которые позволяют изменить последовательное выполнение программы. Некоторые конструкции могут содержать в себе блоки, в которых также могут быть другие конструкции. Все конструкции можно разделить на несколько типов.

Конструкции условного перехода

[if-elif-else](#) Конструкция условия.

[switch](#) Конструкция выбора.

Конструкции цикла

[while-do](#) Простой цикл.

[do-while](#) Простой цикл, с нижней проверкой условия.

[for](#) Цикл с описанием инициализации и приращения.

[fornum](#) Цикл с фиксированным условием и автоматическим приращением.

[foreach](#) Цикл перебора элементов объекта.

Инструкции безусловного перехода

[return](#) Выход из функции.

[break](#) Выход из цикла.

[continue](#) Досрочный переход на следующий этап цикла.

[label](#) Описание метки.

[goto](#) Переход на метку.

Прочие конструкции

[with](#) Сокращенное обращение к полям типа.

Конструкция условия if-elif-else

Условная конструкция состоит из следующих частей:

if

if часть содержит ключевое слово **if**, выражение-условие и блок выражений, который будет выполняться в случае истинности условия. Если условие не выполняется, то управление передаётся следующей части **elif**.

elif

elif часть содержит ключевое слово **elif**, выражение-условие и блок выражений, который будет выполняться в случае истинности условия. Конструкция может содержать несколько **elif** частей следующих друг за другом.

else

else часть содержит только ключевое слово **else** и блок выражений, который будет выполняться, в том случае, если не выполнилось ни одно условие в **if** и **elif** частях.

elif и **else** операторы являются необязательными.

Выражение-условие должно возвращать числовое значение. **ИСТИНОЙ** считается число отличное от 0.

```
//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
```

Смотрите также

- [Конструкции языка](#)

Конструкция выбора switch

Конструкция **switch** позволяет выполнить разные действия при разных значениях выражения. После ключевого слова **switch** следует исходное выражение, которое вычисляется и запоминается как **switch**-значение. Затем в фигурных скобках перечисляются конструкции **case** с всевозможными значениями и исходным кодом, который необходимо выполнить. У одного **case** может быть определено через запятую несколько возможных значений при которых он будет выполнен. После выполнения **case** блока с подходящим значением программа переходит к следующему за **switch** оператору. Остальные **case** блоки не проверяются.

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

Если вы хотите выполнить какие-то действия в случае, если не один **case** блок не был выполнен, то вставьте в конец **switch** конструкцию **default**. Может быть только одна конструкция **default** и она должна стоять последней после всех **case**.

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

Дополнительные возможности

Конструкцию **switch** можно использовать не только для числовых выражений, но и для любых типов поддерживающих операцию сравнения **==**.

Наравне с **case** можно использовать метку **label** для безусловного перехода внутри **switch**. Метка установленная ниже ключевого слова **case**, позволяет зайти в соответствующий **case**-блок из другого блока **case**.

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

Смотрите также

- [Конструкции языка](#)
- [Инструкции label, goto](#)

Конструкции цикла while и do

while

Конструкция **while** представляет собой простейший цикл. Конструкция состоит из ключевого слова, выражения-условия и тела цикла (блока выражений). Тело цикла будет выполняться до тех пор, пока условие цикла не равно нулю. Тело цикла не выполнится ни разу, если при первой проверке условие равно 0.

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

do-while

Конструкция **do-while**, содержит ключевое слово **do**, тело цикла, ключевое слово **while** и выражение-условие. В этом операторе тело цикла также выполняется пока условие не равно 0. Отличие от конструкции **while** состоит в том, что проверка условия происходит после выполнения тела и цикл выполняется хотя бы один раз.

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

Существуют специальные операторы для выхода из цикла, когда это необходимо. Подробнее об этом смотрите на странице [Инструкции return, break, continue](#).

Смотрите также

- [Конструкции языка](#)
- [Инструкции return, break, continue](#)

Конструкции цикла for и fornum

for

Конструкция **for** состоит из ключевого слова **for**, последовательности трёх выражений разделённых запятыми и тела цикла.

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

exp1 - необязательное выражение инициализации. Оно обычно служит для присваивания начального значения переменной-счетчику.

exp2 - выражение-условие. Цикл выполняется пока это условие не равно 0.

exp3 - необязательное выражение приращения. Оно, как правило, увеличивает или уменьшает значение счетчика.

Приведённую конструкцию можно представить с помощью цикла [while](#)

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

Приведенные ниже примеры делают одни и те же действия.

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

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

fornum

Если индекс цикла увеличивается на единицу и максимальное значение счетчика можно определить до начала цикла, то вместо **for** можно использовать конструкцию **fornum**.

После ключевого слова **fornum** следует имя переменной счетчика, затем может стоять операция присваивания и выражение - начальное значение счетчика. Если операция присваивания отсутствует, то начальное значение счетчика остаётся неизменным. В качестве счетчика может выступать только переменная целочисленного типа. Через запятую записывается выражение, результат которого определяет выход из цикла. Это выражение вычисляется один раз перед началом цикла. Цикл выполняется пока значение счетчика меньше значения этого выражения. После выражения следует тело цикла. Операция увеличения счетчика на 1 добавляется компилятором автоматически в конце тела цикла.

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

Смотрите также

- [Конструкции языка](#)
- [Инструкции return, break, continue](#)

Конструкция цикла `foreach`

Цикл `foreach` предназначен для работы с объектами содержащими какое-то множество элементов. Тип объекта должен иметь методы `first`, `next`, `eof`. Подробнее об определении этих методов смотрите на странице [Системные методы для типов](#). С помощью конструкции `foreach` можно перебрать все элементы в исходном объекте.

После ключевого слова `foreach` указывается имя переменной, которая будет указывать на очередной элемент. Затем через запятую следует объект в котором будет перебор и далее тело цикла. Если объект содержит элементы числового типа, то переменная-индекс, будет содержать значения. Если объект состоит из элементов структурного типа, то переменная-индекс будет указывать на очередной элемент. В этом случае, если вы изменяете переменную-индекс, то также будет меняться соответствующий элемент в объекте.

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

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

Смотрите также

- [Конструкции языка](#)
- [Системные методы для типов](#)

Инструкции return, break, continue

return

Инструкция **return** предназначена для возврата значения функции или экстренного выхода из функции. Выход может быть из любого места тела функции, в том числе из циклов и вложенных блоков. Если функция возвращает значение, то инструкция **return** должна обязательно присутствовать, и должна содержать выражение соответствующего типа.

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

break

Инструкция **break** используется для выхода из циклов. **break** может быть внутри вложенных блоков. Если есть несколько вложенных циклов, то произойдёт выход из текущего цикла.

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

continue

Инструкция **continue** действует внутри циклов и позволяет перейти к выражению изменения счетчика (для циклов for, fornum, foreach) или к выражению условию (для циклов while и do-while) не выполняя до конца тело цикла. Инструкция действует на текущий цикл в случае вложенных циклов.

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

Смотрите также

- [Конструкции цикла while и do](#)
- [Конструкции цикла for и fornum](#)
- [Конструкция цикла foreach](#)

Инструкции `label`, `goto`

Инструкции `label` и `goto` обеспечивают безусловный переход внутри тела функции.

`label`

Инструкция `label` предназначена для определения меток. После ключевого слова `label` должно стоять имя - идентификатор метки. Метки указывают на места перехода для команды `goto`. Область видимости метки ограничена блоком, в котором она определена, поэтому можно перейти на метку находящуюся в текущем или в блоках высшего уровня. Переход на метку может встретиться раньше определения метки.

`goto`

С помощью инструкции `goto` можно перейти на указанную метку. После ключевого слова `goto` необходимо указать имя метки, с которой будет продолжено выполнения программы.

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

Конструкция with

Конструкция **with** позволяет упростить обращение к полям переменной структурного типа. Рассмотрим такой пример.

```
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
```

Как видно, нам приходится каждый раз указывать имя переменной. **with** позволяет опускать имя переменной внутри своего блока. Для этого укажите имя переменной после ключевого слова **with** и внутри фигурных скобок вы можете указывать только точку и имя соответствующего поля. Конструкции **with** могут быть вложенными друг в друга.

```
customer mycust
```

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

Арифметические операторы

Арифметические операторы можно разделить на три группы.

Арифметические операторы

+	Сложение. <code>10 + 34 = 44</code>
-	Вычитание. <code>100 - 25 = 75</code>
*	Умножение. <code>11 * 5 = 55</code>
/	Деление. При делении целых отбрасывается дробная часть. <code>10 / 3 = 3</code>
%	Остаток от деления. Операция <code>a % b</code> возвращает остаток от деления a на b или ноль, если деление происходит без остатка. Операция остаток от деления применима только к целочисленным типам. <code>14 % 4 = 2</code>
-(ун)	Унарная операция смены знака. Данная операция меняет знак у целых или действительных чисел. <code>-10 = -10</code>

```
a = ( 54 + b ) * ( ( 2*c - 235 ) / 3 )
```

```
b = a % 10 + 0xFF00
```

Операторы инкремента и декремента

Операторы `++` и `--` являются унарными и применимы только для целочисленных типов.

++	Оператор инкремента. Он имеет два вида записи префиксная <code>++i</code> и постфиксная <code>i++</code> . В префиксной записи сначала увеличивается значение переменной <code>i</code> на 1, а затем возвращается полученное значение, в постфиксной записи в выражение подставляется значение переменной <code>i</code> , а затем переменная увеличивается на 1.
--	Оператор декремента. Префиксная запись <code>--i</code> - значение переменной уменьшается на 1 и возвращается. Постфиксная запись <code>i--</code> - подставляется значение, а затем происходит уменьшение переменной.

```
i = ++k
```

```
while i++ < 100
```

```
{
```

```
    sum += 1--
```

```
}
```

Побитовые операторы

Побитовые операторы предназначены для работы с целочисленными типами.

&	Побитовое И, бинарный. <code>0x124 & 0x107 = 0x104</code>
^	Исключающее ИЛИ, бинарный. <code>0x124 ^ 0x107 = 0x23</code>
<<	Сдвиг влево, бинарный. Операции сдвига сдвигают влево или вправо левый операнд на количество битов указанных в правом операнде, освобождающееся место заполняется нулями. <code>0x124 << 2 = 0x490</code>
>>	Сдвиг вправо, бинарный. <code>0x124 >> 2 = 0x92</code>
~	Побитовое отрицание, унарный. <code>~0x124 = 0xFFFFFEDB</code>

```
a = b & 0x0020 + c | $FLAG_CHECK
```

```
rand=( 16807 * rand ) % 0x7FFFFFFF ) % ( end - begin + 1 ) + begin
```

Все эти операторы можно определить для переменных любого типа. Подробнее смотрите на странице [Переопределение операций operator](#).

Смотрите также

- [Переопределение операций operator](#)

Логические операторы

Логические операторы

Логические операторы предназначены для работы с целочисленными типами. Результатом работы логических операторов является число типа `uint` со значением **0** - результат операции **ЛОЖЬ** или **1** - результат операции **ИСТИНА**.

```
&& Логическое И, бинарный. Возвращается 0 если хотя бы один из операндов равен 0.  
|| логическое ИЛИ, бинарный. Возвращается 1 если хотя бы один из операндов равен 1.  
! Логическое отрицание, унарный. Возвращается 0 если операнд не ноль, и 1 если операнд равен 0.  
if a < 10 && ( b >= 10 || !c ) && k  
{  
    if a || !b  
    { ... }  
}
```

Операторы сравнения

Результатом работы операторов сравнения является число типа `uint` со значением **0** - результат операции **ЛОЖЬ** или **1** - результат операции **ИСТИНА**.

==	Равно.
!=	Не равно.
>	Больше.
<	Меньше.
>=	Больше или равно.
<=	Меньше или равно.

%<, %>, %<=, %>=, Операторы предназначены для альтернативного сравнения. Например, при использовании этих **%==, %!=** операторов для строк сравнение будет производиться без учёта регистра букв.

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

Все эти операторы можно определить для переменных любого типа. Подробнее смотрите на странице [Переопределение операций operator](#).

Операторы присваивания

Операторы присваивания являются бинарными операторами. Левый операнд должен быть переменной, элементом массива, полем структуры и т.д. Порядок вычисления справа налево.

<code>=</code>	Присвоить.
<code>+=</code>	Прибавить к значению переменной. <code>a += b => a = a + b</code>
<code>-=</code>	Вычесть из значения переменной. <code>a -= b => a = a - b</code>
<code>*=</code>	Умножить значение переменной. <code>a *= b => a = a * b</code>
<code>/=</code>	Разделить значение переменной. <code>a /= b => a = a / b</code>
<code>%=</code>	Получить остаток от деления. <code>a %= b => a = a % b</code>
<code>&=</code>	Произвести побитовое И. <code>a &= b => a = a & b</code>
<code>=)</code>	
<code>^=</code>	Произвести побитовое исключающее ИЛИ. <code>a ^= b => a = a ^ b</code>
<code>>>=</code>	Сдвиг вправо и присвоить. <code>a >>= b => a = a >> b</code>
<code><<=</code>	Сдвиг влево и присвоить. <code>a <<= b => a = a << b</code>

Как видите, кроме обычного присваивания, существуют операции с выполнением действия, т.е. после вычисления правого и левого операндов выполняется какая-либо бинарная операция и результат присваивается в левый операнд.

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

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

В одном выражении может быть несколько операций присваивания, так как все эти операции возвращают присвоенное значение. В этом случае вычисление будет идти справа налево.

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

Все эти операторы можно определить для переменных любого типа. Подробнее смотрите на странице [Переопределение операций operator](#).

Приведение типов

Оператор as

Оператор **as** может выполнять две функции: присваивать значение переменной и изменять тип. Оператор бинарный, вычисляется справа налево. Левый операнд обязательно должен быть локальной переменной типа `uint`. В зависимости от правого операнда может быть два варианта работы.

Первый вариант

Правый операнд - имя структурного типа. Значение локальной переменной не изменяется, но её тип временно меняется на указанный, предполагается, что переменная содержит адрес какого-либо объекта и к переменной теперь можно обращаться непосредственно как к объекту, без использования операции взятия значения `->`.

```
str mystr
uint a

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

Второй вариант

Правый операнд - выражение которое возвращает объект. Переменной присваивается адрес на данный объект и она меняет свой тип на тип этого объекта. Объект должен иметь тип отличный от числового.

```
str mystr
uint a

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

Тип переменной остаётся изменённым до конца текущего блока или до следующей операции **as** над этой переменной.

Оператор ->

Часто требуется просто указать, что переменная имеет определенный структурный тип. В этом случае можно использовать оператор `->` с именем требуемого структурного типа. Вместе с именем типа можно указывать размерность в квадратных скобках и тип элементов с помощью **of**. Переменная к которой применяется `->` может быть любого структурного типа, а не только `uint`.

```
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]
        }
    }
}
```

Конвертация типов

По умолчанию автоматически конвертируются друг в друга только целочисленные типы **byte, ubyte, short, ushort, int, uint**, для других типов следует использовать явную конвертацию. Для конвертации выражения в другой тип, пишется имя типа, в который будет произведена конвертация и само выражение в скобках, конвертация будет проведена только в том случае если исходный тип имеет соответствующий метод. Подробнее о реализации таких методов смотрите на странице [Определение метода method](#).

```
str a = "10"
uint b
```

```
b = uint( a )
```

Смотрите также

- [Определение метода method](#)
- [Поля и указатели](#)

Поля и указатели

Обращение к полям

Оператор `.` (точка) применяется для получения или установки значения поля или для вызова метода или [свойства](#). После точки необходимо указать имя поля или свойства. В случае вызова метода необходимо в круглых скобках указать параметры.

```
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()
```

Адреса и указатели

Унарный оператор `&` позволяет получить адрес локальной или глобальной переменной или идентификатор функции. Результат операции имеет тип `uint`. Если результатом какой-то операции является объект, например функция возвращающая строку, то к данному подвыражению также можно применить операцию взятия адреса. Оператор `&` примененный к объекту (структуре) возвращает адрес данного объекта и служит для приведения типа к `uint`.

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

Для получения значения по адресу необходимо использовать оператор `->`. Правым операндом должно быть имя числового типа, а левый операнд должен указывать на значение соответствующего числового типа.

```
int a = 10, b
uint addra

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

Операция взятия элемента объекта

Многие структуры или объекты могут содержать в себе элементы других типов. Для доступа к элементам объекта (элементы массива, символы строки) можно использовать квадратные скобки `[]`. Если объект является многомерным, то размерности разделяются запятыми. Отсчёт элементов начинается с нуля. Для того чтобы к переменной можно было применить данную операцию тип переменной должен иметь соответствующие методы `index`. Подробнее смотрите на странице [Системные методы для типов](#).

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

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

Смотрите также

- [Системные методы для типов](#)
- [Приведение типов](#)

Вызов функций и методов

Вызов функции и метода

Для вызова функции укажите имя функции и в круглых скобках перечислите через запятую параметры. Если параметров нет, то ставятся пустые скобки. Если функция или метод возвращают значение, то их можно использовать внутри выражения. Вызов метода выполняется по аналогии взятия поля и вызова функции, после переменной содержащей структуру ставится точка, затем имя метода и в скобках параметры

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

Вызов функции по адресу

В переменной типа **uint** может храниться адрес (идентификатор) функции. Для вызова функции по ее идентификатору используется операция **->func** и далее в скобках перечисляются параметры. В этом случае следует следить за количеством параметров и их типами, так как компилятор не может проверить совпадение параметров. Таким образом можно вызывать не только функции, но и методы и операторы.

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

Gentee также позволяет вызывать внешние функции по их адресу. Например, при динамическом подключении DLL библиотек, адрес функции можно получить с помощью Windows API функции **GetProcAddress**. Для вызова функции по адресу используется операция **->stdcall** и далее в скобках перечисляются параметры. Если функция имеет тип **cdecl**, то нужно использовать служебное слово **cdecl** вместо **stdcall**.

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

Вызов text функции

Вызов text функции осуществляется с помощью оператора **@**. Операция может быть как унарной, так и бинарной.

Унарная операция

@ name(...)

В случае вызова из функции func или метода, вывод text функции будет осуществляться на консоль. Если произошел вызов из text функции, то вывод будет осуществляться туда же куда он происходил у текущей text функции. При унарном вызове значок '@' можно опускать. То есть вызывать text функцию как обычную функцию.

Бинарная операция

dest @ name(...)

При использовании бинарной операции вызова, слева необходимо указать строку куда будет происходить вывод из text функции. Вывод в строку будет осуществляться в режиме добавления данных.

Вы можете использовать оператор **@** не только для вызова text функций, но и для вывода строк. Если в правой части будет стоять переменная или выражение типа строка, то будет осуществлен вывод этой строки на консоль или добавление к строке вывода, как при вызове text функции.

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

Смотрите также

- [Поля и указатели](#)
- [Определение функции func](#)
- [Определение метода method](#)
- [Определение text функции](#)

Условный оператор ?

Условный оператор ? аналогичен по своей работе конструкции **if-else**, но может использоваться внутри выражения. Он содержит три операнда-выражения. Операнды заключены в скобки и разделены запятыми, в начале вычисляется значение первого логического (целочисленного) выражения. Если значение не равно 0 (ИСТИНА), то вычисляется второе выражение и полученное значение становится результатом работы условного оператора. В противном случае вычисляется третий операнд и возвращается его значение.

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

Операция позднего связывания

Оператор ~ применяется для позднего связывания. Данный оператор во многом схож с использованием "точки" . - взятия поля или вызова метода, отличие состоит в том, что иногда на момент компиляции невозможно определить все возможные поля и методы объекта, а во время выполнения программы вызывается специальный метод объекта, которому передается имя поля/метода, типы и значения параметров. Технология позднего связывания часто применяется при работе с **COM объектами**.

Левым операндом операции ~ является идентификатор объекта, этот объект должен поддерживать позднее связывание, правым операндом является имя поля/метода, с которым необходимо связаться, например **excapp~Visible** - если необходимо получить/установить свойство или **excapp~Cells(3,2)**, если необходимо вызвать какой-то метод.

Объект может поддерживать следующие возможности позднего связывания:

- простой вызов метода **excapp~Quit**, с параметрами или без;
- установка значения **excapp~Cells(3, 2) = "Hello World!"**;
- получение значения **vis = uint(excapp~Visible)**;
- цепочка вызовов **excapp~WorkBooks~Add**, это равнозначно следующей записи
tm pobj = excapp~WorkBooks
tm pobj~Add

Недостатком использования отложенных вызовов является, то что компилятор не может проверить правильность написания полей/методов и соответствие типов, что создает сложности при отладке.

Пример использования отложенных вызовов, данный пример использует библиотеку работы с **COM объектами**.

```
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!"
```

Таблица приоритетов операторов

Как правило все операторы выполняются слева направо, но имеется такое понятие как приоритет операторов. Если следующий оператор имеет более высокий приоритет, то в начале выполнится оператор с более высоким приоритетом. Например, умножение имеет более высокий приоритет и $4 + 5 * 2$ равно **14**, но если мы поставим круглые скобки то $(4 + 5) * 2$ равно **18**.

Символьное обозначение операции	Порядок выполнения
Высший приоритет	
() [] . ~ ->	Слева направо
! &(ун) *(ун) -(ун) ~(ун) ++ -- @(ун)	Справа налево
% * /	Слева направо
+ - @	Слева направо
<< >>	Слева направо
< > <= >= %< %> %<= %>=	Слева направо
!= == %== %!=	Слева направо
&	Слева направо
^	Слева направо
	Слева направо
&&	Слева направо
	Слева направо
?(,,)	Слева направо
= += -= *= /= %= &= = ^= >>= <<= as	Справа налево
Низший приоритет	

Круглые скобки () изменяют порядок вычисления частей выражения. Квадратные скобки применяются для взятия элементов массива или работы с индексными элементами, например символ строки. Унарные операторы это !, &, *, -, ~, ++, --. Все унарные операторы, за исключением инкремента имеют только префиксную запись. Операции инкремента ++ и -- могут быть как префиксными, так и постфиксными. Операторы &, *, -, @, ~ могут быть как бинарными, так и унарными. Остальные операторы являются бинарными.

<элемент строки> ::= {<расш стр символ>} | '\{<выражение>}'

<конст строка> ::= "" { <элемент конст строки> | '\{<элемент конст строки>| ""'\}' } ""

<строка> ::= "" { <элемент строки> | '\{<элемент строки>| ""'\}' } ""

<элемент конст двоичных данных> ::= '\h'<пробел> [('\2' | '\4' | '\8') <пробел>] | '\i'<пробел> [('\2' | '\4' | '\8') <пробел>] | <шестнадцатеричная цифра> {<шестнадцатеричная цифра>} { (<пробел> | '\;' | <конец строки>) } <шестнадцатеричная цифра> {<шестнадцатеричная цифра>} | <целое число> { (<пробел> | '\;' | <конец строки>) } <целое число> } | '\{<конст строка> | '\{<любой символ>'\}' | '\\$'<имя макроса>'\\$' | '\\$' {<элемент макростроки>} "" | '\{<элемент макростроки>'\}'>

<элемент двоичных данных> ::= <элемент конст двоичных данных> | '\{<выражение>}'

<конст двоичные данные> ::= "" {<элемент конст двоичных данных>} ""

<двоичные данные> ::= "" {<элемент двоичных данных>} ""

<конст коллекция> ::= '%{' <константа> {','<константа>} '}'

<коллекция> ::= '%{' <выражение> {','<выражение>} '}'

<константа> ::= <число> | <конст строка> | <конст двоичные данные> | <конст коллекция>

<описание массива> ::= ['[' {','}'] [of <имя типа>]

<объект> ::= <имя переменной> | <указатель> | <элемент массива> | <поле> | <вызов функции> | <вызов метода> | <выражение> | <позднее связывание>

<указатель> ::= <выражение> '->' <имя типа> [<описание массива>]

<параметры> ::= <выражение> {','<выражение>}

<элемент массива> ::= <объект> '['<параметры>']'

<поле> ::= [<объект>] '.'<имя поля>

<позднее связывание> ::= <объект> '->' (<имя поля> | <имя метода> '(' [<параметры>] ')')

<вызов функции> ::= (<имя функции> | <выражение> '->' func) '(' [<параметры>] ')'

<вызов метода> ::= [<объект>] '.'<имя метода> '(' [<параметры>] ')'

<lvalue> ::= <объект> | <имя переменной> | <указатель> | <элемент массива> | <поле>

<операция as> ::= <имя переменной> 'as' ((<имя типа> [<описание массива>]) | <объект>)

<операнд> ::= <lvalue> | <константа> | <строка> | <двоичные данные> | <коллекция> | <вызов функции> | <вызов метода> | <имя типа> | <позднее связывание>

<оператор инкремента> ::= '++' | '--'

<оператор присваивания> ::= '=' | '%=' | '&=' | '*=' | '+=' | '-=' | '/=' | '<<=' | '>>=' | '^=' | '|='

<оператор унарный> ::= '+' | '-' | '*' | '/' | '~' | '@'

<оператор бинарный> ::= '==' | '!=' | '>' | '<' | '<=' | '>=' | '&&' | '||' | '&' | '%' | '*' | '/' | '+' | '-' | '<<' | '>>' | '^' | '%==' | '%!==' | '%>' | '%<' | '%<=' | '%>=' | '@'

<оператор> ::= <оператор инкремента> | <оператор присваивания> | <оператор унарный> | <оператор бинарный>

<выр присв> ::= <lvalue><оператор присваивания><выражение>

<выр с lvalue> ::= '&'<lvalue> | '&'<имя функции> | <оператор инкремента><lvalue> | <lvalue><оператор инкремента>

<выражение-вопрос> ::= '?'' (<выражение>',' [<выражение>] ',' [<выражение>])'

<выражение> ::= <операнд> | <выр присв> | <выр с lvalue> | <выражение><оператор бинарный> [<конец строки>] | <выражение> | '\{<выражение>'\}' | <оператор унарный><выражение> | <операция as> | <выражение-вопрос>

<описание переменной> ::= <имя переменной> '['<выражение> { ','<выражение> } ']' [of <имя типа>]

<перечисление переменных> ::= <описание переменной> '['<выражение>','<перечисление переменных>'] | <описание переменной> '['','<перечисление переменных>']

<определение переменных> ::= <имя типа><перечисление переменных><конец строки>

<if> ::= **if** <выражение><блок> **{elif** <выражение> <блок> } **[else** <блок>]

<while> ::= **while** <выражение><блок>

<dowhile> ::= **do** <блок> **while** <выражение>

<for> ::= **for** [<выражение>] ';' <выражение> ';' [<выражение>] <блок>

<fornum> ::= **fornum** <имя переменной> [=] <выражение> ';' <выражение><блок>

<foreach> ::= **foreach** [<имя типа>] <имя переменной> [<описание массива>] ';' <выражение><блок>

<return> ::= **return** [<выражение>]

<label> ::= **label** <имя>

<goto> ::= **goto** <имя>

<switch> ::= **switch** <выражение> '{ **case** <выражение> { ';' <выражение> } {<label>} <блок> } **[default** {<label>} <блок>] }

<содержимое блока> ::= <команда блока> {<конец строки><команда блока>}

<блок> ::= '{<содержимое блока>}'

<команда блока> ::= {<label>} (<блок> | <выражение> | <определение переменных> | <if> | <for> | <fornum> | <while> | <dowhile> | <foreach> | <switch> | **break** | **continue** | <return>)

<описание параметра> ::= <имя переменной> [<описание массива>]

<описание параметров> ::= <имя типа><описание параметра> { [';'] <описание параметра> } [<описание параметров>]

<атрибуты> ::= '<'<имя атрибута> { [';'] <имя атрибута> } '>'

<подфункция> ::= **subfunc** [<имя типа>] <имя функции> ['(' [<описание параметров>] ')'] <блок>

<тело функции> ::= '{ (<команда блока> | <подфункция>) <команда блока> {<конец строки> (<команда блока> | <подфункция>) } }

<описание функции> ::= **func** [<имя типа>] <имя функции> [<атрибуты>] ['(' [<описание параметров>] ')']

<func> ::= <описание функции><тело функции>

<описание метода> ::= **method** [<имя типа>] <имя типа> '.' <имя метода> [<атрибуты>] ['(' [<описание параметров>] ')']

<method> ::= <описание метода><тело функции>

<описание свойства> ::= **property** [<имя типа>] <имя типа> '.' <имя метода> [<атрибуты>] ['(' [<описание параметров>] ')']

<property> ::= <описание свойства><тело функции>

<описание оператора> ::= **operator** <имя типа> <оператор> [<атрибуты>] ['(' [<описание параметров>] ')']

<operator> ::= <описание оператора><тело функции>

<описание текст-функции> ::= **text** <имя функции> [<атрибуты>] ['(' [<описание параметров>] ')']

<тело текст-функции> ::= { <элемент конст строки> | '@'<имя функции> '(' [<параметры>] | '\<выражение>' | '\{ (<команда блока> | <подфункция>) <команда блока> {<конец строки> (<команда блока> | <подфункция>) } } '\}

<text> ::= <описание текст-функции><конец строки><тело текст-функции>

<определение макроса> ::= <имя макроса> [=] (<константа>|<имя>)

<define> ::= **define** [<имя>][<атрибуты>] '{<определение макроса> {<конец строки><определение макроса> } }

<выражение с макросами> ::= '\$'<имя макроса> | <константа> | '!'<выражение с макросами> | '('<выражение с макросами>)' | <выражение с макросами> ('&&' | '||' | '==' | '!=') <выражение с макросами>

<ifdef> ::= **ifdef** <выражение с макросами> '{ ... }' **{ elif** <выражение с макросами> '{ ... }' } **[else** '{ ... }']

<имя файла> ::= `"" {<стр символ>} ""`

<include> ::= `include '{<имя файла> {<конец строки><имя файла>} '}'`

<описание импортируемой функции> ::= [<имя типа>] <имя функции> `'{ [ <имя типа> { ','<имя типа> } ] }' [ '->' <имя функции> ]`

<import> ::= `import <имя файла> [<атрибуты>] '{<описание импортируемой функции> { <конец строки><описание импортируемой функции> } '}'`

<описание поля> ::= <описание поля> [<определение массива>]

<описание полей> ::= <имя типа><описание поля> `{[ ',' <описание поля> }<конец строки>`

<type> ::= `type [<атрибуты>] '{<описание полей>{<описание полей>} '}'`

<определение массива> ::= `{[ '<натуральное число> { [ ',' <натуральное число> } ]' ] [of <имя типа> ]`

<описание глоб переменной> ::= <имя переменной> [<определение массива>]['=' <константа>]

<описание глоб переменных> ::= <имя типа><описание глоб переменной> `{[ ',' <описание глоб переменной> }<конец строки>`

<global> ::= `global '{<описание глоб переменных>} '}'`

<public> ::= `public`

<private> ::= `private`

<extern> ::= `extern '{ ( <описание функции> | <описание метода> | <описание оператора> | <описание свойства> )<конец строки> } '}'`

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

<программа> ::= <команда> {<конец строки><команда>}

Настройка и запуск Gentee

В данном разделе рассматриваются следующие вопросы

- Способы запуска Gentee программ.
- Настройка и опции компилятора.
- Создание EXE файлов.
- Интеграция с другими языками программирования. В частности, использование gentee.dll.

Содержание

- [Быстрый запуск](#)
- [Запуск из командной строки](#)
- [Использование #!](#)
- [Профили компиляции](#)

Быстрый запуск

Программы на языке Gentee можно редактировать с помощью любого текстового редактора. Программы введенные в текстовом редакторе необходимо сохранять с расширением **.g**, что позволит легко их запускать в Проводнике или любом файловом менеджере. Для этого вам достаточно сделать двойное нажатие мышкой или нажать клавишу *Enter* для запуска текущей программы. Аналогично запускаются файлы с расширением **.ge** (откомпилированные Gentee программы). Использование **.ge** позволяет ускорить процесс запуска так как программа не требует дополнительной компиляции.

В проекте Gentee имеются готовые примеры программ. В файловом менеджере или Проводнике откройте директорию куда был установлен Gentee (как правило, это *C:\Program Files\Gentee*) и выберите поддиректорию **Samples**, в которой вы увидите список программ-примеров.

Для часто используемых Gentee программ можно создавать ярлыки в **Пуск->Программы** или на **Рабочем столе**. В качестве запускаемого файла достаточно указать в ваш файл с расширением **.g** или **.ge**. С помощью ярлыков запуск становится еще проще.

Смотрите также

- [Использование #!](#)
- [Профили компиляции](#)

Запуск из командной строки

Компиляция и выполнение программы на языке Gentee осуществляется с помощью консольного приложения **gentee.exe**. Опции командной строки не охватывают все возможности, используйте [Профили компиляции](#) для указания дополнительных параметров компиляции.

gentee.exe [опции] <исходный файл> [аргументы]

опции

Опции компилятора. При компиляции возможно использование следующих опций.

-a	Компилятор транслирует байт-код в ассемблер. В данный момент транслируются не весь байт-код, тем не менее, использование данной опции позволяет увеличить скорость выполнения некоторых программ в несколько раз.
-c	Только компиляция. Не запускать программу после компиляции.
-d	Компилировать с добавлением отладочной информации.
-m <макросы>	Определение макросов компиляции. После -m вы можете определить необходимые макросы компиляции. Перед кавычками необходимо указывать \ . Определения макросов должны разделяться точкой с запятой. Например: -m "MODE=1;NAME="My Company, Inc!"\
-f	Создавать .ge файл с байт-кодом. Он будет создан в этой же директории и с таким же именем.
-n	Игнорировать первую строку с #! в теле запускаемой программы. Смотрите Использование #! .
-o <имя GE или EXE файла>	Эта опция позволяет создавать .ge или .exe файл с любым именем в любом месте. После -o должно идти имя выходящего файла. Эта возможность используется в том случае, если Вы хотите, чтобы результирующий файл имел отличное имя или местоположение от исходного файла. По умолчанию, откомпилированный байт-код сохраняется в файле с расширением .ge .
-p <имя профиля>	Использовать параметры профиля из файла gentee.ini . Смотрите Профили компиляции .
-s	Не выводить служебные сообщения в процессе компиляции или запуска.
-t	Автоматически конвертировать текст в OEM-кодировку (DOS-кодировку) при выводе на консоль.
-d	Добавлять отладочную информацию в байт-код.
-w	Ожидать нажатие клавиши в конце компиляции.
-z[d][n][u]	Оптимизировать байт-код (совместимо с -f или -x) -zd - Удалять define определения. -zn - Удалять имена. -zu - Удалять неиспользуемые или не вызываемые объекты. -z равно -zdu . Комбинация -zd , -zn и -zu .
-x[d][g][a][r]	Создавать исполняемый EXE файл. -xd - Динамическое подключение gentee.dll. -xg - Создавать GUI приложение. По умолчанию создается консольное приложение. -xa - Укажите эту опцию если ваша программа или ее часть была откомпилирована с опцией -a . -xr - Укажите эту опцию, если вы хотите, чтобы ваш байт-код транслировался в ассемблер только в момент загрузки. Не используйте в этом случае опцию -a . -xdgr - Комбинация -xd , -xg и -xg .
-i <icon file>	Вставлять иконку (совместимо с -x). Пример -i "c:\data\myicon.ico"
-r <res file>	Вставлять .res файл ресурсов (совместимо с -x). Пример -r "c:\data\myres.res"

исходный файл

Этот параметр является обязательным параметром и должен определять имя файла компиляции или файл с байт-кодом для выполнения.

аргументы

Все параметры после имени запускаемого файла являются параметрами командной строки, которые будут переданы запускаемой программе.

Примеры

```
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"
```

Смотрите также

- [Использование #!](#)

- [Профили компиляции](#)

Использование #!

Под Linux указание в первой строке **#!** служит для запуска компилятора. Под Windows вы также можете использовать первую строку в файле для запуска любых программ, в том числе для компиляции с нужными опциями. Если вы щелкните мышкой или нажмете Enter на таком .g файле, то будет запущена указанная командная строка. Это позволяет избежать использования дополнительных командных (.bat) файлов и указывать опции компиляции отличные от опций по умолчанию.

Вы можете указывать как абсолютные так и относительные пути для запускаемой программы и файла. В качестве полного имени текущего файла можно указывать %1. Если путь содержит пробелы, то необходимо заключить его в двойные кавычки.

Примеры

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

Использование профиля

Вы можете описать параметры профиля прямо в начале исходного .g файла. Параметры должны начинаться с первой строки и первым символом должен быть . Имена параметров описаны на странице [Профили компиляции](#).

Пример

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

Профили компиляции

Кроме прямого указания опций компиляции при запуске Gentee программ, вы можете хранить все необходимые вам параметры в отдельном профиле и при запуске компилятора достаточно указать имя этого профиля. Профили должны быть описаны в текстовом файле **gentee.ini** расположенном в той же директории, что и **gentee.exe**. При запуске имя профиля указывается после опции **-p**. Например: **gentee.exe -p myoptions test.g**. По умолчанию, при запуске Gentee программ используется профиль компиляции с именем **default**.

Вы также можете указывать профили компиляции прямо в начале .g файла. Смотрите [Использование #!](#) для более подробной информации.

asm = <0 1>	Если 1, то компилятор транслирует байт-код в ассемблер. В данный момент транслируются не весь байт-код, тем не менее, использование данной опции позволяет увеличить скорость выполнения некоторых программ в несколько раз.
silent = <0 1>	Если 1, то не выводить служебные сообщения в процессе компиляции или запуска.
charoem = <0 1>	Если 1, то конвертировать строки в OEM (DOS) кодировку при выводе на консоль.
debug = <0 1>	Если 1, то при компиляции в байт-код будет добавлена отладочная информация.
gfile = <0 1>	Если 1, то создавать .ge файл при компиляции.
norun = <0 1>	Если 1, то не запускать программу после компиляции.
numsign = <0 1>	Если 0, то игнорировать первую строку с #! в теле запускаемой программы.
output = <имя .ge или .exe файла>	В этом параметре можно указать полный путь и имя создаваемого .ge или .exe файла.
define = <макрос = значение>	Параметр служит для определения макросов компиляции. Можно определить несколько define параметров: define 1,define 2,define 3....
include = <.g или .ge файл>	Можно указать дополнительные .g или .ge файлы которые будут добавляться в начале компиляции. Это эквивалентно использованию команды include в Gentee программе. Можно определить несколько подключаемых файлов с помощью include 1,include 2,include 3....
libdir = <директория>	Параметр позволяет указать путь поиска для подключаемых в программе .g или .ge файлов. Если путь определен, то в программе достаточно указывать только имя файла. Можно определить несколько директорий для поиска с помощью libdir 1,libdir 2,libdir 3....
wait = <0 1..n>	Если 1, то ожидать нажатие клавиши в конце компиляции. Если вы укажете число больше 1, то компилятор будет ждать указанное количество секунд и потом сам закроет окно.
optimizer = <0 1 (d n u)>	Если 1, то оптимизировать байт-код. После единицы через пробел вы можете указать дополнительные параметры d , n или u . d - Удалять define определения. n - Удалять имена. u - Удалять неиспользуемые или не вызываемые объекты. Например: optimizer = 1 d n u
exe = <0 1 (d g a r)>	Если 1, то создавать исполняемый EXE файл. После единицы через пробел вы можете указать дополнительные параметры d g a r . d - Динамическое подключение gentee.dll. g - Создавать GUI приложение. По умолчанию создается консольное приложение. a - Укажите эту опцию если ваша программа или ее часть была откомпилирована с опцией asm . r - Укажите эту опцию, если вы хотите, чтобы ваш байт-код транслировался в ассемблер только в момент загрузки. Не используйте в этом случае опцию asm . Например: exe = 1 d g r
icon = <.ico файл>	Можно указать дополнительные .ico файлы для создаваемого EXE файла. Можно определить несколько файлов иконок с помощью icon 1,icon 2,icon 3....
res = <.res файл>	Можно указать дополнительные .res файлы ресурсов для создаваемого EXE файла. Можно определить несколько файлов ресурсов с помощью res 1,res 2,res 3....
args = <параметр>	Параметры командной строки передаваемые при запуске. Можно определить несколько параметров с помощью args 1,args 2,args 3....

Дополнительные возможности

Вы можете использовать следующие predefined значения.

%GNAME%	Имя запускаемого Gentee файла без расширения.
%GPATH%	Путь к запускаемому Gentee файлу.

`%EXEPATH%`

Путь к компилятору `gentee.exe`.

Пример

`[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."`

Библиотечные функции

Содержание

Array	Массив.
Array Of Strings	Массив строк.
Array Of Unicode Strings	Массив юникодных строк.
Buffer	Двоичные данные.
Clipboard	Буфер обмена.
Collection	Коллекция.
COM/OLE	Работа с COM/OLE объектами.
Console	Консольная библиотека.
CSV	Работа с CSV данными.
Date & Time	Функции для работы с датами и временем.
Dbf	Данная библиотека предназначена для работы с dbf файлами.
Files	Функции для работы с файловой системой.
FTP	FTP протокол.
Gentee API	Gentee API функции для использования gentee.dll.
Hash	Хэш (Ассоциативный массив).
HTTP	HTTP протокол.
INI File	INI файлы.
Keyboard	Данные функции предназначены для эмуляции работы клавиатуры.
Math	Математические функции.
Memory	Функции для работы с памятью.
ODBC (SQL)	Работа с базами данных через ODBC (SQL запросы).
Process	Функции для процесса, запуска по расширению, получению аргументов и переменных окружения.
Registry	Работа с Реестром.
Socket	Сокеты и общие интернет функции.
Stack	Стек.
String	Строки.
String - Filename	Работа с именами файлов.
String - Unicode	Юникодные строки.
System	Системные функции.
Thread	Данная библиотека позволяет создавать потоки и работать с ними.
Tree	Объект дерево.
XML	Обработка XML файлов.

Array

Массив. Вы можете использовать переменные типа **arr** для работы с массивами. Тип **arr** наследуется от типа **buf**.

- [Операторы](#)
- [Методы](#)

Операторы

<code>* arr</code>	Получить количество элементов.
<code>foreach var,arr</code>	Оператор foreach.
<code>arr of type</code>	Указание типа элементов.
<code>arr[i]</code>	Получение [i] элемента массива.

Методы

<code>arr.clear</code>	Очистить массив.
<code>arr.cut</code>	Обрезать массив.
<code>arr.del</code>	Удалить элементы.
<code>arr.expand</code>	Добавить элементы в массив.
<code>arr.insert</code>	Вставить элементы.
<code>arr.move</code>	Переместить элемент в массиве.
<code>arr.sort</code>	Сортировать массив.

*** arr**

Получить количество элементов.

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

Возвращаемое значение

Количество элементов массива.

Смотрите также

- [Array](#)

foreach var,arr

Оператор foreach. Можно использовать оператор **foreach** для перебора в сех элементов массива.

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

Смотрите также

- [Array](#)

arr of type

Указание типа элементов. Вы можете определять тип элементов массива с помощью оператора **of** когда вы описываете переменную типа **arr**. По умолчанию, элементы массива имеют тип **uint**.

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

Смотрите также

- [Array](#)

arr[i]

- [method uint arr.index\(uint i \)](#)
- [method uint arr.index\(uint i, uint j \)](#)
- [method uint arr.index\(uint i, uint j, uint k \)](#)

Получение [i] элемента массива.

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

Возвращаемое значение

[i] элемент массива.

arr[i,j]

Получение [i,j] элемента массива.

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

Возвращаемое значение

[i,j] элемент массива.

arr[i,j,k]

Получение [i,j,k] элемента массива.

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

Возвращаемое значение

[i,j,k] элемент массива.

Смотрите также

- [Array](#)

arr.clear

Очистить массив. Метод удаляет все элементы массива.

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

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Array](#)

arr.cut

Обрезать массив. Будут удалены все элементы сверх указанного количества.

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

Параметры

count Количество оставляемых элементов в массиве.

Смотрите также

- [Array](#)

arr.del

- [method arr.del\(uint num \)](#)
- [method arr arr.del\(uint from, uint count \)](#)

Удалить элементы. Удалить элемент с указанным номером в массиве.

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

Параметры

num Номер удаляемого элемента с 0.

arr.del

Удалить элементы в массиве.

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

Параметры

from Номер первого удаляемого элемента с 0.

count Количество удаляемых элементов.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Array](#)

arr.expand

Добавить элементы в массив.

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

Параметры

count Количество добавляемых элементов.

Возвращаемое значение

Номер первого добавленного элемента.

Смотрите также

- [Array](#)

arr.insert

- [method arr.insert\(uint id \)](#)
- [method uint arr.insert\(uint from, uint count \)](#)

Вставить элементы. Вставить элемент в массив по указанному индексу.

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

Параметры

id Индекс вставляемого элемента с нуля.

arr.insert

Вставить элементы в массив по указанному индексу.

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

Параметры

from Индекс первого вставляемого элемента с 0.

count Количество вставляемых элементов.

Возвращаемое значение

Номер первого вставленного элемента.

Смотрите также

- [Array](#)

arr.move

Переместить элемент в массиве.

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

Параметры

from Текущий индекс элемента с нуля.
to Новый индекс элемента с нуля.

Смотрите также

- [Array](#)

Array Of Strings

Массив строк. Вы можете использовать переменные типа `arrstr` для работы с массивом строк. Тип `arrstr` наследуется от типа `arr`, поэтому вы можете также использовать [методы типа arr](#).

- [Операторы](#)
- [Методы](#)
- [Дополнительные методы](#)
- [Тип](#)

Операторы

<code>arrstr = type</code>	Конвертировать типы в массив строк.
<code>str = arrstr</code>	Конвертировать массив строк в многостроковый текст.
<code>arrstr += type</code>	Добавлять типы к массиву строк.

Методы

<code>arrstr.insert</code>	Вставить строку в массив строк.
<code>arrstr.load</code>	Добавить строки к массиву из многострокового текста.
<code>arrstr.read</code>	Прочитать текстовый файл в массив строк.
<code>arrstr.replace</code>	Замена подстрок для каждого элемента.
<code>arrstr.setmultistr</code>	Создать многостроковый буфер.
<code>arrstr.sort</code>	Сортировать строки в массиве.
<code>arrstr.unite...</code>	Объединить строки массива.
<code>arrstr.write</code>	Записать массив строк в многостроковый текстовый файл.

Дополнительные методы

<code>buf.getmultistr</code>	Конвертировать буфер в массив строк.
<code>str.lines</code>	Конвертировать многостроковый текст в массив строк.
<code>str.split</code>	Разделение строки.

Тип

<code>arrstr</code>	Структура массива строк.
---------------------	--------------------------

arrstr = type

- [operator arrstr =\(arrstr dest, str src \)](#)
- [operator arrstr =\(arrstr dest, arrstr src \)](#)
- [operator arrstr =\(arrstr left, collection right \)](#)

Конвертировать типы в массив строк. Конвертировать многостроковый текст в массив строк.

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

Возвращаемое значение

Массив строк.

arrstr = arrstr

Копировать один массив строк в другой массив строк.

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

arrstr = collection

Копировать коллекцию строк в массив строк.

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

Смотрите также

- [Array Of Strings](#)

str = arrstr

Конвертировать массив строк в многостроковый текст.

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

Возвращаемое значение

Результирующая строка.

Смотрите также

- [Array Of Strings](#)

arrstr += type

- [operator arrstr +=\(arrstr dest, str new str \)](#)
- [operator arrstr +=\(arrstr dest, arrstr src \)](#)

Добавлять типы к массиву строк. Оператор добавляет строку в конец массива строк.

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

Возвращаемое значение

Возвращается объект для которого был вызван метод.

arrstr += arrstr

Оператор добавляет один массив строк к другому массиву строк.

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

Смотрите также

- [Array Of Strings](#)

arrstr.insert

Вставить строку в массив строк.

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

Параметры

index Номер под которым будет вставлена строка.
newstr Вставляемая строка.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Array Of Strings](#)

arrstr.load

- [method arrstr arrstr.load\(str input, uint flag \)](#)
- [method arrstr arrstr.loadtrim\(str input \)](#)

Добавить строки к массиву из многострокового текста.

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

Параметры

inp Исходный текст.

ut

fla Флаги.

g

\$ASTR_APPEND Добавлять строки. В противном случае, массив будет очищен перед добавлением.

\$ASTR_TRIM Удалять символы меньше или равные пробелу справа и слева.

Возвращаемое значение

Возвращается объект для которого был вызван метод..

arrstr.loadtrim

Добавить строки к массиву из многострокового текста с удалением крайних пробелов.

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

Параметры

input Исходный текст.

Смотрите также

- [Array Of Strings](#)

arrstr.replace

Замена подстрок для каждого элемента. Этот метод для каждой строки массива ищет строки из одного массива строк и заменяет их строками из другого.

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

Параметры

aold Массив строк которые ищутся.

anew Новые строки для замены.

flag Флаги.

s

\$QS_IGNCASE	Игнорировать регистр при поиске.
\$QS_WORD	Искать только целые слова.
\$QS_BEGINWORD	Искать слова которые начинаются с указанного шаблона.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Array Of Strings](#)

arrstr.setmultistr

- [method buf arrstr.setmultistr\(buf dest \)](#)
- [method buf arrstr.setmultistr <result>](#)

Создать многостроковый буфер. Метод записывает строки в буфер, в котором они разделены нулевым символом.

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

Параметры

dest Результирующий буфер.

Возвращаемое значение

Результирующий буфер.

Метод записывает строки в новый буфер, в котором они разделены нулевым символом.

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

Возвращаемое значение

Новый результирующий буфер.

Смотрите также

- [Array Of Strings](#)

arrstr.sort

Сортировать строки в массиве.

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

Параметры

mode Укажите 1 для сортировки без учета регистра. По умолчанию указывайте 0.

Смотрите также

- [Array Of Strings](#)

arrstr.unite...

- [method str arrstr.unite\(str dest, str separ \)](#)
- [method str arrstr.unite\(str dest \)](#)
- [method str arrstr.unitelines\(str dest \)](#)

Объединить строки массива. Метод объединяет все элементы массива в одну строку с указанным разделителем.

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

Параметры

dest Результирующая строка.
separ Разделитель строк.

Возвращаемое значение

Результирующая строка.

arrstr.unite

Метод объединяет все элементы массива в одну строку.

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

Параметры

dest Результирующая строка.

arrstr.unitelines

Метод объединяет все элементы массива в многостроковый текст. После каждой добавляемой строки вставляется перевод строки.

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

Параметры

dest Результирующая строка.

Смотрите также

- [Array Of Strings](#)

arrstr

Структура массива строк.

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

Смотрите также

- [Array Of Strings](#)

Array Of Unicode Strings

Массив юникодных строк. Вы можете использовать переменные типа **arrustr** для работы с массивом строк. Тип **arrustr** наследуется от типа **arr**, поэтому вы можете также использовать [методы типа arr](#).

- [Операторы](#)
- [Методы](#)
- [Дополнительные методы](#)
- [Тип](#)

Операторы

arrustr = type Конвертировать типы в массив юникодных строк.

ustr = arrustr Конвертировать массив юникодных строк в многостроковый юникодный текст.

arrustr += type Добавлять типы к массиву юникодных строк.

Методы

arrustr.insert Вставить юникодную строку в массив юникодных строк.

arrustr.load Добавить строки к массиву из многострокового юникодного текста.

arrustr.read Прочитать текстовый юникодный файл в массив юникодных строк.

arrustr.setmultiustr Создать многостроковый буфер.

arrustr.sort Сортировать юникодные строки в массиве.

arrustr.unite... Объединить юникодные строки массива.

arrustr.write Записать массив юникодных строк в многостроковый текстовый файл.

Дополнительные методы

buf.getmultiustr Конвертировать буфер в массив юникодныхх строк.

ustr.lines Конвертировать многостроковый юникодный текст в массив юникодных строк.

ustr.split Разделение юникодной строки.

Тип

arrustr Структура массива юникодных строк.

arrustr = type

- [operator arrustr =\(arrustr dest. ustr src \)](#)
- [operator arrustr =\(arrustr dest. arrustr src \)](#)
- [operator arrustr =\(arrustr left. collection right \)](#)

Конвертировать типы в массив юникодных строк. Конвертировать многостроковый юникодный текст в массив юникодных строк.

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

Возвращаемое значение

Массив юникодных строк.

arrustr = arrustr

Копировать один массив юникодных строк в другой массив строк.

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

arrustr = collection

Копировать коллекцию простых и юникодных строк в массив юникодных строк.

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

Смотрите также

- [Array Of Unicode Strings](#)

ustr = arrustr

Конвертировать массив юникодных строк в многостроковый юникодный текст.

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

Возвращаемое значение

Результирующая юникодная строка.

Смотрите также

- [Array Of Unicode Strings](#)

arrustr += type

- [operator arrustr +=\(arrustr dest, ustr new str \)](#)
- [operator arrustr +=\(arrustr dest, arrustr src \)](#)

Добавлять типы к массиву юникодных строк. Оператор добавляет юникодную строку в конец массива строк.

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

Возвращаемое значение

Возвращается объект для которого был вызван метод.

arrustr += arrustr

Оператор добавляет один массив юникодных строк к другому массиву юникодных строк.

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

Смотрите также

- [Array Of Unicode Strings](#)

arrustr.insert

Вставить юникодную строку в массив юникодных строк.

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

Параметры

index Номер под которым будет вставлена строка.
newstr Вставляемая строка.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Array Of Unicode Strings](#)

arrustr.load

- [method arrustr arrustr.load\(ustr input, uint flag \)](#)
- [method arrustr arrustr.loadtrim\(ustr input \)](#)

Добавить строки к массиву из многострокового юникодного текста.

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

Параметры

inp Исходный юникодный текст.

ut

fla Флаги.

g

\$ASTR_APPEND Добавлять строки. В противном случае, массив будет очищен перед добавлением.

\$ASTR_TRIM Удалять символы меньше или равные пробелу справа и слева.

Возвращаемое значение

Возвращается объект для которого был вызван метод..

arrustr.loadtrim

Добавить строки к массиву из многострокового юникодного текста с удалением крайних пробелов.

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

Параметры

input Исходный юникодный текст.

Смотрите также

- [Array Of Unicode Strings](#)

arrustr.setmultiustr

Создать многостроковый буфер. Метод записывает юникодные строки в буфер, в котором они разделены нулевым ushort.

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

Параметры

dest Результирующий буфер.

Возвращаемое значение

Результирующий буфер.

Смотрите также

- [Array Of Unicode Strings](#)

arrustr.sort

Сортировать юникодные строки в массиве.

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

Параметры

mode Укажите 1 для сортировки без учета регистра. По умолчанию указывайте 0.

Смотрите также

- [Array Of Unicode Strings](#)

arrustr.unite...

- [method ustr arrustr.unite\(ustr dest, ustr separ \)](#)
- [method ustr arrustr.unitelines\(ustr dest \)](#)

Объединить юникодные строки массива. Метод объединяет все элементы массива в одну юникодную строку с указанным разделителем.

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

Параметры

dest Результирующая юникодная строка.
separ Разделитель строк.

Возвращаемое значение

Результирующая юникодная строка.

arrustr.unitelines

Метод объединяет все элементы массива в многостроковый юникодный текст. После каждой добавляемой строки вставляется перевод строки.

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

Параметры

dest Результирующая юникодная строка.

Смотрите также

- [Array Of Unicode Strings](#)

arrustr

Структура массива юникодных строк.

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

Смотрите также

- [Array Of Unicode Strings](#)

Buffer

Двоичные данные. Вы можете использовать переменную типа **buf** для работы с памятью. Используйте этот тип если вы хотите работать с двоичными данными.

- [Операторы](#)
- [Методы](#)

Операторы

<code>* buf</code>	Получить размер буфера.
<code>buf[i]</code>	Получить <i> байт в буфере.
<code>buf = buf</code>	Копирование данных из одного буфера в другой.
<code>buf + buf</code>	Сложить два буфера в один результирующий.
<code>buf += type</code>	Добавить типы к буферу.
<code>buf == buf</code>	Операция сравнения.
<code>buf(type)</code>	Конвертировать типы в buf.

Методы

<code>buf.align</code>	Выравнивание данных.
<code>buf.append</code>	Добавление данных.
<code>buf.clear</code>	Очистить буфер.
<code>buf.copy</code>	Копирование.
<code>buf.crc</code>	Подсчет контрольной суммы.
<code>buf.del</code>	Удаление данных.
<code>buf.expand</code>	Расширение.
<code>buf.free</code>	Освобождение памяти.
<code>buf.findch</code>	Найти данный байт в буфере.
<code>buf.getm ultistr</code>	Конвертировать буфер в массив строк.
<code>buf.getm ultiustr</code>	Конвертировать буфер в массив юникодных строк.
<code>buf.insert</code>	Вставка данных.
<code>buf.ptr</code>	получить указатель на память.
<code>buf.read</code>	Чтение из файла.
<code>buf.replace</code>	Замена данных.
<code>buf.reserve</code>	Резервирование памяти.
<code>buf.write</code>	Запись в файл.
<code>buf.writeappend</code>	Дописать данные к файлу.

* buf

Получить размер буфера.

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

Возвращаемое значение

Размер буфера.

Смотрите также

- [Buffer](#)

buf[i]

Получить <i> байт в буфере.

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

Возвращаемое значение

Значение i-го байта в буфере.

Смотрите также

- [Buffer](#)

buf = buf

Копирование данных из одного буфера в другой.

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

Возвращаемое значение

Результирующий буфер.

Смотрите также

- [Buffer](#)

buf + buf

Сложить два буфера в один результирующий.

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

Возвращаемое значение

Новый результирующий буфер.

Смотрите также

- [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 +=\(buf left, ulong right \)](#)

Добавить типы к буферу. Добавление **buf** к **buf** => **buf += buf**.

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

Возвращаемое значение

Результирующий буфер.

buf += ubyte

Добавить **ubyte** к **buf** => **buf += ubyte**.

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

buf += uint

Добавить **uint** к **buf** => **buf += uint**.

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

buf += ushort

Добавить **ushort** к **buf** => **buf += ushort**.

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

buf += ulong

Добавить **ulong** к **buf** => **buf += ulong**.

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

Смотрите также

- [Buffer](#)

buf == buf

- [operator uint ==\(buf left, buf right \)](#)
- [operator uint !=\(buf left, buf right \)](#)

Операция сравнения.

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

Возвращаемое значение

Возвращает **1** если буферы равны и **0** в противном случае.

buf != buf

Операция сравнения.

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

Возвращаемое значение

Возвращает **0** если буферы равны и **1** в противном случае.

Смотрите также

- [Buffer](#)

buf(type)

Конвертировать типы в buf. Конвертация `uint` в `buf => buf( uint )`.

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

Возвращаемое значение

Результирующий буфер.

Смотрите также

- [Buffer](#)

buf.align

Выравнивание данных. Метод выравнивает размер двоичных данных и добавляет нули, если это необходимо..

```
method buf buf.align
```

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Buffer](#)

buf.append

Добавление данных. Метод добавляет данные к буферу.

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

Параметры

ptr Указатель на добавляемые данные.
size Размер добавляемых данных.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Buffer](#)

buf.clear

Очистить буфер. Этот метод устанавливает размер буфера в ноль.

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

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Buffer](#)

buf.copy

Копирование. Метод копирует двоичные данные в буфер.

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

Параметры

ptr Указатель на копируемые данные.
size Размер копируемых данных.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Buffer](#)

buf.crc

Подсчет контрольной суммы. Метод подсчитывает контрольную сумму данных в буфере.

```
method uint buf.crc
```

Возвращаемое значение

Возвращается контрольная сумма.

Смотрите также

- [Buffer](#)

buf.del

Удаление данных. Метод удаляет часть буфера.

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

Параметры

offset Смещение удаляемых данных.
size Размер удаляемых данных.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Buffer](#)

buf.expand

Расширение. Метод увеличивает отведенный размер памяти.

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

Параметры

size Запрашиваемый дополнительный размер памяти. Это дополнительный размер который будет резервироваться в буфере.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Buffer](#)

buf.free

Освобождение памяти. Метод освобождает память отведенную в буфере.

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

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Buffer](#)

buf.findch

Найти данный байт в буфере.

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

Параметры

offset Смещение для начала поиска.
ch Искомый байт.

Возвращаемое значение

Смещение байта если он найден. Если не найден, то возвращается размер буфера.

Смотрите также

- [Buffer](#)

buf.getmultistr

- [method arrstr buf.getmultistr\(arrstr ret, arr offset \)](#)
- [method arrstr buf.getmultistr\(arrstr ret \)](#)

Конвертировать буфер в массив строк. Загрузить массив строк из многострокового буфера где строки разделены нулевым символом.

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

Параметры

ret Результирующий массив строк.
offset Массив для получения смещений в буфере. Может быть 0->>arr.

Возвращаемое значение

Результирующий массив строк.

buf.getmultistr

Загрузить массив строк из многострокового буфера где строки разделены нулевым символом.

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

Параметры

ret Результирующий массив строк.

Смотрите также

- [Buffer](#)

buf.getmultiustr

Конвертировать буфер в массив юникодных строк. Загрузить массив юникодных строк из многострокового буфера где строки разделены нулевым символом.

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

Параметры

ret Результирующий массив юникодных строк.

Возвращаемое значение

Результирующий массив юникодных строк.

Смотрите также

- [Buffer](#)

buf.insert

- [method buf buf.insert\(uint offset, buf value \)](#)
- [method buf buf.insert\(uint offset, uint ptr, uint size \)](#)

Вставка данных. Этот метод вставляет один буфер в другой.

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

Параметры

offset Смещение, куда будет произведена вставка. Если смещение больше текущего размера, то данные будут добавлены в конец.

value Объект **buf**, данные которого будут вставлены.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

buf.insert

Метод вставляет данные в буфер.

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

Параметры

offset Смещение куда будут вставляться данные. Если смещение больше текущего размера, то данные будут добавляться в конец буфера.

ptr Указатель на вставляемые данные.

size Размер вставляемых данных.

Смотрите также

- [Buffer](#)

buf.ptr

получить указатель на память.

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

Возвращаемое значение

Указатель на отведенную память в буфере.

Смотрите также

- [Buffer](#)

buf.replace

Замена данных. Этот метод заменяет двоичные данные в объекте.

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

Параметры

<i>offset</i>	Смещение заменяемых данных.
<i>size</i>	Размер заменяемых данных.
<i>value</i>	Объект типа buf с вставляемыми данными.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Buffer](#)

buf.reserve

Резервирование памяти. Метод увеличивает размер отведенной памяти в буфере.

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

Параметры

size Суммарный запрашиваемый размер памяти. Если он меньше текущего размера, то ничего не происходит. При увеличении размера текущие данные сохраняются.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Buffer](#)

Clipboard

Буфер обмена. Данные функции предназначены для работы с буфером обмена (clipboard) Windows. Для использования библиотеки необходимо с помощью команды `include` указать файл `clipboard.g`, который находится в поддиректории `lib\clipboard`.

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

- [Методы](#)

<code>clipboard_gettext</code>	Получить строку из буфера обмена.
<code>clipboard_empty</code>	Очистить буфер обмена.
<code>clipboard_settext</code>	Записать строку в буфер обмена.

Методы

<code>buf.getclip</code>	Копировать данные из буфера обмена в переменную <code>buf</code> .
<code>buf.setclip</code>	Копировать данные из переменной типа <code>buf</code> в буфер обмена.
<code>str.getclip</code>	Копировать строку из буфера обмена в переменную типа <code>str</code> .
<code>str.setclip</code>	Копировать строку в буфер обмена.
<code>ustr.getclip</code>	Копировать уникальную строку из буфера обмена в переменную типа <code>ustr</code> .
<code>ustr.setclip</code>	Копировать уникальную строку в буфер обмена.

clipboard_gettext

Получить строку из буфера обмена.

```
func str clipboard_gettext (  
    str data  
)
```

Параметры

data Строка для получения результата.

Возвращаемое значение

Возвращается параметр *data*.

Смотрите также

- [Clipboard](#)

clipboard_empty

Очистить буфер обмена.

```
func uint clipboard_empty()
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Clipboard](#)

clipboard_settext

Записать строку в буфер обмена.

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

Параметры

data Строка для копирования в буфер обмена.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Clipboard](#)

buf.getclip

Копировать данные из буфера обмена в переменную buf.

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

Параметры

cftype Тип данных буфера обмена.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Clipboard](#)

buf.setclip

Копировать данные из переменной типа buf в буфер обмена.

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

Параметры

cftype Тип вставляемых данных.
locale Идентификатор локали. Может быть 0.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Clipboard](#)

str.getclip

Копировать строку из буфера обмена в переменную типа str.

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

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Clipboard](#)

str.setclip

Копировать строку в буфер обмена.

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

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Clipboard](#)

ustr.getclip

Копировать уникадную строку из буфера обмена в переменную типа ustr.

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

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Clipboard](#)

ustr.setclip

Копировать уникадную строку в буфер обмена.

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

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Clipboard](#)

Collection

Коллекция. Вы можете использовать переменные типа **collection** для работы с коллекциями. Коллекция это объект, который может содержать объекты различных типов.

- [Операторы](#)
- [Методы](#)
- [Типы](#)

Операторы

<code>* collection</code>	Получить количество элементов в коллекции.
<code>collection[i]</code>	Получить значение элемента коллекции.
<code>collection = collection</code>	Копирование коллекции.
<code>collection += collection</code>	Добавление элементов одной коллекции к другой коллекции.
<code>collection + collection</code>	Сложить две коллекции в одну результирующую.
<code>foreach var, collection</code>	Оператор <code>foreach</code> .

Методы

<code>collection.append</code>	Добавление объекта или числа к коллекции.
<code>collection.clear</code>	Удаление всех элементов коллекции.
<code>collection.gettype</code>	Получить тип элемента коллекции.
<code>collection.ptr</code>	Получить указатель на элемент коллекции.

Типы

<code>colitem</code>	Структура используемая в операторе <code>foreach</code> .
----------------------	---

* collection

Получить количество элементов в коллекции.

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

Возвращаемое значение

Количество элементов в коллекции.

Смотрите также

- [Collection](#)

collection[i]

Получить значение элемента коллекции. Не используйте этот оператор для коллекция содержащих типы double, ulong или long.

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

Возвращаемое значение

Значение элемента коллекции.

Смотрите также

- [Collection](#)

collection = collection

Копирование коллекции.

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

Смотрите также

- [Collection](#)

collection += collection

Добавление элементов одной коллекции к другой коллекции.

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

Смотрите также

- [Collection](#)

collection + collection

Сложить две коллекции в одну результирующую.

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

Возвращаемое значение

Новая результирующая коллекция.

Смотрите также

- [Collection](#)

foreach var,collection

Оператор foreach. Вы можете использовать оператор **foreach** для перебора в сеъ элементов коллекции. Переменная **var** имеет тип [colitem](#).

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

Смотрите также

- [Collection](#)

collection.append

Добавление объекта или числа к коллекции.

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

Параметры

value Значение 32-битного числа или указатель на 64-битное число или указатель на объект.

itype Тип добавляемого значения.

Возвращаемое значение

порядковый номер добавленного элемента.

Смотрите также

- [Collection](#)

collection.clear

Удаление всех элементов коллекции.

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

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Collection](#)

collection.gettype

Получить тип элемента коллекции.

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

Параметры

ind Индекс элемента с 0.

Возвращаемое значение

Тип элемента коллекции или 0 в случае ошибки.

Смотрите также

- [Collection](#)

collection.ptr

Получить указатель на элемент коллекции.

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

Параметры

ind Индекс элемента с 0.

Возвращаемое значение

Указатель на элемент коллекции или 0 в случае ошибки.

Смотрите также

- [Collection](#)

colitem

Структура используемая в операторе [foreach](#). Переменная оператора foreach имеет этот тип.

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

Поля типа

<i>oftype</i>	Тип элемента.
<i>val</i>	Значение элемента.
<i>hival</i>	hi-uint значение, если элемент является 64-битным числом.
<i>ptr</i>	Указатель на значение.

Смотрите также

- [Collection](#)

COM/OLE

Работа с COM/OLE объектами. Данная библиотека предназначена для работы с **COM/OLE объектами**, использует интерфейс **IDispatch** и поддерживает технологию позднего связывания. Для использования библиотеки необходимо с помощью команды `include` указать файл `olecom.g`, который находится в поддиректории `lib\olecom`.

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

- [Операторы](#)
- [Методы](#)
- [Методы для работы с VARIANT](#)

Описание COM/OLE	Краткое описание COM/OLE библиотеки.
VARIANT	Тип VARIANT.
Операторы	
<code>type = VARIANT</code>	Операция присваивания.
<code>VARIANT = type</code>	Операция присваивания.
<code>type(VARIANT)</code>	Конвертация.
Методы	
<code>oleobj.createobj</code>	Создать COM объект.
<code>oleobj.getres</code>	Результат последней операции.
<code>oleobj.iserr</code>	Произошла ли ошибка при работе с COM объектом.
<code>oleobj.release</code>	Освободить COM объект.
Методы для работы с VARIANT	
<code>variant.arrcreate</code>	Создать SafeArray массив.
<code>variant.arrfromg</code>	Присвоить значение элементу SafeArray массива.
<code>variant.arrgetptr</code>	Получить указатель на элемент SafeArray массива.
<code>variant.clear</code>	Очистить переменную.
<code>variant.ismissing</code>	Проверяет является ли переменная 'пропущенным' (опциональным) параметром метода.
<code>variant.isnull</code>	Является ли переменная NULL.
<code>variant.setmissing</code>	Установить переменную как "пропущенный" (опциональный) параметр.

Описание COM/OLE

Краткое описание COM/OLE библиотеки. В данную библиотеку также включена поддержка типа [VARIANT](#), используемого при передаче/получении данных в/из COM объекты. Для работы с COM объектами используются переменные с типом **oleobj**, такой переменной может соответствовать один COM объект. Чтобы вызвать какой-либо метод COM объекта применяется оператор [позднего связывания ~](#). Связывание COM объекта с переменной может происходить двумя способами:

1. Создание нового COM объекта с помощью метода [oleobj.createobj](#):

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

2. Связь с существующим COM объектом (дочерним), как результат вызова метода COM объекта:

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

Объект **oleobj** может поддерживать следующие возможности позднего связывания:

- простой вызов метода **excapp~Quit**, с параметрами или без;
- установка значения **excapp~Cells(3, 2) = "Hello World!"**;
- получение значения **vis = uint(excapp~Visible)**;
- цепочка вызовов **excapp~WorkBooks~Add**, это равнозначно следующей записи

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

Вызов метода может возвращать только тип **VARIANT**, для последующей конвертации данных в стандартные типы Gentee используются соответствующие операции присваивания и приведения типов. Параметры вызова методов COM объектов, а также присваиваемые значения автоматически конвертируются в соответствующие типы VARIANT, можно использовать следующие типы Gentee - **uint, int, ulong, long, float, double, str, VARIANT**.

После работы COM объект может быть освобожден методом [oleobj.release](#) иначе освобождение произойдет при уничтожении переменной, также объект освобождается если переменная связывается с другим COM объектом. Пример работы с COM объектом:

```
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!"
}
```

Объект **oleobj** имеет свойства:

- **uint flgs** - флаги, можно установить или получить значение флагов, свойство может содержать флаг **\$FOLEOBJ_INT** - при передаче данных в COM объект беззнаковый uint тип Gentee автоматически конвертируется в знаковый тип VARIANT(VT_I4)
- **uint errfunc** - функция для обработки ошибок. Данному свойству можно присвоить адрес функции, которая будет вызываться каждый раз, когда произойдет ошибка при обращении к COM объекту, функция должна иметь один параметр типа uint, через который будет передаваться код ошибки.

Свойства **flgs** и **errfunc**, автоматически устанавливаются у всех дочерних объектов.

Смотрите также

- [COM/OLE](#)

VARIANT

Тип VARIANT. **VARIANT** является универсальным типом для хранения в се возможных данных и позволяет разным программам корректно обмениваться данными. Он представляет собой структуру, которая содержит два основных поля, первое это тип хранящегося значения, второе это само значение или указатель на область памяти. Тип **VARIANT** имеет следующее описание

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

vt - код типа содержащегося значения (константы типа VT_*: \$VT_UI4, \$VT_I4, \$VT_BSTR ...);

val - поле для хранения значений.

В библиотеке поддерживается лишь часть возможностей типа VARIANT, но разрешается работать непосредственно с полями данной структуры Пример создания VARIANT(VT_BOOL):

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

Примеры операций с VARIANT:

```
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
```

Смотрите также

- [COM/OLE](#)

type = VARIANT

- [operator str = \(str left, VARIANT right \)](#)
- [operator oleobj = \(oleobj left, VARIANT right \)](#)

Операция присваивания. `str = VARIANT( VT_BSTR )`.

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

Возвращаемое значение

Результирующая строка.

oleobj = VARIANT

Операция присваивания. `oleobj = VARIANT( VT_DISPATCH )`.

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

Возвращаемое значение

Результирующий `oleobj`.

Смотрите также

- [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 = \(VARIANT left, VARIANT right\)](#)

Операция присваивания: **VARIANT = uint**.

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

Возвращаемое значение

VARIANT(VT_UI4).

VARIANT = int

Операция присваивания: **VARIANT = int**.

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

Возвращаемое значение

VARIANT(VT_I4).

VARIANT = float

Операция присваивания: **VARIANT = float**.

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

Возвращаемое значение

VARIANT(VT_R4).

VARIANT = double

Операция присваивания: **VARIANT = double**.

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

Возвращаемое значение

VARIANT(VT_R8).

VARIANT = long

Операция присваивания: **VARIANT = long**.

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

Возвращаемое значение

VARIANT(VT_I8).

VARIANT = ulong

Операция присваивания: **VARIANT = ulong**.

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

```
VARIANT left,  
ulong right  
)
```

Возвращаемое значение

VARIANT(VT_UI8).

VARIANT = str

Операция присваивания: **VARIANT = str**.

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

Возвращаемое значение

VARIANT(VT_BSTR).

VARIANT = VARIANT

Операция присваивания: **VARIANT = VARIANT**.

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

Возвращаемое значение

VARIANT.

Смотрите также

- [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](#)

Конвертация: `str(VARIANT)`.

```
method str VARIANT.str <result>
```

Возвращаемое значение

Результирующая строка `str`.

VARIANT.ulong

Конвертация: `ulong(VARIANT)`.

```
method ulong VARIANT.ulong
```

Возвращаемое значение

Результирующее `ulong` значение.

VARIANT.long

Конвертация: `long(VARIANT)`.

```
method long VARIANT.long
```

Возвращаемое значение

Результирующее `long` значение.

VARIANT.uint

Конвертация: `uint(VARIANT)`.

```
method uint VARIANT.uint
```

Возвращаемое значение

Результирующее `uint` значение.

VARIANT.int

Конвертация: `int(VARIANT)`.

```
method int VARIANT.int
```

Возвращаемое значение

Результирующее `int` значение.

VARIANT.float

Конвертация: `float(VARIANT)`.

```
method float VARIANT.float
```

Возвращаемое значение

Результирующее `float` значение.

VARIANT.double

Конвертация: `double(VARIANT)`.

```
method double VARIANT.double
```

Возвращаемое значение

Результирующее `double` значение.

Смотрите также

- [COM/OLE](#)

oleobj.createobj

Создать COM объект. Пример использования :

```
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
)
```

Параметры

name Имя создаваемого объекта, или идентификатор объекта в строковом представлении - "{...}".

mashine Имя машины на которой следует создать объект, если данная строка пустая , объект создается на текущей машине.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [COM/OLE](#)

oleobj.getres

Результат последней операции. Метод применяется для получения кода ошибки или предупреждения, код соответствует С типу HRESULT.

```
method uint oleobj.getres()
```

Возвращаемое значение

Возвращает код HRESULT последней операции с COM объектом.

Смотрите также

- [COM/OLE](#)

oleobj.iserr

Произошла ли ошибка при работе с COM объектом. Метод определяет произошла ли ошибка при последней операции с COM объектом.

```
method uint oleobj.iserr()
```

Возвращаемое значение

Возвращает 1, если произошла ошибка, иначе 0.

Смотрите также

- [COM/OLE](#)

oleobj.release

Освободить COM объект. Метод уничтожает связь между переменной и COM объектом и освобождает COM объект.

```
method oleobj.release()
```

Смотрите также

- [COM/OLE](#)

variant.arrcreate

Создать SafeArray массив. Метод создает в переменной типа VARIANT массив **SafeArray**. Элементом массива является VARIANT. Для заполнения элементов массива можно использовать метод [variant.arrfromg](#). Можно получить сам элемент с помощью метода [variant.arrgetptr](#).

Пример работы с 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
excap~Range( excap~Cells( 1, 1 ), excap~Cells( 3, 2 ) ) = v
```

Использование SafeArray позволяет скомпоновать данные, что может ускорить обмен данными с COM объектом.

```
method uint VARIANT.arrcreate (
    collection bounds
)
```

Параметры

bounds Коллекция содержащая параметры массива, для каждой размерности массива задается пара чисел, первое число количество элементов, второе - номер первого элемента в размерности.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [COM/OLE](#)

variant.arrfromg

Присвоить значение элементу SafeArray массива. Пример

```
v.arrfromg( %{0,0, 0.1234f} )  
v.arrfromg( %{0,1, int(100)} )  
v.arrfromg( %{2,1, "Test" } )  
method uint VARIANT.arrfromg (  
    collection item  
)
```

Параметры

item Коллекция содержащая "координаты" элемента, последний элемент коллекции - присваиваемое значение.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [COM/OLE](#)

variant.arrgetptr

Получить указатель на элемент SafeArray массива.

```
method uint VARIANT.arrgetptr (  
    collection item  
)
```

Параметры

item Коллекция содержащая "координаты" элемента.

Возвращаемое значение

Возвращает адрес элемента массива или 0 если невозможно получить данный элемент.

Смотрите также

- [COM/OLE](#)

variant.clear

Очистить переменную. Метод очищает содержимое переменной, в случае необходимости освобождается занимаемая данными память. Тип VARIANT устанавливается как VT_EMPTY. Данный метод вызывается автоматически перед установкой нового значения .

```
method VARIANT.clear ()
```

Смотрите также

- [COM/OLE](#)

variant.ismissing

Проверяет является ли переменная 'пропущенным' (опциональным) параметром метода.

```
method uint VARIANT.ismissing()
```

Возвращаемое значение

Возвращает 1, если переменная является 'пропущенным' (опциональным) параметром.

Смотрите также

- [COM/OLE](#)

variant.isnull

Является ли переменная NULL. Метод проверяет равна ли переменная NULL - тип VARIANT(VT_NULL).

```
method uint VARIANT.isnull()
```

Возвращаемое значение

Возвращает 1, если переменная VARIANT имеет тип VT_NULL, иначе 0.

Смотрите также

- [COM/OLE](#)

variant.setmissing

Установить переменную как "пропущенный" (опциональный) параметр.

```
method VARIANT.setmissing()
```

Смотрите также

- [COM/OLE](#)

Console

Консольная библиотека. Функции для работы с консолью.

<code>congetch</code>	Вывод текста и ожидание нажатия клавиши.
<code>congetstr</code>	Получение строки после вывода текста.
<code>conread</code>	Получение строки от пользователя.
<code>conrequest</code>	Вывод многовариантного запроса на консоль.
<code>conyesno</code>	Вывод вопроса на консоль.

congetstr

Получение строки после вывода текста. Получение строки введенной пользователем с предварительным выводом текста.

```
func str congetstr (  
    str output,  
    str input  
)
```

Параметры

output Выводимый текст.
input Переменная типа str для получения данных.

Возвращаемое значение

Возвращается параметр *input*.

Смотрите также

- [Console](#)

conread

Получение строки от пользователя.

```
func str conread (  
    str input  
)
```

Параметры

input Переменная типа str для получения данных.

Возвращаемое значение

Возвращается параметр *input*.

Смотрите также

- [Console](#)

conrequest

Вывод многовариантного запроса на консоль.

```
func uint conrequest (  
    str output,  
    str answer  
)
```

Параметры

output Текст запроса.

answer Перечисление возможных букв для ответов. Варианты ответов разделяются '|'. Например, "Nn|Yy"

Возвращаемое значение

Функция возвращает номер выбранного варианта ответа с 0.

Смотрите также

- [Console](#)

conyesno

Вывод вопроса на консоль.

```
func uint conyesno (  
    str output  
)
```

Параметры

output Текст вопроса.

Возвращаемое значение

Функция возвращает 1 если ответ 'да' и 0 в противном случае.

Смотрите также

- [Console](#)

CSV

Работа с CSV данными. Переменные типа `csv` обеспечивают работу с данными в csv-формате.

```
string1_1,"string1_2",string1_3  
string2_1,"string2_2",string2_3
```

Тип `csv` наследуется от типа `str`, следовательно вы можете использовать [операторы и методы string](#). Для использования библиотеки необходимо с помощью команды `include` указать файл `csv.g`, который находится в поддиректории `lib/csv`.

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

- [Операторы](#)
- [Методы](#)

Операторы

`foreach var, csv`

Оператор `foreach`.

Методы

`csv.append` Добавить запись к объекту `csv`.

`csv.clear` Очистить объект данных `csv`.

`csv.read` Чтение данных из `csv` файла.

`csv.settings` Установка разделителя и ограничителей элементов для `csv` данных.

`csv.write` Запись `csv` данных в файл.

foreach var, csv

Оператор `foreach`. Перебор всех элементов с помощью оператора **foreach**. Элементом у объекта типа `csv` является массив строк **arrstr**. Каждая строка разбивается на отдельные элементы в соответствии с разделителем и эти элементы записываются в передаваемый массив.

```
csv mycsv
uint i k
...
foreach item, mycsv
{
    print( "Item: \(++i)\n" )
    fornum k = 0, *item
    {
        print( "\ (item[k])\n" )
    }
}
foreach variable, csv {...}
```

Смотрите также

- [CSV](#)

csv.append

Добавить запись к объекту csv.

```
method csv.append (  
  arrstr arrs  
)
```

Параметры

arrs Массив строк содержащий элементы записи. Все строки будут объединены в одну запись и добавлены к объекту **csv**.

Смотрите также

- [CSV](#)

csv.clear

Очистить объект данных csv.

```
method uint csv.clear()
```

Смотрите также

- [CSV](#)

csv.settings

Установка разделителя и ограничителей элементов для csv данных.

```
method csv.settings (  
  uint separ,  
  uint open,  
  uint close  
)
```

Параметры

separ Символ-разделитель. По умолчанию, запятая.
open Символ-ограничитель поля слева. По умолчанию, двойные кавычки.
close Символ-ограничитель поля справа. По умолчанию, двойные кавычки.

Смотрите также

- [CSV](#)

csv.write

Запись csv данных в файл.

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

Параметры

filename Имя файла для записи. Если файл существует, то он будет перезаписан.

Возвращаемое значение

Размер записанных данных.

Смотрите также

- [CSV](#)

Date & Time

Функции для работы с датами и временем.

- [Операторы](#)
- [Функции](#)
- [Методы](#)
- [Функции и операторы для времени файлов](#)
- [Типы](#)

Операторы

<code>datetime = datetime</code>	Операции присваивания.
<code>datetime += uint</code>	Добавление дней к дате.
<code>datetime -= uint</code>	Вычитание дней из даты.
<code>datetime - datetime</code>	Разность двух дат в днях и во времени.
<code>datetime + datetime</code>	Сложение двух структуры datetime.
<code>datetime == datetime</code>	Сравнение на равенство.
<code>datetime < datetime</code>	Проверка на меньше.
<code>datetime > datetime</code>	Проверка на больше.

Функции

<code>abbrnameofday</code>	Получить краткое название дня недели на языке пользователя.
<code>days</code>	Количество дней между двумя датами.
<code>daysinmonth</code>	Количество дней в месяце.
<code>firstdayofweek</code>	Получить первый день недели для региона пользователя.
<code>getdateformat</code>	Получение даты в соответствии с указанным форматом.
<code>getdatetime</code>	Получение времени и даты в виде строк.
<code>gettimeformat</code>	Получение времени в соответствии с указанным форматом.
<code>isleapyear</code>	Проверка на високосный год.
<code>nameofmonth</code>	Получить название месяца на языке пользователя.

Методы

<code>datetime.dayofweek</code>	Получить день недели.
<code>datetime.dayofyear</code>	Получить порядковый номер дня в году.
<code>datetime.fromstr</code>	Конвертирование строки вида ССММЧЧДДММГГГГ в структуру datetime.
<code>datetime.gettime</code>	Получить текущее время.
<code>datetime.getsystemtime</code>	Получить текущее системное время и дату.
<code>datetime.normalize</code>	Нормализация структуры datetime.
<code>datetime.setdate</code>	Установить дату для данной структуры datetime.
<code>datetime.tostr</code>	Конвертирование структуры datetime в строку вида ССММЧЧДДММГГГГ .

Функции и операторы для времени файлов

<code>filetime = filetime</code>	Присваивание объектов filetime.
<code>filetime == filetime</code>	Сравнение на равенство.
<code>filetime < filetime</code>	Проверка на меньше.

`filetime > filetime`

Проверка на больше.

`datetime toftime`

Сконвертировать дату из datetime в filetime.

`ftime todatetime`

Сконвертировать дату из filetime в datetime.

`getfiledatetime`

Получение даты и времени в виде строк.

Типы

`datetime`

Структура datetime.

`filetime`

Структура filetime.

datetime = datetime

Операции присваивания.

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

Возвращаемое значение

Результирующая структура datetime.

Смотрите также

- [Date & Time](#)

datetime += uint

Добавление дней к дате.

```
operator datetime += (  
    datetime left,  
    uint next  
)
```

Возвращаемое значение

Результирующий объект datetime.

Смотрите также

- [Date & Time](#)

datetime -= uint

Вычитание дней из даты.

```
operator datetime -= (  
    datetime left,  
    uint next  
)
```

Возвращаемое значение

Результирующий объект datetime.

Смотрите также

- [Date & Time](#)

datetime - datetime

- [operator datetime -<result>\(datetime left, datetime right \)](#)
- [operator datetime -=\(datetime left, datetime right \)](#)

Разность двух дат в днях и во времени. Все значения будут положительными.

```
operator datetime -<result> (  
    datetime left,  
    datetime right  
)
```

Возвращаемое значение

Результирующий объект datetime.

datetime -= datetime

Разность двух дат в днях и во времени. Все значения будут положительными.

```
operator datetime -= (  
    datetime left,  
    datetime right  
)
```

Возвращаемое значение

Результирующий объект datetime.

Смотрите также

- [Date & Time](#)

datetime + datetime

- [operator datetime +<result>\(datetime left, datetime right \)](#)
- [operator datetime +=\(datetime left, datetime right \)](#)

Сложение двух структуры datetime.

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

Возвращаемое значение

Результирующий объект datetime.

datetime += datetime

Добавление одной структуры datetime к другой.

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

Возвращаемое значение

Результирующий объект datetime.

Смотрите также

- [Date & Time](#)

datetime == datetime

- [operator uint ==\(datetime left, datetime right \)](#)
- [operator uint !=\(datetime left, datetime right \)](#)

Сравнение на равенство.

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

Возвращаемое значение

Возвращает **1** если объекты равны и **0** в противном случае.

datetime != datetime

Сравнение на равенство.

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

Возвращаемое значение

Возвращает **0** если объекты равны и **1** в противном случае.

Смотрите также

- [Date & Time](#)

datetime < datetime

- [operator uint <\(datetime left, datetime right \)](#)
- [operator uint <=\(datetime left, datetime right \)](#)

Проверка на меньше.

```
operator uint < (  
    datetime left,  
    datetime right  
)
```

Возвращаемое значение

Возвращает **1** если первое время меньше второго и **0** в противном случае.

datetime <= datetime

Проверка на меньше или равно.

```
operator uint <= (  
    datetime left,  
    datetime right  
)
```

Возвращаемое значение

Возвращает **1** если первое время меньше или равно второму и **0** в противном случае.

Смотрите также

- [Date & Time](#)

datetime > datetime

- [operator uint >\(datetime left, datetime right \)](#)
- [operator uint >=\(datetime left, datetime right \)](#)

Проверка на больше.

```
operator uint > (  
    datetime left,  
    datetime right  
)
```

Возвращаемое значение

Возвращает **1** если первое время больше второго и **0** в противном случае.

datetime >= datetime

Проверка на больше или равно.

```
operator uint >= (  
    datetime left,  
    datetime right  
)
```

Возвращаемое значение

Возвращает **1** если первое время больше или равно второму и **0** в противном случае.

Смотрите также

- [Date & Time](#)

abbrnameofday

Получить краткое название дня недели на языке пользователя.

```
func str abbrnameofday (  
    str ret,  
    uint dayofweek  
)
```

Параметры

ret Строка для получения результата.
dayofweek Номер дня недели. 0 - воскресенье, 1 - понедельник...

Возвращаемое значение

Возвращается параметр **ret**.

Смотрите также

- [Date & Time](#)

days

Количество дней между двумя датами.

```
func int days (  
    datetime left,  
    datetime right  
)
```

Параметры

left Первая сравниваемая дата.
right Вторая сравниваемая дата.

Возвращаемое значение

Возвращает количество дней между двумя датами. Если первая дата больше второй, то возвращаемое значение будет отрицательным.

Смотрите также

- [Date & Time](#)

daysinmonth

Количество дней в месяце. Для февраля учитывается високосный год.

```
func uint daysinmonth (  
    ushort year,  
    ushort month  
)
```

Параметры

year

Год.

month

Месяц.

Возвращаемое значение

Возвращает количество дней в месяце.

Смотрите также

- [Date & Time](#)

firstdayofweek

Получить первый день недели для региона пользователя.

```
func uint firstdayofweek()
```

Возвращаемое значение

Возвращает номер дня недели. 0 - воскресенье, 1 - понедельник...

Смотрите также

- [Date & Time](#)

getdateformat

Получение даты в соответствии с указанным форматом.

```
func str getdateformat (  
    datetime systime,  
    str format,  
    str date  
)
```

Параметры

systime Переменная содержащая дату.
format Формат представления даты. Может содержать следующие значения:

dd	День в виде числа.
ddd	День недели в виде аббревиатуры.
dddd	Полный день недели.
MM	Месяц в виде числа.
MMM	Месяц в виде аббревиатуры.
MMMM	Полное название месяца.
yy	Последние две цифры года.
yyyy	Год.

date Строка для получения даты.

Возвращаемое значение

Возвращается параметр **date**.

Смотрите также

- [Date & Time](#)

getdatetime

Получение времени и даты в виде строк. Получение даты и времени в текущем строковом формате Windows.

```
func getdatetime (  
    datetime systime,  
    str date,  
    str time  
)
```

Параметры

<i>systime</i>	Структура datetime.
<i>date</i>	Строка для получения даты. Может быть 0->str.
<i>time</i>	Строка для получения времени. Может быть 0->str.

Смотрите также

- [Date & Time](#)

gettimeformat

Получение времени в соответствии с указанным форматом.

```
func str gettimeformat (  
    datetime systime,  
    str format,  
    str time  
)
```

Параметры

systime Переменная содержащая время.

format Формат представления времени. Может содержать следующие значения:

hh	Часы - 12-и часовое время.
HH	Часы - 24-х часовое время.
mm	Минуты.
ss	Секунды.
tt	Временной маркер, такой как AM или PM.

time Строка для получения времени.

Возвращаемое значение

Возвращается параметр *time*.

Смотрите также

- [Date & Time](#)

nameofmonth

Получить название месяца на языке пользователя.

```
func str nameofmonth (  
    str ret,  
    uint month  
)
```

Параметры

ret Строка для получения результата.
month Номер месяца с 1.

Возвращаемое значение

Возвращается параметр *ret*.

Смотрите также

- [Date & Time](#)

datetime.dayofweek

Получить день недели.

```
method uint datetime.dayofweek
```

Возвращаемое значение

Возвращает день недели. 0 - воскресенье, 1 - понедельник...

Смотрите также

- [Date & Time](#)

datetime.dayofyear

Получить порядковый номер дня в году.

```
method uint datetime.dayofyear
```

Возвращаемое значение

Возвращает порядковый номер данного дня в году.

Смотрите также

- [Date & Time](#)

datetime.fromstr

Конвертирование строки вида **ССММЧЧДДММГГГГ** в структуру `datetime`.

```
method datetime datetime.fromstr (  
    str data  
)
```

Параметры

`data` Строка с исходными данными для конвертации.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Date & Time](#)

datetime.gettime

Получить текущее время. День недели устанавливается автоматически.

```
method datetime datetime.gettime()
```

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Date & Time](#)

datetime.getsystemtime

Получить текущее системное время и дату.

```
method datetime datetime.getsystemtime()
```

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Date & Time](#)

datetime.normalize

Нормализация структуры datetime. Например, если значение часа равно 32, то после нормализации оно будет равно 8 и значение дней увеличится на 1.

```
method datetime datetime.normalize()
```

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Date & Time](#)

datetime.setdefault

Установить дату для данной структуры datetime. День недели устанавливается автоматически.

```
method datetime datetime.setdefault (  
    uint day,  
    uint month,  
    uint year  
)
```

Параметры

<i>day</i>	День.
<i>month</i>	Месяц.
<i>year</i>	Год.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Date & Time](#)

datetime.tostr

Конвертирование структуры `datetime` в строку вида **ССММЧЧДДММГГГГ**.

```
method str datetime.tostr (  
    str ret  
)
```

Параметры

`ret` Строка для получения результата.

Возвращаемое значение

Возвращается параметр `ret`.

Смотрите также

- [Date & Time](#)

filetime = filetime

Присваивание объектов filetime.

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

Возвращаемое значение

Результирующая структура filetime.

Смотрите также

- [Date & Time](#)

datetime == datetime

- [operator datetime ==\(datetime left, datetime right \)](#)
- [operator datetime !=\(datetime left, datetime right \)](#)

Сравнение на равенство.

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

Возвращаемое значение

Возвращает **1** если объекты равны и **0** в противном случае.

datetime != datetime

Сравнение на равенство.

```
operator datetime != (  
    datetime left,  
    datetime right  
)
```

Возвращаемое значение

Возвращает **0** если объекты равны и **1** в противном случае.

Смотрите также

- [Date & Time](#)

filetime < filetime

- [operator uint <\(filetime left, filetime right \)](#)
- [operator uint <=\(filetime left, filetime right \)](#)

Проверка на меньше.

```
operator uint < (  
    filetime left,  
    filetime right  
)
```

Возвращаемое значение

Возвращает **1** если первое время меньше второго и **0** в противном случае.

filetime <= filetime

Проверка на меньше или равно.

```
operator uint <= (  
    filetime left,  
    filetime right  
)
```

Возвращаемое значение

Возвращает **1** если первое время меньше или равно второму и **0** в противном случае.

Смотрите также

- [Date & Time](#)

filetime > filetime

- [operator uint >\(filetime left, filetime right \)](#)
- [operator uint >=\(filetime left, filetime right \)](#)

Проверка на больше.

```
operator uint > (  
    filetime left,  
    filetime right  
)
```

Возвращаемое значение

Возвращает **1** если первое время больше второго и **0** в противном случае.

filetime >= filetime

Проверка на больше или равно.

```
operator uint >= (  
    filetime left,  
    filetime right  
)
```

Возвращаемое значение

Возвращает **1** если первое время больше или равно второму и **0** в противном случае.

Смотрите также

- [Date & Time](#)

datetimetoftime

Сконвертировать дату из datetime в filetime.

```
func uint datetimetoftime (  
    datetime dt,  
    filetime ft  
)
```

Параметры

dt Структура datetime.

ft Переменная типа filetime для получения результата.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Date & Time](#)

ftimetodatetime

Сконвертировать дату из `filetime` в `datetime`.

```
func datetime ftimetodatetime (  
    filetype ft,  
    datetime dt,  
    uint local  
)
```

Параметры

ft Структура типа `filetime`. Может быть взята из структуры `finfo`.
dt Структура `datetime` для получения результата.
local Укажите 1 если необходимо учитывать локальное время.

Возвращаемое значение

Возвращается параметр `dt`.

Смотрите также

- [Date & Time](#)

getfiledatetime

Получение даты и времени в виде строк. Получить дату и время последнего изменения файла в виде строк.

```
func getfiledatetime (  
    filetype ftime,  
    str date,  
    str time  
)
```

Параметры

ftime Структура типа filetype. Может быть взята из структуры finfo.
date Строка для записи даты. Может быть 0->str.
time Строка для записи времени. Может быть 0->str.

Смотрите также

- [Date & Time](#)

datetime

Структура `datetime`. Для работы со временем используется объект типа `datetime`. Этот тип может содержать информацию о дате и времени.

```
type datetime
{
  ushort year
  ushort month
  ushort dayofweek
  ushort day
  ushort hour
  ushort minute
  ushort second
  ushort msec
}
```

Поля типа

<code>year</code>	Год.
<code>month</code>	Месяц.
<code>dayofweek</code>	День недели. Отсчет идет с 0. 0 - воскресенье, 1 - понедельник...
<code>day</code>	День.
<code>hour</code>	Часы.
<code>minute</code>	Минуты.
<code>second</code>	Секунды.
<code>msec</code>	Миллисекунды.

Смотрите также

- [Date & Time](#)

filetime

Структура filetime. Для работы со временем файлов используется тип filetime.

```
type filetime
{
    uint lowdtimе
    uint highdtimе
}
```

Поля типа

<i>lowdtimе</i>	Нижнее значение uint.
<i>highdtimе</i>	Верхнее значение uint.

Смотрите также

- [Date & Time](#)

Dbf

Данная библиотека предназначена для работы с **dbf** файлами. Поддерживаются форматы **dBase III** и **dBase IV**. Для использования библиотеки необходимо с помощью команды `include` указать файл `dbf.g`, который находится в поддиректории `lib`.

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

- [Операторы](#)
- [Методы](#)
- [Методы для полей](#)

Операторы

<code>* dbf</code>	Получить количество записей в базе данных.
<code>foreach var,dbf</code>	Оператор <code>foreach</code> .

Методы

<code>dbf.append</code>	Добавить запись.
<code>dbf.bof</code>	Проверка на первую запись.
<code>dbf.bottom</code>	Встать на последнюю запись.
<code>dbf.close</code>	Закрыть базу данных.
<code>dbf.create</code>	Создать dbf файл и открыть его.
<code>dbf.del</code>	Поставить/снять пометку удаления у текущей записи.
<code>dbf.empty</code>	Создать пустую копию.
<code>dbf.eof</code>	Проверка выхода за пределы базы.
<code>dbf.geterror</code>	Получить код ошибки.
<code>dbf.go</code>	Встать на запись с указанным номером.
<code>dbf.isdel</code>	Получить признак удаления записи.
<code>dbf.open</code>	Открыть базу данных (dbf файл).
<code>dbf.pack</code>	Упаковать базу данных.
<code>dbf.recno</code>	Номер текущей записи или 0, если запись не определена.
<code>dbf.skip</code>	Сдвиг текущей записи.
<code>dbf.top</code>	Встать на первую запись.

Методы для полей

<code>dbf.f_count</code>	Количество полей.
<code>dbf.f_date</code>	Получить дату.
<code>dbf.f_decimal</code>	Получить размер дробной части у числового поля.
<code>dbf.f_double</code>	Получить числовое значение.
<code>dbf.f_find</code>	Получить номер поля по имени.
<code>dbf.f_int</code>	Получить значение в виде целого числа.
<code>dbf.f_logic</code>	Получить значение логического поля у текущей записи.
<code>dbf.f_memo</code>	Получить значение мемо-поля у текущей записи.
<code>dbf.f_name</code>	Получить имя указанного поля.
<code>dbf.f_offset</code>	Получить смещение данного поля.

<code>dbf.f_ptr</code>	Указатель на данные.
<code>dbf.f_str</code>	Получить значение.
<code>dbf.f_type</code>	Получить тип поля.
<code>dbf.f_width</code>	Получить ширину указанного поля.
<code>dbf.fw_date</code>	Записать дату.
<code>dbf.fw_double</code>	Записать числовое значение.
<code>dbf.fw_int</code>	Записать целое число.
<code>dbf.fw_logic</code>	Записать логическое значение.
<code>dbf.fw_memo</code>	Записать значение в мемо-поле.
<code>dbf.fw_str</code>	Записать значение.

* dbf

Получить количество записей в базе данных.

```
operator uint * (  
    dbf dbase  
)
```

Возвращаемое значение

Количество записей.

Смотрите также

- [Dbf](#)

foreach var,dbf

Оператор foreach. Вы можете использовать оператор **foreach** для перебора в всех записей в базе данных. Значение **variable** является номером текущей записи.

```
foreach variable,dbf {...}
```

Смотрите также

- [Dbf](#)

dbf.append

Добавить запись. Метод добавляет запись в базу данных.

```
method uint dbf.append()
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Dbf](#)

dbf.bof

Проверка на первую запись. Определить, является ли текущая запись первой.

```
method uint dbf.bof()
```

Возвращаемое значение

Возвращается 1 если текущая запись первая.

Смотрите также

- [Dbf](#)

dbf.bottom

Встать на последнюю запись.

```
method uint dbf.bottom()
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Dbf](#)

dbf.close

Закрывает базу данных.

`method dbf.close()`

Смотрите также

- [Dbf](#)

dbf.create

Создать dbf файл и открыть его..

```
method uint dbf.create (  
    str filename,  
    str fields,  
    uint ver  
)
```

Параметры

filena Имя создаваемого dbf файла.

me

fields Описание полей базы данных. Строка в которой через перенос строки или ';' перечислены описание полей: Имя поля,Тип поля,Ширина,Размер дробной части для числового типа. Имя поля не может быть больше 10 символов. Тип поля может быть:

\$DBFF_CHAR	Строка.
\$DBFF_DATE	Дата.
\$DBFF_LOGIC	Логическое.
\$DBFF_NUMERIC	Целое.
\$DBFF_FLOAT	Действительное.
\$DBFF_MEMO	Мето поле.

ver Версия. 0 для dBase III или 1 для dBase IV.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Dbf](#)

dbf.del

Поставить/снять пометку удаления у текущей записи.

```
method uint dbf.del
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Dbf](#)

dbf.empty

Создать пустую копию. Метод создает такую же, но пустую базу данных.

```
method uint dbf.empty (  
    str outfile  
)
```

Параметры

filename Полное имя создаваемого dbf файла.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Dbf](#)

dbf.eof

Проверка выхода за пределы базы.

```
method uint dbf.eof (  
    fordata fd  
)
```

Параметры

fd Параметр используется для оператора foreach. Укажите 0->fordata.

Возвращаемое значение

Возвращает 1 если текущая запись не определена/не установлена и 0 в противном случае.

Смотрите также

- [Dbf](#)

dbf.geterror

Получить код ошибки. Получить код ошибки в случае неудачного завершения какого-либо метода.

```
method uint dbf.geterror()
```

Возвращаемое значение

Возвращается код последней ошибки.

<code>\$ERRDBF_OPEN</code>	Невозможно открыть dbf файл.
<code>\$ERRDBF_READ</code>	Невозможно прочитать dbf файл.
<code>\$ERRDBF_POS</code>	Ошибка позиционирования в файле.
<code>\$ERRDBF_EOF</code>	Нет текущей записи.
<code>\$ERRDBF_WRITE</code>	Ошибка записи в dbf файл.
<code>\$ERRDBF_FOVER</code>	Длина записываемой строки больше размера поля.
<code>\$ERRDBF_TYPE</code>	Несовместимый тип поля.
<code>\$ERRDBT_OPEN</code>	Невозможно открыть dbt файл.
<code>\$ERRDBT_READ</code>	Невозможно прочитать dbt файл.
<code>\$ERRDBT_POS</code>	Ошибка позиционирования в dbt файле.
<code>\$ERRDBT_WRITE</code>	Ошибка записи в dbt файл.

Смотрите также

- [Dbf](#)

dbf.go

Встать на запись с указанным номером.

```
method uint dbf.go (  
    uint num  
)
```

Параметры

num Требуемый номер записи с 1.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Dbf](#)

dbf.isdel

Получить признак удаления записи. Определить, помечена ли текущая запись как удаленная.

```
method uint dbf.isdel()
```

Возвращаемое значение

Возвращается 1 если текущая запись помечена как удаленная.

Смотрите также

- [Dbf](#)

dbf.open

Открыть базу данных (dbf файл).

```
method uint dbf.open (  
    str name  
)
```

Параметры

name Имя открываемого dbf файла.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Dbf](#)

dbf.recno

Номер текущей записи или 0, если запись не определена.

```
method uint dbf.recno()
```

Возвращаемое значение

Номер текущей записи или 0, если запись не определена.

Смотрите также

- [Dbf](#)

dbf.skip

Сдвиг текущей записи. Сдвинуться вперед или назад на указанное количество записей.

```
method uint dbf.skip (  
    int step  
)
```

Параметры

step Шаг сдвига. Если меньше нуля, то сдвиг будет к началу базы.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Dbf](#)

dbf.top

Встать на первую запись.

```
method uint dbf.top()
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Dbf](#)

dbf.f_count

Количество полей.

```
method uint dbf.f_count()
```

Возвращаемое значение

Возвращает количество полей.

Смотрите также

- [Dbf](#)

dbf.f_date

- [method datetime dbf.f_date\(datetime dt, uint num \)](#)
- [method str dbf.f_date\(str val, uint num \)](#)

Получить дату. Получение даты из указанного поля у текущей записи в структуру [datetime](#).

```
method datetime dbf.f_date (  
    datetime dt,  
    uint num  
)
```

Параметры

dt Структура для получения даты.

num Номер поля с 1.

Возвращаемое значение

Возвращается параметр *dt*.

dbf.f_date

Получение даты из указанного поля у текущей записи в виде строки.

```
method str dbf.f_date (  
    str val,  
    uint num  
)
```

Параметры

val Строка для получения даты.

num Номер поля с 1.

Возвращаемое значение

Возвращается параметр *val*.

Смотрите также

- [Dbf](#)

dbf.f_decimal

Получить размер дробной части у числового поля.

```
method uint dbf.f_decimal (  
    uint num  
)
```

Параметры

num Номер поля с 1.

Возвращаемое значение

Размер дробной части.

Смотрите также

- [Dbf](#)

dbf.f_double

Получить числовое значение. Получение числового значения типа double из указанного поля у текущей записи.

```
method double dbf.f_double (  
    uint num  
)
```

Параметры

num Номер поля с 1.

Возвращаемое значение

Значение типа double.

Смотрите также

- [Dbf](#)

dbf.f_int

Получить значение в виде целого числа. Получить числовое значение типа int из указанного поля текущей записи.

```
method int dbf.f_int (  
    uint num  
)
```

Параметры

num Номер поля с 1.

Возвращаемое значение

Возвращается число типа int.

Смотрите также

- [Dbf](#)

dbf.f_logic

Получить значение логического поля у текущей записи.

```
method uint dbf.f_logic (  
    uint num  
)
```

Параметры

num Номер поля с 1.

Возвращаемое значение

Возвращает значение логического поля.

\$DBF_LFALSE	Значение логического поля FALSE.
\$DBF_LTRUE	Значение логического поля TRUE.
\$DBF_LUNKNOWN	Значение логического поля неопределено.

Смотрите также

- [Dbf](#)

dbf.f_memo

Получить значение мемо-поля у текущей записи.

```
method uint dbf.f_memo (  
  str val,  
  uint num  
)
```

Параметры

val Строка для записи значения.
num Номер поля с 1.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Dbf](#)

dbf.f_name

Получить имя указанного поля.

```
method str dbf.f_name (  
    uint num  
)
```

Параметры

num Номер поля с 1.

Возвращаемое значение

Возвращает имя указанного поля.

Смотрите также

- [Dbf](#)

dbf.f_offset

Получить смещение данного поля.

```
method uint dbf.f_offset (  
    uint num  
)
```

Параметры

num Номер поля с 1.

Возвращаемое значение

Возвращает смещение данного поля.

Смотрите также

- [Dbf](#)

dbf.f_str

Получить значение. Получить в виде строки значение поля у текущей записи.

```
method str dbf.f_str (  
  str val,  
  uint num  
)
```

Параметры

val Строка для получения значения.

num Номер поля с 1.

Возвращаемое значение

Возвращается параметр *val*.

Смотрите также

- [Dbf](#)

dbf.f_type

Получить тип поля.

```
method uint dbf.f_type (  
    uint num  
)
```

Параметры

num Номер поля с 1.

Возвращаемое значение

Возвращает тип данного поля. Можно использовать следующие значения:

<code>\$DBFF_CHAR</code>	Строка.
<code>\$DBFF_DATE</code>	Дата.
<code>\$DBFF_LOGIC</code>	Логическое.
<code>\$DBFF_NUMERIC</code>	Целое.
<code>\$DBFF_FLOAT</code>	Действительное.
<code>\$DBFF_MEMO</code>	Мето поле.

Смотрите также

- [Dbf](#)

dbf.f_width

Получить ширину указанного поля.

```
method uint dbf.f_width (  
  uint num  
)
```

Параметры

num Номер поля с 1.

Возвращаемое значение

Возвращает ширину поля.

Смотрите также

- [Dbf](#)

dbf.fw_date

Записать дату. Записать логическое значение в указанное поле у текущей записи.

```
method uint dbf.fw_date (  
    datetime dt,  
    uint num  
)
```

Параметры

dt Структура [datetime](#) содержащая дату.

num Номер поля с 1.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Dbf](#)

dbf.fw_double

Записать числовое значение. Записать числовое значение в указанное поле у текущей записи.

```
method uint dbf.fw_double (  
    double dval,  
    uint num  
)
```

Параметры

dval Записываемое число.
num Номер поля с 1.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Dbf](#)

dbf.fw_int

Записать целое число. Записать значение типа int в указанное поле у текущей записи.

```
method uint dbf.fw_int (  
    int ival,  
    uint num  
)
```

Параметры

ival Записываемое число.
num Номер поля с 1.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Dbf](#)

dbf.fw_logic

Записать логическое значение. Записать логическое значение в указанное поле у текущей записи.

```
method uint dbf.fw_logic (  
    uint val,  
    uint num  
)
```

Параметры

val Число 1 или 0.
num Номер поля с 1.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Dbf](#)

dbf.fw_memo

Записать значение в мемо-поле. Записать значение в указанное мемо-поле у текущей записи.

```
method uint dbf.fw_memo (  
  str val,  
  uint num  
)
```

Параметры

val Записываемая строка.
num Номер поля с 1.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Dbf](#)

dbf.fw_str

Записать значение. Записать значение в указанное поле у текущей записи.

```
method uint dbf.fw_str (  
  str val,  
  uint num  
)
```

Параметры

val Записываемая строка.

num Номер поля с 1.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Dbf](#)

Files

Функции для работы с файловой системой.

- [Методы](#)
- [Функции](#)
- [Функции поиска и получения информации](#)
- [Дополнительные методы](#)

Методы

<code>file.close</code>	Закреть файл.
<code>file.getsize</code>	Получить размер файла.
<code>file.gettime</code>	Получить время последнего изменения файла.
<code>file.open</code>	Открыть файл.
<code>file.read</code>	Чтение файла.
<code>file.setpos</code>	Установить текущую позицию в файле.
<code>file.settime</code>	Установить время файла.
<code>file.write</code>	Запись в файл.

Функции

<code>copyfile</code>	Скопировать файл.
<code>copyfiles</code>	Копирование файлов и директорий по маске.
<code>createdir</code>	Создать директорию.
<code>deletedir</code>	Удалить директорию.
<code>deletefile</code>	Удалить файл.
<code>delfiles</code>	Удаление файлов и директорий по маске.
<code>direxist</code>	Проверка существования директории.
<code>fileexist</code>	Проверка существования файла.
<code>getcurdir</code>	Получить текущую директорию.
<code>getdrives</code>	Получить имена доступных дисков.
<code>getdrivetype</code>	Получить тип диска.
<code>getfileattrib</code>	Получение атрибутов файла.
<code>getmodulename</code>	Получить имя файла у текущего запущенного приложения.
<code>getmodulepath</code>	Получить путь где расположен запущенный EXE файл.
<code>gettempdir</code>	Получить временную директорию приложения.
<code>isqualfiles</code>	Проверить файлы на равенство.
<code>movefile</code>	Переименовать, переместить файл или директорию.
<code>setattribnormal</code>	Установка атрибута <code>\$FILE_ATTRIBUTE_NORMAL</code> .
<code>setcurdir</code>	Установить текущую директорию.
<code>setfileattrib</code>	Установить атрибуты файла.
<code>verifypath</code>	Проверка пути и создание всех отсутствующих директорий.

Функции поиска и получения информации

<code>info</code>	Структура информации о файле.
<code>ffind</code>	Структура для поиска файлов.
<code>foreach var,ffind</code>	Оператор foreach.
<code>ffind.init</code>	Инициализация поиска файлов.
<code>getfileinfo</code>	Получить информацию о файле или директории.

Дополнительные методы

<code>arrstr.read</code>	Прочитать текстовый файл в массив строк.
<code>arrstr.write</code>	Записать массив строк в многостроковый текстовый файл.
<code>buf.read</code>	Чтение из файла.
<code>buf.write</code>	Запись в файл.
<code>buf.writeappend</code>	Дописать данные к файлу.
<code>str.read</code>	Чтение строки из файла.
<code>str.write</code>	Запись строки в файл.
<code>str.writeappend</code>	Добавление строки к файлу.

file.close

Закреть файл.

```
method uint file.close( )
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Files](#)

file.getsize

Получить размер файла.

```
method uint file.getsize( )
```

Возвращаемое значение

Размер файла меньше 4GB.

Смотрите также

- [Files](#)

file.gettime

Получить время последнего изменения файла.

```
method uint file.gettime (  
    filetype ft  
)
```

Параметры

ft Переменная для получения времени файла.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Files](#)

file.open

Открыть файл.

```
method uint file.open (  
    str name,  
    uint flag  
)
```

Параметры

name Имя открываемого файла.

flag Возможно использование следующих флагов.

\$OP_READONLY	Открывать только для чтения.
\$OP_EXCLUSIVE	Открывать в защищенном режиме.
\$OP_CREATE	Создать файл.
\$OP_ALWAYS	Создать файл только если он не существует.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Files](#)

file.read

- [method uint file.read\(uint ptr, uint size \)](#)
- [method uint file.read\(buf rbuf, uint size \)](#)

Чтение файла.

```
method uint file.read (  
    uint ptr,  
    uint size  
)
```

Параметры

ptr Указатель куда будут записаны прочитанные данные.

size Размер читаемых данных.

Возвращаемое значение

Функция возвращает размер прочитанных данных.

file.read

Чтение файла.

```
method uint file.read (  
    buf rbuf,  
    uint size  
)
```

Параметры

rbuf Буфер куда будут читаться данные. Чтение идет в режиме добавления данных в буфер. То есть прочитанные данные будут добавлены к уже существующим в буфере.

size Размер читаемых данных.

Возвращаемое значение

Функция возвращает размер прочитанных данных.

Смотрите также

- [Files](#)

file.setpos

Установить текущую позицию в файле.

```
method uint file.setpos (  
    int offset,  
    uint mode  
)
```

Параметры

offset

Смещение позиции.

mode

Тип по которому изменяется смещение.

\$FILE_BEGIN

От начала файла.

\$FILE_CURRENT

От текущей позиции.

\$FILE_END

С конца файла.

Возвращаемое значение

Функция возвращает текущую позицию в файле.

Смотрите также

- [Files](#)

file.settime

Установить время файла.

```
method uint file.settime (  
    filetype ft  
)
```

Параметры

ft Новое время файла.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Files](#)

file.write

- [method uint file.write\(uint data, uint size \)](#)
- [method uint file.write\(buf rbuf \)](#)
- [method uint file.writepos\(uint pos, uint data, uint size \)](#)

Запись в файл.

```
method uint file.write (
    uint data,
    uint size
)
```

Параметры

data Указатель на данные которые будут записаны.
size Размер записываемых данных.

Возвращаемое значение

Метод возвращает размер записанных данных.

file.write

Запись в файл.

```
method uint file.write (
    buf rbuf
)
```

Параметры

rbuf Буфер, который будет записан.

Возвращаемое значение

Метод возвращает размер записанных данных.

file.writepos

Запись в файл от текущей позиции.

```
method uint file.writepos (
    uint pos,
    uint data,
    uint size
)
```

Параметры

pos Начальное смещение для записи.
data Указатель на данные которые будут записаны.
size Размер записываемых данных.

Возвращаемое значение

Метод возвращает размер записанных данных.

Смотрите также

- [Files](#)

copyfile

Скопировать файл.

```
func uint copyfile (  
    str name,  
    str newname  
)
```

Параметры

name Имя существующего файла.

newname Новое имя и путь файла. Если файл уже существует, он будет перезаписан.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Files](#)

copyfiles

- [func uint copyfiles\(str src, str dir, uint flag, uint mode, uint process \)](#)
- [func uint defcopyproc\(uint code, uint left, uint right \)](#)

Копирование файлов и директорий по маске.

```
func uint copyfiles (
    str src,
    str dir,
    uint flag,
    uint mode,
    uint process
)
```

Параметры

src Имя или маска копируемых файлов или директорий.

dir Директория, куда будут скопированы файлы.

flag Комбинация флагов поиска и копирования.

\$FIND_DIR	Искать только директории.
\$FIND_FILE	Искать только файлы.
\$FIND_RECURSE	Рекурсивный поиск.
\$COPYF_RO	Копировать поверх read-only файлов.
\$COPYF_SAVEPATH	Сохранять относительные пути.
\$COPYF_ASK	Спрашивать перед копированием.

mode Что делать если копируемый файл уже существует exists.

\$COPY_OVER	Копировать поверх.
\$COPY_SKIP	Пропустить.
\$COPY_NEWER	Копировать если новее.
\$COPY_MODIFIED	Копировать если разные.

process Идентификатор функции обработчика. Вы можете использовать **&defcopyproc** в качестве функции-обработчика по умолчанию.

Возвращаемое значение

Возвращает 1 если копирование прошло успешно, в противном случае возвращается 0.

defcopyproc

Это функция-обработчик по умолчанию для функции **copyfiles**. Вы можете определить и указать свою собственную функцию обработчик подобную этой. develop and use your own process function like it.

```
func uint defcopyproc (
    uint code,
    uint left,
    uint right
)
```

Параметры

code Код сообщения.

\$COPYN_FOUND	Найден объект для копирования.
\$COPYN_NEWDIR	Создана директория.
\$COPYN_ERRDIR	Невозможно создать директорию.
\$COPYN_ASK	Запрос на копирование.
\$COPYN_ERRFILE	Ошибка создания файла.
\$COPYN_NEWFILE	Создан файл.
\$COPYN_BEGIN	Начало копирования.
\$COPYN_PROCESS	Идет копирование.

	\$COPYN_END	Конец копирования.
	\$COPYN_ERRWRITE	Ошибка записи файла.
<i>left</i>	Дополнительный параметр.	
<i>right</i>	Дополнительный параметр.	

Возвращаемое значение

Функция должна возвращать одно из следующих значений:

\$COPYR_NOTHING	Ничего не делать.
\$COPYR_BREAK	Прервать копирование.
\$COPYR_RETRY	Повторить попытку.
\$COPYR_SKIP	Перейти к следующему.
\$COPYR_OVER	Копировать поверх.
\$COPYR_OVERALL	Копировать поверх все.
\$COPYR_SKIPALL	Пропустить все.

Смотрите также

- [Files](#)

createdir

Создать директорию.

```
func uint createdir (  
    str name  
)
```

Параметры

name Имя создаваемой директории.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Files](#)

deletedir

Удалить директорию.

```
func uint deletedir (  
    str name  
)
```

Параметры

name Имя удаляемой директории.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Files](#)

deletefile

Удалить файл.

```
func uint deletefile (  
    str name  
)
```

Параметры

name Имя удаляемого файла.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Files](#)

delfiles

Удаление файлов и директорий по маске. Директории удаляются вместе со всеми файлами и поддиректориями. Будьте очень внимательны при использовании этой функции. Например, вызов

```
delfiles( "c:\\temp", $FIND_DIR | $FIND_FILE | $FIND_RECURSE )
```

удалит все файлы и директории с именем temp на диске C: включая поиск во всех поддиректориях. В данном случае temp рассматривается как маска, а так как указан флаг \$FIND_RECURSE, то поиск будет вестись по всему диску C:. Если надо просто удалить директорию temp со всеми ее поддиректориями и файлами, то вызов должен быть таким

```
delfiles( "c:\\temp", $FIND_DIR )
```

Вызов

```
delfiles( "c:\\temp\\*.tmp", $FIND_FILE )
```

удалит все файлы в директории с расширением tmp исключая поддиректории.

```
func delfiles (
    str name,
    uint flag
)
```

Параметры

name Имя или маска для поиска.

flag Комбинация флагов поиска и удаления.

\$FIND_DIR	Искать только директории.
\$FIND_FILE	Искать только файлы.
\$FIND_RECURSE	Рекурсивный поиск.
	Удалять файлы с атрибутом 'Только для чтения'.
\$DELF_RO	

Смотрите также

- [Files](#)

getcurdir

Получить текущую директорию.

```
func str getcurdir (  
    str dir  
)
```

Параметры

dir Строка для получения результата.

Возвращаемое значение

Возвращается параметр *dir*.

Смотрите также

- [Files](#)

getdrives

Получить имена доступных дисков.

```
func arrstr getdrives <result>()
```

Возвращаемое значение

Массив (arrstr) имен дисков.

Смотрите также

- [Files](#)

getdrivetype

Получить тип диска.

```
func uint getdrivetype (  
    str name  
)
```

Параметры

drive Имя диска с заключительным слэшем. Например: C:\

Возвращаемое значение

Возвращает одно из следующих значений:

\$DRIVE_UNKNOWN	Тип неизвестен.
\$DRIVE_NO_ROOT_DIR	Неверный корневой путь.
\$DRIVE_REMOVABLE	Извлекаемый диск.
\$DRIVE_FIXED	Постоянный диск.
\$DRIVE_REMOTE	Сетевой диск.
\$DRIVE_CDROM	CD/DVD-ROM диск.
\$DRIVE_RAMDISK	RAM диск.

Смотрите также

- [Files](#)

getmodulename

Получить имя файла у текущего запущенного приложения.

```
func str getmodulename (  
    str dest  
)
```

Параметры

dest Строка для получения имени.

Возвращаемое значение

Возвращается параметр **dest**.

Смотрите также

- [Files](#)

getmodulepath

Получить путь где расположен запущенный EXE файл.

```
func str getmodulepath (  
    str dest,  
    str subfolder  
)
```

Параметры

dest Строка для получения результата.

subfolder Дополнительный путь. Эта строка будет добавлена к полученному результату. Может быть пустой строкой.

Возвращаемое значение

Возвращается параметр **dest**.

Смотрите также

- [Files](#)

gettempdir

Получить временную директорию приложения. При первом вызове этой функции во временной директории пользователя будет создана директория с именем genteeXX, где XX уникальный номер для данного запущенного приложения. При закрытии приложения директория будет удалена вместе со всеми файлами.

```
func str gettempdir (  
    str dir  
)
```

Параметры

dir Строка для получения результата.

Возвращаемое значение

Возвращается параметр *dir*.

Смотрите также

- [Files](#)

isequalfiles

Проверить файлы на равенство. Сравнить содержимое двух файлов.

```
func uint isequalfiles (  
    str left,  
    str right  
)
```

Параметры

left Имя первого сравниваемого файла.
right Имя второго сравниваемого файла.

Возвращаемое значение

Возвращает 1 если файлы равны, в противном случае возвращается 0.

Смотрите также

- [Files](#)

movefile

Переименовать, переместить файл или директорию.

```
func uint movefile (  
    str name,  
    str newname  
)
```

Параметры

name Имя существующего файла или директории.
newname Новое имя и путь файла или директории.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Files](#)

setcurdir

Установить текущую директорию.

```
func uint setcurdir (  
    str dir  
)
```

Параметры

dir Имя новой текущей директории.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Files](#)

setfileattrib

Установить атрибуты файла.

```
func uint setfileattrib (  
    str name,  
    uint attrib  
)
```

Параметры

name Имя файла.
attrib Атрибуты файла.

<code>\$FILE_ATTRIBUTE_READONLY</code>	Только чтение.
<code>\$FILE_ATTRIBUTE_HIDDEN</code>	Скрытый.
<code>\$FILE_ATTRIBUTE_SYSTEM</code>	Системный.
<code>\$FILE_ATTRIBUTE_DIRECTORY</code>	Директория.
<code>\$FILE_ATTRIBUTE_ARCHIVE</code>	Архивный.
<code>\$FILE_ATTRIBUTE_NORMAL</code>	Нормальный.
<code>\$FILE_ATTRIBUTE_TEMPORARY</code>	Временный.
<code>\$FILE_ATTRIBUTE_COMPRESSED</code>	Сжатый.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Files](#)

verifypath

Проверка пути и создание в всех отсутствующих директорий.

```
func uint verifypath (  
    str name,  
    arrstr dirs  
)
```

Параметры

name Имя проверяемого пути.

dirs Массив для получения всех создаваемых директорий. Может быть 0->arrstr.

Возвращаемое значение

Функция возвращает 1 если проверка и создание директорий прошли успешно. В случае ошибки возвращается 0 и последний элемент *dirs* содержит имя, на котором произошла ошибка создания директории.

Смотрите также

- [Files](#)

finfo

Структура информации о файле. Эта структура используется в функции [getfileinfo](#) и операторе [foreach](#).

```
type finfo
{
    str fullname
    str name
    uint attrib
    filetype created
    filetype lastwrite
    filetype lastaccess
    uint sizehi
    uint sizelo
}
```

Поля типа

<i>fullname</i>	Полное имя файла или директории.
<i>name</i>	Имя файла или директории.
<i>attrib</i>	Аттрибуты файла.
<i>created</i>	Время создания.
<i>lastwrite</i>	Время последнего изменения.
<i>lastaccess</i>	Время последнего обращения.
<i>sizehi</i>	Старший uint размера.
<i>sizelo</i>	Младший uint размера.

Смотрите также

- [Files](#)

ffind

Структура для поиска файлов. Этот тип используется в операторе [foreach](#). Вы должны инициализировать переменную с помощью метода [ffind.init](#) и не должны самостоятельно изменять поля.

```
type ffind <index = finfo>
{
  stack deep
  str initname
  str wildcard
  uint flag
}
```

Поля типа

<i>deep</i>	Скрытые данные.
<i>initname</i>	Скрытые данные.
<i>wildcard</i>	Скрытые данные.
<i>flag</i>	Скрытые данные.

Смотрите также

- [Files](#)

foreach var,ffind

Оператор `foreach`. Вы можете использовать оператор **foreach** для перебора файлов по определенной маске в директории. Структура [finfo](#) возвращается для каждого найденного файла. Вы должны вызвать метод [ffind.init](#) перед использованием **foreach**.

```
ffind fd
fd.init( "c:\\*.exe", $FIND_FILE | $FIND_RECURSE )
foreach finfo cur,fd
{
    print( "\\( cur.fullname )\n" )
}
foreach variable,ffind {...}
```

Смотрите также

- [Files](#)

ffind.init

Инициализация поиска файлов. Для поиска файлов или директорий по маске используется объект типа `ffind`. Перед началом поиска надо вызвать метод инициализации. После этого можно использовать инициализированный объект в цикле `foreach`. На каждом найденном файле будет возвращаться структура [finfo](#).

```
method ffind.init (  
    str name,  
    uint flag  
)
```

Параметры

name Маска для поиска файлов или директорий.
flag Комбинация следующих флагов:

<code>\$FIND_DIR</code>	Искать только директории.
<code>\$FIND_FILE</code>	Искать только файлы.
<code>\$FIND_RECURSE</code>	Рекурсивный поиск.

Смотрите также

- [Files](#)

getfileinfo

Получить информацию о файле или директории.

```
func uint getfileinfo (  
    str name,  
    finfo fi  
)
```

Параметры

name Имя файла или директории.

fi Структура [finfo](#) в которую будет записана вся информация.

Возвращаемое значение

Возвращает 1 если файл был найден и 0 в противном.

Смотрите также

- [Files](#)

FTP

FTP протокол. Вы должны вызвать функцию [inet_init](#) перед использованием библиотеки. Для использования библиотеки необходимо с помощью команды `include` указать файл `ftp.g`, который находится в поддиректории `lib\ftp`.

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

- [Общие интернет функции](#)
- [URL строки](#)

ftp.close	Закрывает FTP соединение.
ftp.command	Послать команду.
ftp.createdir	Создать директорию.
ftp.deldir	Удалить директорию.
ftp.delfile	Удалить файл.
ftp.getcurdir	Получить текущую директорию.
ftp.getfile	Скачать файл.
ftp.getsize	Получить размер файла на FTP сервере.
ftp.gettime	Получить время файла.
ftp.lastresponse	Последний ответ FTP-сервера.
ftp.list	Список файлов.
ftp.open	Открыть FTP соединение.
ftp.putfile	Выложить файл на FTP-сервер.
ftp.rename	Переименовать файл.
ftp.setattrib	Установить атрибуты.
ftp.setcurdir	Установить текущую директорию.

Общие интернет функции

inet_close	Закончить работу.
inet_error	Получить код ошибки.
inet_init	Инициализация библиотеки.
inet_proxy	Использовать прокси-сервер.
inet_proxyenable	Включить/отключить прокси-сервер.
inetnotify_func	Функция обработки сообщений.

URL строки

str.iencoding	Перекодировка строки.
str.ihead	Получить заголовок.
str.ihttpinfo	Обработать заголовок.
str.iurl	Метод служит для разбора URL адреса на составляющие.

ftp.close

Закрыть FTP соединение. Метод закрывает FTP соединение с сервером.

```
method uint ftp.close()
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [FTP](#)

ftp.command

Послать команду. Метод посылает указанную команду FTP серверу. Ответ можно получить с помощью метода [ftp.lastresponse](#).

```
method uint ftp.command (  
    str cmd  
)
```

Параметры

cmd Текст посылаемой команды.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [FTP](#)

ftp.createdir

Создать директорию. Метод позволяет создать директорию на FTP сервере.

```
method uint ftp.createdir (  
    str dirname  
)
```

Параметры

dirname Имя создаваемой директории.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [FTP](#)

ftp.deldir

Удалить директорию. Метод удаляет директорию на FTP сервере.

```
method uint ftp.deldir (  
    str dirname  
)
```

Параметры

dirname Имя удаляемой директории.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [FTP](#)

ftp.getcurdir

Получить текущую директорию. Метод получает имя текущей директории на FTP сервере.

```
method uint ftp.getcurdir (  
    str dirname  
)
```

Параметры

dirname Строка для получения результата.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [FTP](#)

ftp.getfile

- [method uint ftp.getfile\(str filename, buf databuf, uint flag \)](#)
- [method uint ftp.getfile\(str srcname, str destname, uint flag \)](#)

Скачать файл. Метод позволяет скачивать файлы с FTP сервера.

```
method uint ftp.getfile (  
    str filename,  
    buf databuf,  
    uint flag  
)
```

Параметры

filename Имя скачиваемого файла.
databuf Буфер для получения данных. Скачивание будет происходить без записи на диск.
flag Флаги.

\$FTP_BINARY	Скачивать как двоичный файл
\$FTP_TEXT	Скачивать как текстовый файл. Режим по умолчанию. Добавлять 0 в конец полученных данных.
\$FTP_STR	

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

ftp.getfile

Метод позволяет скачивать файлы с FTP сервера.

```
method uint ftp.getfile (  
    str srcname,  
    str destname,  
    uint flag  
)
```

Параметры

srcname Имя скачиваемого файла.
destname Имя файла для записи.
flag Флаги.

\$FTP_BINARY	Скачивать как двоичный файл
\$FTP_TEXT	Скачивать как текстовый файл. Режим по умолчанию. Продолжить скачивание.
\$FTP_CONTINUE	
\$FTP_SETTIME	Установить время как на FTP-сервере.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [FTP](#)

ftp.getsize

Получить размер файла на FTP сервере.

```
method uint ftp.getsize (  
    str name,  
    uint psize  
)
```

Параметры

name Имя файла.
psize Указатель на uint для получения значения размера.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [FTP](#)

ftp.gettime

Получить время файла. Получить время последнего изменения файла на FTP сервере.

```
method uint ftp.gettime (  
    str name,  
    datetime dt  
)
```

Параметры

name Имя файла.

dt Переменная типа [datetime](#) для получения времени.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [FTP](#)

ftp.lastresponse

Последний ответ FTP-сервера. Метод позволяет получить последний ответ FTP сервера.

```
method str ftp.lastresponse (  
    str out  
)
```

Параметры

out Строка для получения результата.

Возвращаемое значение

Возвращается параметр *out*.

Смотрите также

- [FTP](#)

ftp.list

Список файлов. Метод позволяет получить список файлов и директорий на FTP сервере.

```
method uint ftp.list (  
    str data,  
    str mode  
)
```

Параметры

list Строка для получения результата.
cmd Команда для получения списка.

"LIST"	Список в формате команды LIST.
"NLST"	Список в виде имен без дополнительной информации.
"MLSD"	Список в формате команды MLSD.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [FTP](#)

ftp.open

Открыть FTP соединение. Метод открывает FTP соединение с сервером. Метод должен быть вызван до вызова прочих методов работы с FTP сервером.

```
method uint ftp.open (  
    str url,  
    str user,  
    str password,  
    uint flag,  
    str notify  
)
```

Параметры

<i>url</i>	Имя или адрес FTP сервера.				
<i>user</i>	Имя пользователя. Если указана пустая строка, то подключение будет анонимным.				
<i>password</i>	Пароль пользователя. При анонимном подключении укажите email.				
<i>d</i>					
<i>flag</i>	Флаги подключения. <table border="1"><tr><td>\$FTP_ANONYM</td><td>Анонимное подключение.</td></tr><tr><td>\$FTP_PASV</td><td>Подключение в пассивном режиме.</td></tr></table>	\$FTP_ANONYM	Анонимное подключение.	\$FTP_PASV	Подключение в пассивном режиме.
\$FTP_ANONYM	Анонимное подключение.				
\$FTP_PASV	Подключение в пассивном режиме.				
<i>notify</i>	Функция для получения уведомлений. Может быть 0.				

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [FTP](#)

ftp.putfile

Выложить файл на FTP-сервер. Метод позволяет закачивать файл с локальной машины на FTP сервер.

```
method uint ftp.putfile (  
    str srcname,  
    str destname,  
    uint flag  
)
```

Параметры

srcname Имя исходного закачиваемого файла.
destname Имя файла записываемого файла на FTP сервере.
flag Флаги. Если не указан флаг двоичного или текстового режима, то метод определяет тип файла самостоятельно.

\$FTP_BINARY	Закачивать как двоичный файл.
\$FTP_TEXT	Закачивать как текстовый файл.
\$FTP_CONTINUE	Продолжить закачивание.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [FTP](#)

ftp.rename

Переименовать файл. Метод переименовывает файл или директорию на FTP сервере.

```
method uint ftp.rename (  
    str from,  
    str to  
)
```

Параметры

from Текущее имя файла или директории.

to Новое имя.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [FTP](#)

ftp.setattrib

Установить атрибуты. Метод устанавливает атрибуты у файла или директории.

```
method uint ftp.setattrib (  
    str name,  
    uint mode  
)
```

Параметры

name Имя файла или директории.
mode Устанавливаемые атрибуты.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [FTP](#)

ftp.setcurdir

Установить текущую директорию. Метод устанавливает новую текущую директорию.

```
method uint ftp.setcurdir (  
    str dirname  
)
```

Параметры

dirname Имя новой текущей директории.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [FTP](#)

Gentee API

Для разработчиков программного обеспечения имеется возможность выполнять программы на языке Gentee в своих приложениях. Для этого достаточно подключить файл `gentee.dll`. Он содержит несколько импортируемых функций, которые отвечают за компиляцию и выполнение программ.

- [Типы](#)

gentee_call	Вызов функции из байт-кода.
gentee_compile	Компиляция Gentee программы.
gentee_deinit	Окончание работы с gentee.
gentee_getid	Получить идентификатор объекта по его имени.
gentee_init	Инициализация gentee.
gentee_load	Загрузить и запустить байт-код.
gentee_ptr	Получить Gentee структуры.
gentee_set	Эта функция устанавливает некоторые gentee параметры.

Типы

gentee	Главная структура gentee.
compileinfo	Структура для передачи в функцию gentee_compile .
optimize	Структура для использования в типе compileinfo .

gentee_call

Вызов функции из байт-кода. Байт-код должен быть до этого загружен с помощью функций [gentee_load](#) или [gentee_compile](#).

```
uint CDECLCALL gentee_call (  
    uint id,  
    puint result,  
    ...  
)
```

Параметры

id Идентификатор вызываемого объекта. Может быть получен с помощью функции [gentee_getid](#).
result Указатель на область памяти куда будет записан результат. Это может быть указатель на **uint**, **long** или **double**.
... Требуемые параметры функции.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Gentee API](#)

gentee_compile

Компиляция Gentee программы. Эта функция позволяет компилировать и запускать программы на Gentee.

```
uint STDCALL gentee_compile (  
    pcompileinfo compinit  
)
```

Параметры

compinit Указатель на структуру [compileinfo](#) с определенными параметрами компиляции.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Gentee API](#)

gentee_deinit

Окончание работы с gentee.dll. Эта функция должна быть вызвана когда работа с Gentee закончена.

```
uint STDCALL gentee_deinit( void )
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Gentee API](#)

gentee_getid

Получить идентификатор объекта по его имени.

```
uint CDECLCALL gentee_getid (  
    pubyte name,  
    uint count,  
    ...  
)
```

Параметры

name Имя объекта. Если вы хотите найти метод, то добавляйте слева от имени '@'. Например, "@myMethod". Если вы хотите найти оператор, то добавляйте слева от имени '#'. Например, "#+=".

count Количество следующих далее параметров. Если вы хотите найти любой объект с данным именем, то укажите следующий флаг.

GID_ANYOBJ	Искать любой объект.
-------------------	----------------------

... Укажите последовательность идентификаторов типов параметров. Если кроме типов имеется подтип с "of" то укажите его значение в старшем слове (HIWORD).

Возвращаемое значение

Возвращается идентификатор объекта или 0, если объект не был найден.

Смотрите также

- [Gentee API](#)

gentee_init

Инициализация gentee.dll. Эта функция должна вызываться пенред началом работы с Gentee.

```
uint STDCALL gentee_init (  
    uint flags  
)
```

Параметры

flags

Флаги.

G_CONSOLE	Консольное приложение.
G_SILENT	Не показывать служебных сообщений.
G_CHARPRN	Выводить в Window s кодировке.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Gentee API](#)

gentee_load

Загрузить и запустить байт-код. Эта функция загружает байт-код из файла или из памяти и запускает его если это необходимо. Вы можете создавать байт-код с помощью функции [gentee_compile](#).

```
uint STDCALL gentee_load (  
    pubyte bytecode,  
    uint flag  
)
```

Параметры

bytecode Указатель на байт-код или имя .ge файл.
de
flag Флаги.

GLOAD_ARGS	Автоматически получить аргументы командной строки.
GLOAD_FILE	Прочитать байт-код из файла. <i>bytecode</i> в этом случае должен содержать имя загружаемого файла.
GLOAD_RUN	Запускать <code><entry></code> и <code><main></code> функции.
G_ASM	Конвертировать байт-код в ассемблер "на лету".

Возвращаемое значение

Возвращает результат выполнения байт-кода если был указан флаг GLOAD_RUN.

Смотрите также

- [Gentee API](#)

gentee_ptr

Получить Gentee структуры. Эта функция возвращает указатели на глобальные Gentee структуры.

```
pvoid STDCALL gentee_ptr (  
    uint par  
)
```

Параметры

par Идентификатор получаемого значения.

GPTR_GENTEE	Указатель на gentee структуру. Смотрите gentee .
GPTR_VM	Указатель на vm структуру
GPTR_COMPILE	Указатель на compile структуру

Возвращаемое значение

Указатель на соответствующую Gentee структуру.

Смотрите также

- [Gentee API](#)

gentee_set

Эта функция устанавливает некоторые gentee параметры.

```
uint STDCALL gentee_set (  
    uint state,  
    pvoid val  
)
```

Параметры

state Идентификатор устанавливаемого параметра.

GSET_TEMPDIR	Указать свою временную директорию
GSET_PRINT	Указать свою print функцию
GSET_MESSAGE	Указать свою message функцию
GSET_EXPORT	Указать свою export функцию
GSET_ARGS	Указать параметры командной строки
GSET_FLAG	Указать флаги
GSET_DEBUG	Указать свою debug функцию
GSET_GETCH	Указать свою getch функцию

val Устанавливаемое значение параметра.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Gentee API](#)

gentee

Главная структура gentee.

```
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;
```

Поля типа

<i>flags</i>	Флаги.
	G_CONSOLE Консольное приложение.
	G_SILENT Не показывать служебных сообщений.
	G_CHARPRN Выводить в Windows кодировке.
<i>multib</i>	1 если текущая кодировка двухбайтовая.
<i>tempid</i>	Идентификатор временной директории.
<i>tempdir</i>	Временная директория.
<i>tempfile</i>	Идентификатор файла для защиты временной директории.
<i>print</i>	print функция не по умолчанию.
<i>getch</i>	getch и scan не по умолчанию.
<i>message</i>	message функция не по умолчанию.
<i>export</i>	export функция.
<i>debug</i>	debug функция.
<i>args</i>	Аргументы командной строки. arg1 0 arg2 00

Смотрите также

- [Gentee API](#)

compileinfo

Структура для передачи в функцию [gentee_compile](#).

```
typedef struct
{
    pubyte input;
    uint flag;
    pubyte libdirs;
    pubyte include;
    pubyte defargs;
    pubyte output;
    pvoid hthread;
    uint result;
    optimize opti;
} compileinfo, * pcompileinfo;
```

Поля типа

input Имя Gentee файла. Вы можете указать текст Gentee программы если определите флаг CMPL_SRC.

flag Флаги компиляции.

CMPL_SRC	Укажите если compileinfo.input указывает на исходный текст.
CMPL_NORUN	Ничего не запускать после компиляции.
CMPL_GE	Создавать GE файл с байт-кодом.
CMPL_LINE	Обрабатывать #! в первой строке.
CMPL_DEBUG	Компиляции с добавлением отладочной информации.
CMPL_THREAD	Компиляция в отдельном потоке.
CMPL_NOWAIT	Не ждать окончания компиляции в потоке. Использовать только с CMPL_THREAD.
CMPL_OPTIMIZE	Оптимизировать результирующий байт-код (GE файл).
CMPL_NOCLEAR	Учитывать (не очищать) при компиляции существующие объекты в виртуальной машине.
CMPL_ASM	Конвертировать байт-код в ассемблер.

libdirs Директории для поиска файлов: name1 0 name2 0 ... 00. Может равно NULL.

include Include файлы: name1 0 name2 0 ... 00. Эти файлы будут откомпилированы в начале процесса компиляции. Может равно NULL.

defargs Define аргументы: name1 0 name2 0 ... 00. Вы можете указать дополнительные макроопределения. Например, **MYMODE = 10**. В этом случае вы можете использовать макрос **\$MYMODE** в Gentee программе. Может быть NULL.

output Имя результирующего GE файла. По умолчанию, .ge файл создается в той же директории что и главный .g файл. Вы можете указать любой путь и имя для результирующего файла с байт-кодом. Вы должны определить флаг CMPL_GE чтобы создавался файл с байт-кодом.

hthread Возвратится идентификатор потока если вы определили флаги CMPL_THREAD | CMPL_NOWAIT.

ad

result Результат выполнения программы если она была запущена.

t

opti Структура оптимизации. Используется если определен флаг CMPL_OPTIMIZE.

Смотрите также

- [Gentee API](#)

optimize

Структура для использования в типе [compileinfo](#).

```
typedef struct
{
    uint flag;
    pubyte nameson;
    pubyte avoidon;
} optimize, * poptimize;
```

Поля типа

flag

Флаги оптимизации.

OPTI_DEFINE Удалять 'define' объекты.

OPTI_NAME Удалять имена объектов.

OPTI_AVOID Удалять не используемые объекты.

OPTI_MAIN Оставлять только последнюю main функцию. Используется с OPTI_AVOID.

nameson

Не удалять имена со следующими масками разделенных 0 если определен флаг OPTI_NAME.

on

avoidon

Не удалять объекты со следующими масками разделенных 0 если определен флаг OPTI_AVOID.

on

Смотрите также

- [Gentee API](#)

Hash

Хэш (Ассоциативный массив). Переменные типа hash обеспечивают работу с ассоциативными массивами или хэш-таблицами. Каждому элементу такого массива соответствует уникальная строка-ключ. Обращение к элементам происходит с помощью указания соответствующей строки-ключа.

- [Операторы](#)
- [Методы](#)
- [Типы](#)

Операторы

<code>hash of type</code>	Указание типа элементов.
<code>* hash</code>	Получить количество элементов.
<code>hash[name]</code>	Получение элемента по строке-ключу.
<code>foreach var,hash</code>	Оператор foreach.

Методы

<code>hash.clear</code>	Очистить хэш-массив.
<code>hash.create</code>	Создать элемент с данным ключом.
<code>hash.del</code>	Удалить элемент с данным ключом.
<code>hash.find</code>	Найти элемент с данным ключом.
<code>hash.ignorecase</code>	Игнорировать регистр ключей.
<code>hash.sethashtablesize</code>	Установить размер таблицы значений для поиска ключей.

Типы

<code>hash</code>	Главная структура типа hash.
-------------------	------------------------------

hash of type

Указание типа элементов. Вы можете определять тип элементов массива с помощью оператора **of** когда вы описываете переменную типа **hash**. По умолчанию, элементы массива имеют тип **uint**.

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

Смотрите также

- [Hash](#)

* hash

Получить количество элементов.

```
operator uint * (  
    hash left  
)
```

Возвращаемое значение

Количество элементов в хэш-массиве.

Смотрите также

- [Hash](#)

hash[name]

Получение элемента по строке-ключу. В случае отсутствия элемент создается автоматически.

```
method uint hash.index (  
  str key  
)
```

Возвращаемое значение

Элемент ["key"] из хэш-массива.

Смотрите также

- [Hash](#)

foreach var,hash

- [foreach variable.hash {...}](#)
- [foreach variable.hash.keys {...}](#)

Оператор foreach. Вы можете использовать оператор **foreach** для перебора в всех элементов хэш-массива. **Variable** является указателем на элемент хэш-массива.

```
foreach variable,hash {...}
```

foreach var,hash.keys

Вы можете использовать оператор **foreach** для перебора в всех ключей хэш-массива.

```
foreach variable,hash.keys {...}
```

Смотрите также

- [Hash](#)

hash.clear

Очистить хэш-массив. Метод удаляет все элементы массива.

```
method hash.clear()
```

Смотрите также

- [Hash](#)

hash.del

Удалить элемент с данным ключом.

```
method uint hash.del (  
  str key  
)
```

Параметры

key Строка-ключ.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Hash](#)

hash.find

Найти элемент с данным ключом.

```
method uint hash.find (  
  str key  
)
```

Параметры

key Строка-ключ.

Возвращаемое значение

Возвращается указатель на найденный элемент или 0, если элемент с таким ключом отсутствует.

Смотрите также

- [Hash](#)

hash.ignorecase

Игнорировать регистр ключей. Работать с ключами данной хэш-таблицы без учета регистра. Метод должен вызываться до добавления любых элементов.

```
method uint hash.ignorecase
```

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Hash](#)

hash.sethasize

Установить размер таблицы значений для поиска ключей. Метод должен вызываться до добавления любых элементов. В параметре указывается степень 2 для вычисления размера таблицы, так как количество элементов должно быть степенью 2. Увеличение размера таблицы приводит к уменьшению количества коллизий, и увеличению требуемой памяти. По умолчанию, размер таблицы равен 4096 элементам.

```
method uint hash.sethasize (  
  uint power  
)
```

Параметры

power Степень двойки для вычисления размера таблицы.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Hash](#)

hash

Главная структура типа hash.

```
type hash
{
  arr hashes
  uint itype
  uint isize
  uint count
  uint igncase
  hkeys keys
}
```

Поля типа

<i>hashes</i>	Массив хэш значений. Указатели на hashkey.
<i>itype</i>	Тип элементов.
<i>isize</i>	Размер одного элемента.
<i>count</i>	Количество элементов.
<i>igncase</i>	Равно 1 если игнорируется регистр.
<i>keys</i>	Структура для просмотра ключей.

Смотрите также

- [Hash](#)

HTTP

HTTP протокол. Вы должны вызвать функцию [inet_init](#) перед использованием библиотеки. Для использования библиотеки необходимо с помощью команды `include` указать файл `http.g`, который находится в поддиректории `lib\http`.

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

- [Общие интернет функции](#)
- [URL строки](#)

http_get	Получить данные по протоколу HTTP.
http_getfile	Скачать файл по протоколу HTTP.
http_head	Получить заголовок по протоколу HTTP.
http_post	Отправить данные по протоколу HTTP.

Общие интернет функции

inet_close	Закончить работу.
inet_error	Получить код ошибки.
inet_init	Инициализация библиотеки.
inet_proxy	Использовать прокси-сервер.
inet_proxyenable	Включить/отключить прокси-сервер.
inetnotify_func	Функция обработки сообщений.

URL строки

str.ieencoding	Перекодировка строки.
str.ihead	Получить заголовок.
str.ihttpinfo	Обработать заголовок.
str.iurl	Метод служит для разбора URL адреса на составляющие.

http_get

Получить данные по протоколу HTTP. Метод посылает запрос GET по указанному URL и записывает полученные данные в буфер databuf.

```
func uint http_get (  
    str url,  
    buf databuf,  
    uint notify,  
    uint flag  
)
```

Параметры

url URL адрес из которого будут получены данные.
databuf Буфер для получения данных.
uf
notif [Функция](#) для получения уведомлений. Может быть 0.
y
flag Флаги.

\$HTTPF_REDIRECT	Если используется перенаправление адреса, то скачивать по новому адресу.
\$HTTPF_STR	Добавлять 0 к databuf после получения данных. Используйте этот флаг если databuf является строкой.
\$HTTPF_CONTINUE	Если файл уже существует, то производить докачивание. Используется в функции http_getfile .
\$HTTPF_SETTIME	Установить у файла время такое же как на сервере. Используется в функции http_getfile .

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [HTTP](#)

http_getfile

Скачать файл по протоколу HTTP. Метод посылает запрос GET по указанному URL и записывает полученные данные в указанный файл.

```
func uint http_getfile (  
    str url,  
    str filename,  
    uint notify,  
    uint flag  
)
```

Параметры

url URL адрес для скачивания.
filena Имя файла для записи.
me
notify [Функция](#) для получения уведомлений. Может быть 0.
flag Флаги.

\$HTTPF_REDIRECT	Если используется перенаправление адреса, то скачивать по новому адресу.
\$HTTPF_STR	Добавлять 0 к databuf после получения данных. Используйте этот флаг если databuf является строкой.
\$HTTPF_CONTINUE	Если файл уже существует, то производить докачивание. Используется в функции http_getfile .
\$HTTPF_SETTIME	Установить у файла время такое же как на сервере. Используется в функции http_getfile .

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [HTTP](#)

http_head

Получить заголовок по протоколу HTTP. Метод посылает запрос HEAD по указанному URL адресу и делает частичный разбор полученных данных.

```
func uint http_head (  
    str url,  
    str head,  
    httpinfo hi  
)
```

Параметры

url URL адрес для получения заголовка.
head Строка для получения текста заголовка.
hi Переменная типа [httpinfo](#) для получения информации о заголовке.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [HTTP](#)

http_post

Отправить данные по протоколу HTTP. Метод посылает запрос POST с указанной строкой по данному URL адресу. Служит для автоматического заполнения форм.

```
func uint http_post (  
    str url,  
    str data,  
    str result,  
    uint notify  
)
```

Параметры

url URL адрес, куда будут отправлены данные.

data Строка отправляемых данных. Перед отправкой необходимо провести перекодировку строк-параметров запроса с помощью метода [str.iencoding](#).

result Строка для получения ответа сервера.

notify [Функция](#) для получения уведомлений. Может быть 0.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [HTTP](#)

INI File

INI файлы. Данная библиотека позволяет работать с ini-файлами. Переменные типа ini обеспечивают работу с ними. Для использования библиотеки необходимо с помощью команды include указать файл ini.g, который находится в поддиректории lib\ini.

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

- [Методы](#)
- [Функции](#)

Методы

ini.delkey	Удалить ключ.
ini.delsection	Удалить секцию.
ini.getnum	Получить числовое значение ключа.
ini.getvalue	Получить значение ключа.
ini.keys	Получить список ключей данной секции.
ini.read	Прочитать данные из файла.
ini.sections	Получить список секций.
ini.setnum	Записать числовое значение ключа.
ini.setvalue	Записать значение ключа.
ini.write	Сохранить данные в ini-файле.

Функции

inigetval	Получить значение ключа из ini-файла.
inisetval	Записать значение ключа в ini-файл.

ini.delkey

Удалить ключ.

```
method ini.delkey (  
    str section,  
    str key  
)
```

Параметры

section

Имя секции.

key

Имя удаляемого ключа.

Смотрите также

- [INI File](#)

ini.getnum

Получить числовое значение ключа.

```
method uint ini.getnum (  
    str section,  
    str key,  
    uint defvalue  
)
```

Параметры

<i>section</i>	Имя секции.
<i>key</i>	Имя ключа.
<i>defval</i>	Присваиваемое значение если ключ не найден.

Возвращаемое значение

Числовое значение ключа.

Смотрите также

- [INIFile](#)

ini.getvalue

Получить значение ключа.

```
method uint ini.getvalue (  
  str section,  
  str key,  
  str value,  
  str defvalue  
)
```

Параметры

<i>section</i>	Имя секции.
<i>key</i>	Имя ключа.
<i>value</i>	Строка для получения значения.
<i>defval</i>	Присваиваемое значение если ключ не найден.

Возвращаемое значение

Возвращет 1 если ключ был найден и 0 в противном случае.

Смотрите также

- [INIFile](#)

ini.keys

Получить список ключей данной секции. Все ключи будут записаны в массив строк.

```
method arrstr ini.keys (  
    str section,  
    arrstr ret  
)
```

Параметры

section Имя секции.

ret Массив строк куда будут записаны наименования ключей.

Возвращаемое значение

Возвращается параметр **ret**.

Смотрите также

- [INI File](#)

ini.sections

Получить список секций. Все секции будут записаны в массив строк.

```
method arrstr ini.sections (  
  arrstr ret  
)
```

Параметры

ret Массив строк куда будут записаны наименования секций.

Возвращаемое значение

Возвращается параметр *ret*.

Смотрите также

- [INI File](#)

ini.setnum

Записать числовое значение ключа.

```
method ini.setnum (  
    str section,  
    str key,  
    uint value  
)
```

Параметры

<i>section</i>	Имя секции.
<i>key</i>	Имя ключа.
<i>value</i>	Записываемое значение ключа.

Смотрите также

- [INI File](#)

ini.setvalue

Записать значение ключа.

```
method ini.setvalue (  
    str section,  
    str key,  
    str value  
)
```

Параметры

<i>section</i>	Имя секции.
<i>key</i>	Имя ключа.
<i>value</i>	Записываемое значение ключа.

Смотрите также

- [INI File](#)

inigetval

Получить значение ключа из ini-файла.

```
func str inigetval (  
    str ininame,  
    str section,  
    str key,  
    str value,  
    str defval  
)
```

Параметры

ininame Имя ini-файла.

section Имя секции.

key Имя ключа.

value Строка для записи значения.

defval Значение, которое будет подставлено в случае ошибки или отсутствия данного ключа.

Возвращаемое значение

Возвращается параметр *value*.

Смотрите также

- [INIFile](#)

inisetval

Записать значение ключа в ini-файл.

```
func uint inisetval (  
    str ininame,  
    str section,  
    str key,  
    str value  
)
```

Параметры

<i>ininame</i>	Имя ini-файла.
<i>section</i>	Имя секции.
<i>key</i>	Имя ключа.
<i>value</i>	Записываемое значение ключа.

Возвращаемое значение

#InIretf

Смотрите также

- [INI File](#)

Keyboard

Данные функции предназначены для эмуляции работы клавиатуры. Для использования библиотеки необходимо с помощью команды `include` указать файл `keyboard.g`, который находится в поддиректории `lib\keyboard`.

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

<code>sendstr</code>	Произвести ввод строки с клавиатуры.
----------------------	--------------------------------------

<code>sendvkey</code>	Нажать клавишу.
-----------------------	-----------------

sendstr

Произвести ввод строки с клавиатуры.

```
func uint sendstr (  
    str input  
)
```

Параметры

data Строка для ввода с клавиатуры.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Keyboard](#)

sendvkey

Нажать клавишу. Нажать клавишу одну или в комбинации с **Shift**, **Ctrl**, **Alt**.

```
func uint sendvkey (  
    ushort vkey,  
    uint flag  
)
```

Параметры

vkey Виртуальный код клавиши.

flag Флаги для нажатия дополнительных клавиш.

<code>\$\$VK_SHIFT</code>	Нажата клавиша Shift .
<code>\$\$VK_ALT</code>	Нажата клавиша Alt .
<code>\$\$VK_CONTROL</code>	Нажата клавиша Ctrl .

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Keyboard](#)

Math

Математические функции.

abs	Абсолютное значение для целых чисел $ x $.
acos	Вычисление арккосинуса.
asin	Вычисление арксинуса.
atan	Вычисление арктангенса.
ceil	Наименьшее целое double не меньше данного.
cos	Вычисление косинуса.
exp	Экспоненциальная функция.
fabs	Абсолютное значение для double $ x $.
floor	Наибольшее целое double не больше данного.
ln	Натуральный логарифм.
log	Десятичный логарифм.
modf	Разделение на целую и дробную часть.
pow	Возведение в степень.
sin	Вычисление синуса.
sqrt	Квадратный корень.
tan	Вычисление тангенса.

abs

Абсолютное значение для целых чисел $|x|$.

```
func uint abs (  
    int x  
)
```

Параметры

x Целое число.

Возвращаемое значение

Абсолютное значение.

Смотрите также

- [Math](#)

acos

Вычисление арккосинуса.

```
func double acos (  
    double x  
)
```

Параметры

x Значение для вычисления арккосинуса.

Возвращаемое значение

Арккосинус *x* в диапазоне [0; π].

Смотрите также

- [Math](#)

asin

Вычисление арксинуса.

```
func double asin (  
    double x  
)
```

Параметры

x Значение для вычисления арксинуса.

Возвращаемое значение

Арксинус x в диапазоне $[-\pi/2 ; \pi/2]$.

Смотрите также

- [Math](#)

atan

Вычисление арктангенса.

```
func double atan (  
    double x  
)
```

Параметры

x Значение для вычисления арктангенса.

Возвращаемое значение

Арктангенс x в диапазоне $[-\pi/2; \pi/2]$.

Смотрите также

- [Math](#)

ceil

Получение наименьшего целого в виде double, которое не меньше данного числа.

```
func double ceil (  
    double x  
)
```

Параметры

`x` Значение с плавающей точкой.

Возвращаемое значение

Ближайшее наименьшее целое.

Смотрите также

- [Math](#)

COS

Вычисление косинуса.

```
func double cos (  
    double x  
)
```

Параметры

x Угол в радианах.

Возвращаемое значение

Косинус x .

Смотрите также

- [Math](#)

exp

Экспоненциальная функция.

```
func double exp (  
    double x  
)
```

Параметры

x Степень числа e .

Возвращаемое значение

Число e в степени x .

Смотрите также

- [Math](#)

fabs

Абсолютное значение для double |x|.

```
func double fabs (  
    double x  
)
```

Параметры

x Значение с плавающей точкой.

Возвращаемое значение

Абсолютное значение.

Смотрите также

- [Math](#)

floor

Получение наибольшего целого в виде double, которое меньше или равно данному числу.

```
func double floor (  
    double x  
)
```

Параметры

`x` Значение с плавающей точкой.

Возвращаемое значение

Ближайшее наибольшее целое.

Смотрите также

- [Math](#)

In

Натуральный логарифм.

```
func double ln (  
    double x  
)
```

Параметры

x Значение с плавающей точкой.

Возвращаемое значение

Натуральный логарифм $\ln(x)$.

Смотрите также

- [Math](#)

log

Десятичный логарифм.

```
func double log (  
    double x  
)
```

Параметры

x Значение с плавающей точкой.

Возвращаемое значение

Десятичный логарифм $\log_{10}(x)$.

Смотрите также

- [Math](#)

modf

Разделение на целую и дробную часть.

```
func double modf (  
    double x,  
    uint y  
)
```

Параметры

x Значение с плавающей точкой.
y Указатель на double для получения целой части.

Возвращаемое значение

Дробная часть *x*.

Смотрите также

- [Math](#)

pow

Возведение в степень.

```
func double pow (  
    double x,  
    double y  
)
```

Параметры

x Основание.

y Степень.

Возвращаемое значение

Возведение x в степень y .

Смотрите также

- [Math](#)

sin

Вычисление синуса.

```
func double sin (  
    double x  
)
```

Параметры

x Угол в радианах.

Возвращаемое значение

Синус x .

Смотрите также

- [Math](#)

sqrt

Квадратный корень.

```
func double sqrt (  
    double x  
)
```

Параметры

x Положительное значение с плавающей точкой.

Возвращаемое значение

Квадратный корень *x*.

Смотрите также

- [Math](#)

tan

Вычисление тангенса.

```
func double tan (  
    double x  
)
```

Параметры

x Угол в радианах.

Возвращаемое значение

Тангенс x .

Смотрите также

- [Math](#)

Memory

Gentee имеет собственный менеджер памяти. Здесь описаны возможности предоставляемые Gentee для работы с памятью. Вы можете отводить и использовать память с помощью этих функций.

<code>m alloc</code>	Отвести память.
<code>m cmp</code>	Сравнение памяти.
<code>m copy</code>	Копирование памяти.
<code>m free</code>	Освобождение памяти.
<code>m len</code>	Размер до нуля.
<code>m move</code>	Сдвиг памяти.
<code>m zero</code>	Заполнение нулями.

malloc

Отвести память. Функция отводит память указанного размера.

```
func uint malloc (  
    uint size  
)
```

Параметры

size Размер отводимого блока памяти.

Возвращаемое значение

Указатель на отведенный блок памяти или 0 в случае ошибки.

Смотрите также

- [Memory](#)

mcmp

Сравнение памяти. Функция сравнивает два участка памяти.

```
func int mcmp (  
    uint dest,  
    uint src,  
    uint len  
)
```

Параметры

dest Указатель на первый участок памяти.
src Указатель на второй участок памяти.
len Сравнимый размер.

Возвращаемое значение

0	Участки равны.
<0	Первый участок меньше.
>0	Второй участок меньше.

Смотрите также

- [Memory](#)

memcpy

Копирование памяти. Функция копирует данные из одного блока памяти в другой.

```
func uint memcpy (  
    uint dest,  
    uint src,  
    uint len  
)
```

Параметры

dest Указатель для копируемых данных.
src Указатель на источник копируемых данных.
len Размер копируемых данных.

Возвращаемое значение

Указатель на скопированные данные.

Смотрите также

- [Memory](#)

mfree

Освобождение памяти. Функция освобождает память.

```
func uint mfree (  
    uint ptr  
)
```

Параметры

ptr Указатель на освобождаемый блок памяти.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Memory](#)

mlen

Размер до нуля. Определяет количество байт до нуля.

```
func uint mlen (  
    uint data  
)
```

Параметры

data Указатель на участок памяти.

Возвращаемое значение

Количество байт до нулевого символа.

Смотрите также

- [Memory](#)

mmove

Сдвиг памяти. Функция сдвигает указанный участок. Исходные и конечные данные могут перекрываться.

```
func mmove (  
    uint dest,  
    uint src,  
    uint len  
)
```

Параметры

dest Указатель для копируемых данных.
src Указатель на источник копируемых данных.
len Размер копируемых данных.

Смотрите также

- [Memory](#)

mzero

Заполнение нулями. Функция обнуляет участок памяти.

```
func uint mzero (  
    uint dest,  
    uint len  
)
```

Параметры

dest Указатель на участок памяти.
len Размер обнуляемых данных.

Возвращаемое значение

Указатель на обнуленные данные.

Смотрите также

- [Memory](#)

ODBC (SQL)

Работа с базами данных через ODBC (SQL запросы). Данная библиотека предназначена для выполнения SQL запросов к базам данных с помощью ODBC. В текущей версии не поддерживается работа с параметрами запросов. Смотрите [Описание ODBC](#) для более подробной информации. Для использования библиотеки необходимо с помощью команды `include` указать файл `odbc.g`, который находится в поддиректории `lib\odbc`.

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

- [Методы](#)
- [Методы для работы с SQL запросами](#)
- [Методы для работы с полями](#)

Описание ODBC	Краткое описание ODBC библиотеки.
----------------------	-----------------------------------

Методы

odbc.connect	Соединение с базой данных.
odbc.disconnect	Отсоединение от базы данных.
odbc.geterror	Получить описание последней ошибки.
odbc.newquery	Создать новый запрос ODBC.

Методы для работы с SQL запросами

odbcquery.active	Проверить наличие результирующего набора после выполнения SQL запроса.
odbcquery.close	Завершить работу с набором данных.
odbcquery.fieldbyname	Получить поле записи по его имени.
odbcquery.first	Перейти на первую запись в результирующем наборе.
odbcquery.geterror	Получить описание последней ошибки.
odbcquery.getrecordcount	Количество записей в результирующем наборе.
odbcquery.last	Перейти на последнюю запись в результирующем наборе.
odbcquery.moveby	Перейти на указанное количество записей в результирующем наборе.
odbcquery.next	Перейти на следующую запись в результирующем наборе.
odbcquery.prior	Перейти на предыдущую запись в результирующем наборе.
odbcquery.run	Выполнить SQL запрос.
odbcquery.settimeout	Установить время отводимое на выполнение запроса.

Методы для работы с полями

odbcfield.getbuf	Метод для получения значения поля в виде буфера.
odbcfield.getdatetime	Получить значение поля в виде даты и времени - тип <code>datetime</code> .
odbcfield.getdouble	Метод для получения значения поля в виде числа с плавающей точкой.
odbcfield.getindex	Получить порядковый номер поля.
odbcfield.getint	Метод для получения значения поля в виде целого числа.
odbcfield.getlong	Метод для получения значения поля в виде длинного целого числа.
odbcfield.getname	Получить имя поля.
odbcfield.getnumeric	Метод для получения значения поля в виде числа с фиксированной точкой.
odbcfield.getstr	Метод для получения значения поля в виде строки.
odbcfield.gettype	Метод для получения типа значения поля.
odbcfield.isnull	Определить содержит ли поле значение NULL.

Описание ODBC

Краткое описание ODBC библиотеки. Объект типа **odbc** обеспечивает соединение с базой данных. Объекты типа **odbcquery** обеспечивают выполнение SQL запросов, и перемещение курсора по набору данных. Данный объект имеет массив **arr fields[] of odbcfield**, содержащий поля набора данных **odbcfield**, количество элементов этого массива равно количеству полей.

Объекты типа **odbcfield**, позволяют получить информацию о поле и его значение для текущего курсора в наборе данных.

Порядок работы с базой данных:

- создаем соединение ODBC с помощью метода [odbc.connect](#);
- создаем новый запрос ODBC с помощью метода [odbc.new query](#), для одного соединения может быть создано несколько запросов;
- выполняем SQL запрос, с помощью метода [odbcquery.run](#), запрос может возвращать набор данных (команда **SELECT**) или не возвращать данные (команды **INSERT**, **UPDATE** и т.д.);
- можно осуществлять навигацию по таблице методами [odbcquery.first](#), [odbcquery.next](#) и т.д. Доступ к полям осуществляется с помощью массива полей **odbcquery.fields[i]**, где *i* - номер поля с 0, также можно использовать метод `odbcquery.fieldbyname`;
- для получения значений полей необходимо пользоваться методами [odbcfield.getstr](#), [odbcfield.getint](#) и т.д.;
- после обработки можно выполнить следующий SQL запрос;
- разрываем соединение ODBC методом `odbc.disconnect`.

При работе следует учитывать особенности некоторых драйверов ODBC:

при выполнении SQL запроса с большим количеством последовательных команд "INSERT ...", выполняется только часть команд такого запроса (количество команд может колебаться от 300 до 1000 для драйвера "SQL server"), при этом никаких сообщений об ошибке не выдается. Необходимо дробить запросы такого вида на несколько частей; некоторые драйвера не позволяют вычислить общее количество записей полученных SQL запросом.

Смотрите также

- [ODBC \(SQL\)](#)

odbc.connect

- [method uint odbc.connect\(str connectstr \)](#)
- [method uint odbc.connect\(str dsn, str user, str psw \)](#)

Соединение с базой данных. К базам данных можно подсоединится двумя способами - через строку соединеия и через имя DSN.

Метод позволяет подсоединится к базе данных с помощью строки соединения. Используется специальная строка соединения ODBC, в ней указывается драйвер, имя базы и дополнительные параметры. Например строка соединения с SQL сервером может иметь вид "Driver={SQL

Server};Server=MSSQLSERVER;Database=mydatabase;Trusted_Connection=yes;"

```
method uint odbc.connect (
    str connectstr
)
```

Параметры

connectstr Строка соединения.

Возвращаемое значение

Возвращает 1 в случае успешного соединения, иначе 0.

odbc.connect

Метод позволяет подсоединится к базе данных через заранее созданное описание соединения (DSN имя).

```
method uint odbc.connect (
    str dsn,
    str user,
    str psw
)
```

Параметры

dsn Имя предварительного описанного соединения - DSN.

user Имя пользователя.

psw Пароль пользователя.

Возвращаемое значение

Возвращает 1 в случае успешного соединения, иначе 0.

Смотрите также

- [ODBC \(SQL\)](#)

odbc.disconnect

Отсоединение от базы данных. Метод для разрыва соединения с базой данных.

```
method odbc.disconnect()
```

Смотрите также

- [ODBC \(SQL\)](#)

odbc.geterror

Получить описание последней ошибки. Метод для получения описания последней ошибки соединения с базой данных.

```
method uint odbc.geterror (  
    str state,  
    str message  
)
```

Параметры

state В эту строку будет записано текущее состояние.
message В эту строку будет записано сообщение об ошибке.

Возвращаемое значение

Возвращает код последней ошибки.

Смотрите также

- [ODBC \(SQL\)](#)

odbc.newquery

Создать новый запрос ODBC. Метод создает новый запрос ODBC, для данного ODBC соединения. Для одного соединения может быть создано несколько запросов. Запросы создаются внутри ODBC объекта и удаляются при его уничтожении.

```
method odbcquery odbc.newquery ()
```

Возвращаемое значение

Новый запрос ODBC.

Смотрите также

- [ODBC \(SQL\)](#)

odbcquery.active

Проверить наличие результирующего набора после выполнения SQL запроса. Если успешно выполнен SQL запрос "SELECT ...", то данный метод должен вернуть не ноль.

```
method uint odbcquery.active()
```

Возвращаемое значение

Возвращается не ноль если имеется результирующий набор.

Смотрите также

- [ODBC \(SQL\)](#)

odbcquery.close

Завершить работу с набором данных. Метод для завершения работы с набором данных. Применяется для объекта, в котором был выполнен SQL запрос **SELECT**.... Данный метод вызывается автоматически при каждом вызове [odbcquery.run](#).

```
method odbcquery.close ()
```

Смотрите также

- [ODBC \(SQL\)](#)

odbcquery.first

Перейти на первую запись в результирующем наборе.

```
method uint odbcquery.first()
```

Возвращаемое значение

Если перемещение курсора произошло успешно возвращается не ноль.

Смотрите также

- [ODBC \(SQL\)](#)

odbcquery.geterror

Получить описание последней ошибки. Метод для получения описания последней ошибки возникшей при выполнении SQL запроса.

```
method uint odbcquery.geterror (  
    str state,  
    str message  
)
```

Параметры

state В эту строку будет записано текущее состояние.
message В эту строку будет записано сообщение об ошибке.

Возвращаемое значение

Возвращает код последней ошибки.

Смотрите также

- [ODBC \(SQL\)](#)

odbcquery.getrecordcount

Количество записей в результирующем наборе. Метод позволяет получить количество записей в результирующем наборе после выполнения SQL запроса "SELECT ...".

```
method uint odbcquery.getrecordcount()
```

Возвращаемое значение

Количество записей, если невозможно определить количество записей возвращается -1.

Смотрите также

- [ODBC \(SQL\)](#)

odbcquery.last

Перейти на последнюю запись в результирующем наборе.

```
method uint odbcquery.last()
```

Возвращаемое значение

Если перемещение курсора произошло успешно возвращается не ноль.

Смотрите также

- [ODBC \(SQL\)](#)

odbcquery.moveby

Перейти на указанное количество записей в результирующем наборе.

```
method uint odbcquery.moveby (  
    int off  
)
```

Параметры

off Количество записей на которое следует сдвинуть курсор. Если число отрицательное курсор смещается в направлении к началу результирующего набора.

Возвращаемое значение

Если перемещение курсора произошло успешно возвращается не ноль.

Смотрите также

- [ODBC \(SQL\)](#)

odbcquery.next

Перейти на следующую запись в результирующем наборе.

```
method uint odbcquery.next()
```

Возвращаемое значение

Если перемещение курсора произошло успешно возвращается не ноль. В противном случае, в том числе если записей больше нет возвращается ноль.

Смотрите также

- [ODBC \(SQL\)](#)

odbcquery.prior

Перейти на предыдущую запись в результирующем наборе.

```
method uint odbcquery.prior()
```

Возвращаемое значение

Если перемещение курсора произошло успешно, возвращает не ноль.

Смотрите также

- [ODBC \(SQL\)](#)

odbcquery.run

Выполнить SQL запрос.

```
method uint odbcquery.run (  
  str sqlstr  
)
```

Параметры

sqlstr Строка содержащая SQL запрос.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [ODBC \(SQL\)](#)

odbcquery.settimeout

Установить время отводимое на выполнение запроса.

```
method odbcquery.settimeout (  
    uint timeout  
)
```

Параметры

timeout Время в секундах отводимое на выполнение SQL запроса. Если 0 время не ограничивается.

Смотрите также

- [ODBC \(SQL\)](#)

odbcfield.getbuf

Метод для получения значения поля в виде буфера. Метод применим только для полей содержащих двоичные данные.

```
method buf odbcfield.getbuf (  
    buf dest  
)
```

Параметры

dest Объект **buf** для получения данных.

Возвращаемое значение

Возвращается параметр **dest**.

Смотрите также

- [ODBC \(SQL\)](#)

odbcfield.getdatetime

Получить значение поля в виде даты и времени - тип datetime. Метод применим только для полей содержащих дату и/или время.

```
method datetime odbcfield.getdatetime (  
    datetime dt  
)
```

Параметры

dt Объект [datetime](#) для получения даты и времени.

Возвращаемое значение

Возвращается параметр *dt*.

Смотрите также

- [ODBC \(SQL\)](#)

odbcfield.getdouble

Метод для получения значения поля в виде числа с плавающей точкой. Метод применим для полей содержащих числа с плавающей точкой.

```
method double odbcfield.getdouble()
```

Возвращаемое значение

Возвращает значение поля.

Смотрите также

- [ODBC \(SQL\)](#)

odbcfield.getindex

Получить порядковый номер поля.

```
method uint odbcfield.getindex()
```

Возвращаемое значение

Порядковый номер поля.

Смотрите также

- [ODBC \(SQL\)](#)

odbcfield.getint

- [method int odbcfield.getint\(\)](#)
- [method uint odbcfield.getuint\(\)](#)

Метод для получения значения поля в виде целого числа. Метод применим для полей содержащих целые числа, размером до 4-х байт.

```
method int odbcfield.getint()
```

Возвращаемое значение

Возвращает значение поля.

odbcfield.getuint

Метод для получения значения поля в виде беззнакового целого числа. Метод применим для полей содержащих целые числа, размером до 4-х байт.

```
method uint odbcfield.getuint()
```

Возвращаемое значение

Возвращает значение поля.

Смотрите также

- [ODBC \(SQL\)](#)

odbcfield.getlong

- [method long odbcfield.getlong\(\)](#)
- [method ulong odbcfield.getulong\(\)](#)

Метод для получения значения поля в виде длинного целого числа. Метод применим для полей содержащих длинные целые числа, размером 8 байт.

```
method long odbcfield.getlong()
```

Возвращаемое значение

Возвращает значение поля.

odbcfield.getulong

Метод для получения значения поля в виде длинного беззнакового целого числа. Метод применим для полей содержащих длинные целые числа, размером 8 байт.

```
method ulong odbcfield.getulong()
```

Возвращаемое значение

Возвращает значение поля.

Смотрите также

- [ODBC \(SQL\)](#)

odbcfield.getname

Получить имя поля.

```
method str odbcfield.getname (  
    str result  
)
```

Параметры

result Строка для получения результата.

Возвращаемое значение

Возвращается параметр *result*.

Смотрите также

- [ODBC \(SQL\)](#)

odbcfield.getnumeric

Метод для получения значения поля в виде числа с фиксированной точкой. Метод применим для полей содержащих числа с фиксированной точкой. Для работы с такими данными используется структура:

```
type numeric {  
    long val  
    uint scale  
}
```

В поле **val** содержится целое представление числа, а в поле **scale** масштаб - показывает сколько раз надо разделить val на 10, чтобы получить реальное число (количество значащих цифр после запятой).

```
method numeric odbcfield.getnumeric (  
    numeric num  
)
```

Параметры

num Структура для получения числа.

Возвращаемое значение

Возвращается параметр *num*.

Смотрите также

- [ODBC \(SQL\)](#)

odbcfield.getstr

Метод для получения значения поля в виде строки. Метод применим для полей содержащих строку, дату, время и числовых полей.

```
method str odbcfield.getstr (  
    str dest  
)
```

Параметры

dest Объект **str** для получения данных.

Возвращаемое значение

Возвращается параметр **dest**.

Смотрите также

- [ODBC \(SQL\)](#)

odbcfield.gettype

Метод для получения типа значения поля. Возвращает идентификатор одного из следующих типов: **buf**, **str**, **int**, **long**, **numeric**, **double**, **datetime**.

```
method uint odbcfield.gettype ()
```

Возвращаемое значение

Идентификатор полученного типа.

Смотрите также

- [ODBC \(SQL\)](#)

odbcfield.isnull

Определить содержит ли поле значение NULL.

```
method uint odbcfield.isnull()
```

Возвращаемое значение

Возвращает не ноль, если поле содержит значение NULL, иначе 0.

Смотрите также

- [ODBC \(SQL\)](#)

Process

Функции для процесса, запуска по расширению, получению аргументов и переменных окружения.

<code>argc</code>	Получить количество аргументов.
<code>argv</code>	Получить аргумент.
<code>exit</code>	Завершить выполнение текущей программы.
<code>getenv</code>	Получить переменную окружения.
<code>process</code>	Запустить процесс.
<code>setenv</code>	Установить переменную окружения.
<code>shell</code>	Запустить или открыть файл в соответствующем ему приложении.

argc

Получить количество аргументов. Функция возвращает количество аргументов командной строки.

```
func uint argc()
```

Возвращаемое значение

Количество аргументов переданных в командной строке.

Смотрите также

- [Process](#)

argv

Получить аргумент. Функция возвращает аргумент командной строки.

```
func str argv (  
    str ret,  
    uint num  
)
```

Параметры

ret Переменная, в которую будет записано полученное значение.

num Порядковый номер получаемого аргумента с 1.

Возвращаемое значение

Возвращается параметр **ret**.

Смотрите также

- [Process](#)

exit

Завершить выполнение текущей программы.

```
func exit (  
    uint code  
)
```

Параметры

code Код возврата или результат работы программы.

Смотрите также

- [Process](#)

getenv

Получить переменную окружения.

```
func str getenv (  
    str varname,  
    str ret  
)
```

Параметры

varname Имя переменной окружения.
ret Строка для получения результата.

Возвращаемое значение

Возвращается параметр *ret*.

Смотрите также

- [Process](#)

process

Запустить процесс.

```
func uint process (  
    str cmdline,  
    str workdir,  
    uint result  
)
```

Параметры

cmdline Командная строка запуска.

workdir Рабочая директория. Может быть 0->str.

result Указатель на uint для получения результата. Если 0, то функция не будет ждать окончания работы процесса.

Возвращаемое значение

В случае успешного выполнения возвращается 1, в противном случае 0.

Смотрите также

- [Process](#)

setenv

Установить переменную окружения. Функция записывает значение переменной окружения. Новое значение будет действительно только в текущем процессе.

```
func uint setenv (  
    str varname,  
    str varvalue  
)
```

Параметры

varname Имя переменной окружения.
varvalue Новое значение переменной окружения.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Process](#)

Registry

Работа с Реестром. Данная библиотека реализует работу с Реестром Windows. Для использования библиотеки необходимо с помощью команды `include` указать файл `registry.g`, который находится в поддиректории `lib\registry`.

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

- [Функции](#)
- [Методы](#)

Функции

<code>regdelkey</code>	Удалить ключ реестра.
<code>regdelvalue</code>	Удалить значение ключа.
<code>reggetmultistr</code>	Получить последовательность строк.
<code>reggetnum</code>	Получить числовое значение ключа реестра.
<code>regkeys</code>	Получить список подключей данного ключа.
<code>regsetmultistr</code>	Записать последовательность строк.
<code>regsetnum</code>	Записать число как значение ключа реестра.
<code>regvaltype</code>	Имя значения данного ключа у которого определяется тип.
<code>regvalues</code>	Получить список имен значений данного ключа.
<code>regverify</code>	Создать недостающие ключи.

Методы

<code>buf.regget</code>	Получить значение.
<code>buf.regset</code>	Записать значение.
<code>str.regget</code>	Получить значение в виде строки.
<code>str.regset</code>	Записать строку как значение ключа реестра.

regdelkey

Удалить ключ реестра.

```
func uint regdelkey (  
    uint root,  
    str subkey  
)
```

Параметры

root

Корневой ключ.

\$HKEY_CLASSES_ROOT

Classes Root.

\$HKEY_CURRENT_USER

Настройки текущего пользователя.

\$HKEY_LOCAL_MACHINE

Настройки компьютера.

\$HKEY_USERS

Настройки в сех пользователей.

subkey

Имя удаляемого ключа реестра.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Registry](#)

regdelvalue

Удалить значение ключа.

```
func uint regdelvalue (  
    uint root,  
    str subkey,  
    str value  
)
```

Параметры

root

Корневой ключ.

\$HKEY_CLASSES_ROOT

Classes Root.

\$HKEY_CURRENT_USER

Настройки текущего пользователя.

\$HKEY_LOCAL_MACHINE

Настройки компьютера.

\$HKEY_USERS

Настройки всех пользователей.

subkey

Имя ключа реестра.

value

Имя удаляемого значения.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Registry](#)

reggetmultistr

Получить последовательность строк. Получить значение ключа реестра типа \$REG_MULTISZ в массив строк.

```
func arrstr reggetmultistr (  
    uint root,  
    str subkey,  
    str valname,  
    arrstr val  
)
```

Параметры

<i>root</i>	Корневой ключ.
<i>subkey</i>	Имя ключа реестра.
<i>valname</i>	Имя значения указанного ключа.
<i>val</i>	Массив куда запишутся строки.

\$HKEY_CLASSES_ROOT	Classes Root.
\$HKEY_CURRENT_USER	Настройки текущего пользователя.
\$HKEY_LOCAL_MACHINE	Настройки компьютера.
\$HKEY_USERS	Настройки в всех пользователей.

Возвращаемое значение

Возвращается параметр *val*.

Смотрите также

- [Registry](#)

reggetnum

- [func uint reggetnum\(uint root, str subkey, str valname \)](#)
- [func uint reggetnum\(uint root, str subkey, str valname, uint defval \)](#)

Получить числовое значение ключа реестра.

```
func uint reggetnum (  
    uint root,  
    str subkey,  
    str valname  
)
```

Параметры

<i>root</i>	Корневой ключ.
<i>subkey</i>	Имя ключа реестра.
<i>valname</i>	Имя значения указанного ключа.

\$HKEY_CLASSES_ROOT	Classes Root.
\$HKEY_CURRENT_USER	Настройки текущего пользователя.
\$HKEY_LOCAL_MACHINE	Настройки компьютера.
\$HKEY_USERS	Настройки в сех пользователей.

Возвращаемое значение

Возвращается числовое значение.

reggetnum

Получить числовое значение ключа реестра.

```
func uint reggetnum (  
    uint root,  
    str subkey,  
    str valname,  
    uint defval  
)
```

Параметры

<i>root</i>	Корневой ключ.
<i>subkey</i>	Имя ключа реестра.
<i>valname</i>	Имя значения указанного ключа.
<i>defval</i>	Число по умолчанию в случае отсутствия значения.

\$HKEY_CLASSES_ROOT	Classes Root.
\$HKEY_CURRENT_USER	Настройки текущего пользователя.
\$HKEY_LOCAL_MACHINE	Настройки компьютера.
\$HKEY_USERS	Настройки в сех пользователей.

Возвращаемое значение

Возвращается числовое значение.

Смотрите также

- [Registry](#)

regkeys

Получить список подключей данного ключа.

```
func uint regkeys (  
    uint root,  
    str subkey,  
    arrstr ret  
)
```

Параметры

root Корневой ключ.

<code>\$HKEY_CLASSES_ROOT</code>	Classes Root.
<code>\$HKEY_CURRENT_USER</code>	Настройки текущего пользователя.
<code>\$HKEY_LOCAL_MACHINE</code>	Настройки компьютера.
<code>\$HKEY_USERS</code>	Настройки в сех пользователей.

subkey Имя ключа реестра.

ret Массив куда запишутся наименования ключей.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Registry](#)

regsetmultistr

Записать последовательность строк. Записать массив строк как значение ключа реестра типа \$REG_MULTISZ. Если ключ отсутствует, то он будет создан.

```
func uint regsetmultistr (  
    uint root,  
    str subkey,  
    str valname,  
    arrstr val,  
    arrstr ret  
)
```

Параметры

<i>root</i>	Корневой ключ.								
	<table border="1"><tr><td>\$HKEY_CLASSES_ROOT</td><td>Classes Root.</td></tr><tr><td>\$HKEY_CURRENT_USER</td><td>Настройки текущего пользователя.</td></tr><tr><td>\$HKEY_LOCAL_MACHINE</td><td>Настройки компьютера.</td></tr><tr><td>\$HKEY_USERS</td><td>Настройки в сех пользователей.</td></tr></table>	\$HKEY_CLASSES_ROOT	Classes Root.	\$HKEY_CURRENT_USER	Настройки текущего пользователя.	\$HKEY_LOCAL_MACHINE	Настройки компьютера.	\$HKEY_USERS	Настройки в сех пользователей.
\$HKEY_CLASSES_ROOT	Classes Root.								
\$HKEY_CURRENT_USER	Настройки текущего пользователя.								
\$HKEY_LOCAL_MACHINE	Настройки компьютера.								
\$HKEY_USERS	Настройки в сех пользователей.								
<i>subkey</i>	Имя ключа реестра.								
<i>valname</i>	Имя записываемого значения.								
<i>val</i>	Записываемый массив строк.								
<i>ret</i>	Массив строк куда будут записаны все созданные ключи. Может быть 0.								

Возвращаемое значение

0	Данные не записались.
1	Запись произошла с созданием значения ключа.
2	Данные записались в существующее значение.

Смотрите также

- [Registry](#)

regsetnum

Записать число как значение ключа реестра. Если ключ отсутствует, то он будет создан.

```
func uint regsetnum (  
    uint root,  
    str subkey,  
    str valname,  
    uint value,  
    arrstr ret  
)
```

Параметры

<i>root</i>	Корневой ключ.								
	<table border="1"><tr><td>\$HKEY_CLASSES_ROOT</td><td>Classes Root.</td></tr><tr><td>\$HKEY_CURRENT_USER</td><td>Настройки текущего пользователя.</td></tr><tr><td>\$HKEY_LOCAL_MACHINE</td><td>Настройки компьютера.</td></tr><tr><td>\$HKEY_USERS</td><td>Настройки в всех пользователей.</td></tr></table>	\$HKEY_CLASSES_ROOT	Classes Root.	\$HKEY_CURRENT_USER	Настройки текущего пользователя.	\$HKEY_LOCAL_MACHINE	Настройки компьютера.	\$HKEY_USERS	Настройки в всех пользователей.
\$HKEY_CLASSES_ROOT	Classes Root.								
\$HKEY_CURRENT_USER	Настройки текущего пользователя.								
\$HKEY_LOCAL_MACHINE	Настройки компьютера.								
\$HKEY_USERS	Настройки в всех пользователей.								
<i>subkey</i>	Имя ключа реестра.								
<i>valname</i>	Имя записываемого значения.								
<i>value</i>	Записываемое число.								
<i>ret</i>	Массив строк куда будут записаны все созданные ключи. Может быть 0.								

Возвращаемое значение

0	Данные не записались.
1	Запись произошла с созданием значения ключа.
2	Данные записались в существующее значение.

Смотрите также

- [Registry](#)

regvaltype

Имя значения данного ключа у которого определяется тип.

```
func uint regvaltype (  
    uint root,  
    str subkey,  
    str valname  
)
```

Параметры

root

Корневой ключ.

\$HKEY_CLASSES_ROOT

Classes Root.

\$HKEY_CURRENT_USER

Настройки текущего пользователя.

\$HKEY_LOCAL_MACHINE

Настройки компьютера.

\$HKEY_USERS

Настройки в сех пользователей.

subkey

Имя ключа реестра.

valname

Имя значения данного ключа у которого определяется тип.

Возвращаемое значение

Возвращается 0, если тип не определен или данное значение отсутствует. Кроме этого возможны следующие значения:

\$REG_NONE

Неизвестен.

\$REG_SZ

Строка.

\$REG_EXPAND_SZ

Расширенная строка. Строка с переменными окружения.

\$REG_BINARY

Двоичные данные.

\$REG_DWORD

Целое число.

\$REG_MULTI_SZ

Последовательность строк.

Смотрите также

- [Registry](#)

regvalues

Получить список имен значений данного ключа.

```
func uint regvalues (  
    uint root,  
    str subkey,  
    arrstr ret  
)
```

Параметры

root Корневой ключ.

<code>\$HKEY_CLASSES_ROOT</code>	Classes Root.
<code>\$HKEY_CURRENT_USER</code>	Настройки текущего пользователя.
<code>\$HKEY_LOCAL_MACHINE</code>	Настройки компьютера.
<code>\$HKEY_USERS</code>	Настройки в сех пользователей.

subkey Имя ключа реестра.

ret Массив куда запишутся наименования значений ключа.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Registry](#)

regverify

Создать недостающие ключи. Проверить наличие ключа реестра и создать его в случае отсутствия.

```
func uint regverify (  
    uint root,  
    str subkey,  
    arrstr ret  
)
```

Параметры

root

Корневой ключ.

\$HKEY_CLASSES_ROOT

Classes Root.

\$HKEY_CURRENT_USER

Настройки текущего пользователя.

\$HKEY_LOCAL_MACHINE

Настройки компьютера.

\$HKEY_USERS

Настройки всех пользователей.

subkey

Имя проверяемого ключа реестра.

ret

Массив строк куда будут записаны все созданные ключи. Может быть 0.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Registry](#)

buf.regget

Получить значение. Данный метод записывает значение ключа реестра в объект типа [Buffer](#).

```
method buf buf.regget (  
    uint root,  
    str subkey,  
    str valname,  
    uint regtype  
)
```

Параметры

root

Корневой ключ.

\$HKEY_CLASSES_ROOT

Classes Root.

\$HKEY_CURRENT_USER

Настройки текущего пользователя.

\$HKEY_LOCAL_MACHINE

Настройки компьютера.

\$HKEY_USERS

Настройки в всех пользователей.

subkey

Имя ключа реестра.

valname

Имя значения указанного ключа.

regtype

Указатель на uint куда будет записан тип данного значения. Может быть 0.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Registry](#)

buf.regset

Записать значение. Записать данные объекта buf как значение ключа реестра. Если ключ отсутствует, то он будет создан.

```
method uint buf.regset (  
    uint root,  
    str subkey,  
    str valname,  
    uint regtype,  
    arrstr ret  
)
```

Параметры

root Корневой ключ.

\$HKEY_CLASSES_ROOT	Classes Root.
\$HKEY_CURRENT_USER	Настройки текущего пользователя.
\$HKEY_LOCAL_MACHINE	Настройки компьютера.
\$HKEY_USERS	Настройки всех пользователей.

subkey Имя ключа реестра.

valname Имя записываемого значения.

regtype Тип значения.

\$REG_NONE	Неизвестен.
\$REG_SZ	Строка.
\$REG_EXPAND_SZ	Расширенная строка. Строка с переменными окружения.
\$REG_BINARY	Двоичные данные.
\$REG_DWORD	Целое число.
\$REG_MULTI_SZ	Последовательность строк.

ret Массив строк куда будут записаны все созданные ключи. Может быть 0.

Возвращаемое значение

- 0 Данные не записались.
- 1 Запись произошла с созданием значения ключа.
- 2 Данные записались в существующее значение.

Смотрите также

- [Registry](#)

str.regget

- [method str str.regget\(uint root, str subkey, str valname \)](#)
- [method str str.regget\(uint root, str subkey, str valname, str defval \)](#)

Получить значение в виде строки. Данный метод получает значение ключа реестра в виде строки.

```
method str str.regget (  
    uint root,  
    str subkey,  
    str valname  
)
```

Параметры

<i>root</i>	Корневой ключ.
<i>subkey</i>	Имя ключа реестра.
<i>valname</i>	Имя значения указанного ключа.

\$HKEY_CLASSES_ROOT	Classes Root.
\$HKEY_CURRENT_USER	Настройки текущего пользователя.
\$HKEY_LOCAL_MACHINE	Настройки компьютера.
\$HKEY_USERS	Настройки в сех пользователей.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

str.regget

Данный метод получает значение ключа реестра в виде строки.

```
method str str.regget (  
    uint root,  
    str subkey,  
    str valname,  
    str defval  
)
```

Параметры

<i>root</i>	Корневой ключ.
<i>subkey</i>	Имя ключа реестра.
<i>valname</i>	Имя значения указанного ключа.
<i>defval</i>	Строка по умолчанию в случае отсутствия значения.

\$HKEY_CLASSES_ROOT	Classes Root.
\$HKEY_CURRENT_USER	Настройки текущего пользователя.
\$HKEY_LOCAL_MACHINE	Настройки компьютера.
\$HKEY_USERS	Настройки в сех пользователей.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Registry](#)

str.regset

- [method uint str.regset\(uint root, str subkey, str valname, arrstr ret \)](#)
- [method uint str.regset\(uint root, str subkey, str valname \)](#)

Записать строку как значение ключа реестра. Если ключ отсутствует, то он будет создан.

```
method uint str.regset (  
    uint root,  
    str subkey,  
    str valname,  
    arrstr ret  
)
```

Параметры

<i>root</i>	Корневой ключ.								
	<table border="1"><tr><td>\$HKEY_CLASSES_ROOT</td><td>Classes Root.</td></tr><tr><td>\$HKEY_CURRENT_USER</td><td>Настройки текущего пользователя.</td></tr><tr><td>\$HKEY_LOCAL_MACHINE</td><td>Настройки компьютера.</td></tr><tr><td>\$HKEY_USERS</td><td>Настройки всех пользователей.</td></tr></table>	\$HKEY_CLASSES_ROOT	Classes Root.	\$HKEY_CURRENT_USER	Настройки текущего пользователя.	\$HKEY_LOCAL_MACHINE	Настройки компьютера.	\$HKEY_USERS	Настройки всех пользователей.
\$HKEY_CLASSES_ROOT	Classes Root.								
\$HKEY_CURRENT_USER	Настройки текущего пользователя.								
\$HKEY_LOCAL_MACHINE	Настройки компьютера.								
\$HKEY_USERS	Настройки всех пользователей.								
<i>subkey</i>	Имя ключа реестра.								
<i>valname</i>	Имя записываемого значения.								
<i>ret</i>	Массив строк куда будут записаны все созданные ключи. Может быть 0.								

Возвращаемое значение

- 0 Данные не записались.
- 1 Запись произошла с созданием значения ключа.
- 2 Данные записались в существующее значение.

str.regset

Записать строку как значение ключа реестра. Если ключ отсутствует, то он будет создан.

```
method uint str.regset (  
    uint root,  
    str subkey,  
    str valname  
)
```

Параметры

<i>root</i>	Корневой ключ.								
	<table border="1"><tr><td>\$HKEY_CLASSES_ROOT</td><td>Classes Root.</td></tr><tr><td>\$HKEY_CURRENT_USER</td><td>Настройки текущего пользователя.</td></tr><tr><td>\$HKEY_LOCAL_MACHINE</td><td>Настройки компьютера.</td></tr><tr><td>\$HKEY_USERS</td><td>Настройки всех пользователей.</td></tr></table>	\$HKEY_CLASSES_ROOT	Classes Root.	\$HKEY_CURRENT_USER	Настройки текущего пользователя.	\$HKEY_LOCAL_MACHINE	Настройки компьютера.	\$HKEY_USERS	Настройки всех пользователей.
\$HKEY_CLASSES_ROOT	Classes Root.								
\$HKEY_CURRENT_USER	Настройки текущего пользователя.								
\$HKEY_LOCAL_MACHINE	Настройки компьютера.								
\$HKEY_USERS	Настройки всех пользователей.								
<i>subkey</i>	Имя ключа реестра.								
<i>valname</i>	Имя записываемого значения.								

Возвращаемое значение

- 0 Данные не записались.
- 1 Запись произошла с созданием значения ключа.
- 2 Данные записались в существующее значение.

Смотрите также

- [Registry](#)

Socket

Сокеты и общие интернет функции. Вы должны вызвать функцию [inet_init](#) перед использованием библиотеки. Для использования библиотеки необходимо с помощью команды `include` указать файл `internet.g`, который находится в поддиректории `lib/socket`.

```
include : $"...\gentee\lib\socket\internet.g"
```

- [Общие интернет функции](#)
- [Методы сокетов](#)
- [URL строки](#)
- [Типы](#)

Общие интернет функции

inet_close	Закончить работу.
inet_error	Получить код ошибки.
inet_init	Инициализация библиотеки.
inet_proxy	Использовать прокси-сервер.
inet_proxyenable	Включить/отключить прокси-сервер.
inetnotify_func	Функция обработки сообщений.

Методы сокетов

socket.close	Закреть сокет.
socket.connect	Подключить сокет.
socket.isproxy	Подключение через прокси или нет.
socket.recv	Метод получает пакет от подключенного сервера.
socket.send	Метод посылает запрос к подключенному серверу.
socket.urlconnect	Создание и подключение сокета к URL.

URL строки

str.iencoding	Перекодировка строки.
str.ihead	Получить заголовок.
str.ihttpinfo	Обработать заголовок.
str.iurl	Метод служит для разбора URL адреса на составляющие.

Типы

httpinfo	Данные HTTP заголовка.
inetnotify	Тип для обработки сообщений.
socket	Структура socket.

inet_close

Закончить работу. Функция должна вызываться при окончании работы с библиотекой.

```
func uint inet_close()
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Socket](#)

inet_error

Получить код ошибки. Функция возвращает код последней возникшей ошибки. Коды больше 10000 являются кодами ошибок библиотеки **WinSock 2** (wsock2.dll).

```
func uint inet_error()
```

Возвращаемое значение

Код последней ошибки.

<code>\$ERRINET_DLLVERSION</code>	Не поддерживаемая версия wsock2.dll.
<code>\$ERRINET_HTTPDATA</code>	Получены не HTTP данные.
<code>\$ERRINET_USERBREAK</code>	Процесс прерван пользователем.
<code>\$ERRINET_OPENFILE</code>	Невозможно открыть файл.
<code>\$ERRINET_WRITEFILE</code>	Невозможно записать файл.
<code>\$ERRINET_READFILE</code>	Невозможно прочитать файл.
<code>\$ERRFTP_RESPONSE</code>	Неверный ответ сервера.
<code>\$ERRFTP_QUIT</code>	Неверный ответ сервера на QUIT
<code>\$ERRFTP_BADUSER</code>	Неверное имя пользователя.
<code>\$ERRFTP_BADPSW</code>	Неверный пароль.
<code>\$ERRFTP_PORT</code>	Ошибка PORT.

Смотрите также

- [Socket](#)

inet_init

Инициализация библиотеки. Функция должна быть вызвана перед началом работы с библиотекой.

```
func uint inet_init()
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Socket](#)

inet_proxy

Использовать прокси-сервер. Функция позволяет указывать прокси-сервер для подключения к Интернету.

```
func uint inet_proxy (  
    uint flag,  
    str proxyname  
)
```

Параметры

flag

Флаг указывающий для каких протоколов подключать данный прокси-сервер.

\$PROXY_HTTP	Использовать прокси-сервер для протокола HTTP.
\$PROXY_FTP	Использовать прокси-сервер для протокола FTP.
\$PROXY_ALL	Использовать прокси-сервер для всех протоколов.
\$PROXY_EXPLORER	Взять информация о прокси-сервере из настроек Internet Explorer . В этом случае параметр <i>proxyname</i> может быть пустой строкой.

proxyname

Имя прокси-сервера. Оно должно содержать имя хоста и через двоеточие номер порта.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Socket](#)

inet_proxyenable

Включить/отключить прокси-сервер. Функция позволяет включать или отключать прокси-сервер для разных протоколов. Первоначально прокси-сервер должен быть определен с помощью функции [inet_proxy](#).

```
func uint inet_proxyenable (  
    uint flag,  
    uint enable  
)
```

Параметры

flag Флаг указывающий для каких протоколов включить или отключить прокси-сервер.

\$PROXY_HTTP	Использовать прокси-сервер для протокола HTTP.
\$PROXY_FTP	Использовать прокси-сервер для протокола FTP.
\$PROXY_ALL	Использовать прокси-сервер для всех протоколов.
\$PROXY_EXPLORER	Взять информация о прокси-сервере из настроек Internet Explorer . В этом случае параметр <i>enable</i> может быть пустой строкой.

enable Укажите 1 для включения и 0 для отключения прокси-сервера.

le

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Socket](#)

inetnotify_func

Функция обработки сообщений. При вызове некоторых функций вы можете указать функцию для обработки входящих уведомлений. В частности, она позволяет показывать процесс работы пользователю. Эта функция-обработчик должна иметь следующий вид.

```
func uint inetnotify_func (  
    uint code,  
    inetnotify ni  
)
```

Параметры

co Код сообщения.

<i>de</i>	\$NFYINET_ERROR	Возникла ошибка. Код ошибки можно получить с помощью функции inet_error .
	\$NFYINET_CONNECT	Соединение с сервером.
	\$NFYINET_SEND	Посылаем запрос.
	\$NFYINET_POST	Отправляем данные.
	\$NFYINET_HEAD	Обработали заголовок. <i>ni.param</i> указывает на httpinfo .
	\$NFYINET_REDIRECT	Перенаправление вызова. <i>ni.sparam</i> содержит новый URL.
	\$NFYINET_GET	Получили данные. <i>ni.param</i> содержит суммарный размер данных.
	\$NFYINET_PUT	Отправили данные. <i>ni.param</i> содержит суммарный размер данных.
	\$NFYINET_END	Конец соединения.
	\$NFYFTP_RESPONSE	Ответ FTP сервера в поле <i>ni.head</i> .
	\$NFYFTP_SENDCMD	Посылаемая команда FTP серверу в поле <i>ni.head</i> .
	\$NFYFTP_NOTPASV	Связь с FTP сервером не может быть в пассивном режиме.

ni Переменная типа [inetnotify](#) с дополнительными данными.

Возвращаемое значение

Функция должна возвращать 1 для продолжения работы и 0 в противном случае.

Смотрите также

- [Socket](#)

socket.close

Закрывает сокет.

```
method uint socket.close()
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Socket](#)

socket.connect

Подключить сокет. Метод создает сокет и устанавливает связь с хостом и портом указанными в полях **host** и **port** структуры [socket](#).

```
method uint socket.connect()
```

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Socket](#)

socket.isproxy

Подключение через прокси или нет. С помощью этого метода можно определить подключен ли сокет через прокси-сервер или нет.

```
method uint socket.isproxy()
```

Возвращаемое значение

Возвращается 1 если сокет подключен через прокси-сервер и 0 в противном случае.

Смотрите также

- [Socket](#)

socket.recv

Метод получает пакет от подключенного сервера.

```
method uint socket.recv (  
    buf data  
)
```

Параметры

data Буфер для записи данных. Полученный пакет будет добавлен к уже существующим данным в буфере.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Socket](#)

socket.send

- [method uint socket.send\(str data \)](#)
- [method uint socket.send\(buf data \)](#)

Метод посылает запрос к подключенному серверу.

```
method uint socket.send (  
    str data  
)
```

Параметры

data Строка запроса.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

socket.send

Метод посылает запрос к подключенному серверу.

```
method uint socket.send (  
    buf data  
)
```

Параметры

data Буфер запроса.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Socket](#)

socket.urlconnect

Создание и подключение сокета к URL. Метод служит для создания и подключения сокета к указанному интернет адресу. Если включен прокси-сервер, то подключение будет происходить через него.

```
method uint socket.urlconnect (  
    str url,  
    str host,  
    str path  
)
```

Параметры

url URL адрес для подключения.
host Строка для получения хоста из URL.
path Строка для получения относительного пути из URL.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Socket](#)

str.iencoding

Перекодировка строки. Метод перекодирует указанную строку для отправки с помощью метода POST. Пробелы заменяются '+', служебные символы заменяются на шестнадцатеричное представление **%XX**. Результат запишется в строку для которой вызывается метод.

```
method str str.iencoding (  
    str src  
)
```

Параметры

src Строка для перекодировки.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Socket](#)

str.ihead

Получить заголовок. Метод служит для получения заголовка сообщения. Он запишется в строку для которой вызывается метод. Кроме этого, заголовок будет удален из объекта `data`.

```
method str str.ihead (  
    buf data  
)
```

Параметры

`data` Буфер или строка содержащие обрабатываемые данные.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Socket](#)

str.ihhttpinfo

Обработать заголовок. Метод обрабатывает строку как заголовок HTTP сообщения и записывает полученные данные в структуру [httpinfo](#).

```
method uint str.ihhttpinfo (  
    httpinfo hi  
)
```

Параметры

hi Переменная типа [httpinfo](#) для получения результатов.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Socket](#)

str.iurl

Метод служит для разбора URL адреса на составляющие.

```
method uint str.iurl (  
    str host,  
    str port,  
    str path  
)
```

Параметры

host Строка для получения имени хоста.
port Строка для получения порта.
path Строка для получения относительного пути.

Возвращаемое значение

Возвращается 1 если был указан протокол FTP. В противном случае возвращается 0.

Смотрите также

- [Socket](#)

httpinfo

Данные HTTP заголовка. Структура служит для получения данных из HTTP заголовка. В зависимости от заголовка некоторые поля могут быть пустыми.

```
type httpinfo
{
    uint code
    datetime dt
    str size
    str location
}
```

Поля типа

<i>code</i>	Код сообщения.
<i>dt</i>	Время последней модификации файла.
<i>size</i>	Размер файла.
<i>location</i>	Новое местоположение файла.

Смотрите также

- [Socket](#)

inetnotify

Тип для обработки сообщений. Данная структура передается [функции обработки сообщений](#) в качестве параметра. Дополнительные параметры принимают разные значения в зависимости от кода сообщения.

```
type inetnotify
{
    str url
    str head
    uint param
    str sparam
}
```

Поля типа

<i>url</i>	Обрабатываемый URL адрес.
<i>head</i>	Заголовок полученного пакета.
<i>param</i>	Дополнительный числовой параметр.
<i>sparam</i>	Дополнительный строковый параметр.

Смотрите также

- [Socket](#)

socket

Структура socket.

```
type socket
{
    str host
    ushort port
    uint socket
    uint flag
}
```

Поля типа

host Имя хоста.
port Номер порта.
socket Идентификатор открытого сокета.
flag Дополнительные флаги. **\$SOCKF_PROXY** - Сокет открыт через прокси-сервер.

Смотрите также

- [Socket](#)

Stack

Стек. Вы можете использовать переменные типа **stack** для работы со стеком. Тип **stack** наследуется от типа **arr**, поэтому вы можете также использовать [методы типа arr](#).

- [Методы](#)
- [Типы](#)

Методы

stack.pop	Извлечь верхний элемент.
stack.popval	Извлечь число.
stack.push	Добавить элемент в стек.
stack.top	Получить верхний элемент в стеке.

Типы

stack	Структура типа stack.
-----------------------	-----------------------

stack.pop

- [method uint stack.pop](#)
- [method str stack.pop\(str val \)](#)

Извлечь верхний элемент. Метод удаляет верхний элемент из стека.

```
method uint stack.pop
```

Возвращаемое значение

Указатель на новый верхний элемент.

stack.pop

Метод извлекает строку из стека. Стек должен быть описан как **stack of str**.

```
method str stack.pop (  
  str val  
)
```

Параметры

val Строка для получения результата.

Возвращаемое значение

Возвращается параметр *val*.

Смотрите также

- [Stack](#)

stack.popval

Извлечь число. Метод извлекает число из стека.

```
method uint stack.popval
```

Возвращаемое значение

Возвращается извлеченное из стека число.

Смотрите также

- [Stack](#)

stack.top

Получить верхний элемент в стэке.

```
method uint stack.top
```

Возвращаемое значение

Указатель на верхний элемент.

Смотрите также

- [Stack](#)

stack

Структура типа stack.

```
type stack <inherit = arr>  
{  
}
```

Смотрите также

- [Stack](#)

String

Строки. Вы можете использовать переменные типа `str` для работы со строками. Тип `str` наследуется от типа `buf`. Поэтому вы можете также использовать [методы типа buf](#).

- [Операторы](#)
- [Методы](#)
- [Поиск в строке](#)

Операторы

<code>* str</code>	Получить длины строки.
<code>str + str</code>	Сложение двух строк с созданием результирующей строки.
<code>str = str</code>	Скопировать строку.
<code>str += type</code>	Добавить значение к строке.
<code>str == str</code>	Сравнение на равенство.
<code>str < str</code>	Проверка на меньше.
<code>str > str</code>	Проверка на больше.
<code>str(type)</code>	Конвертирование типов в <code>str</code> .
<code>type(str)</code>	Приведение строки к другим типам.

Методы

<code>str.append</code>	Добавить данные.
<code>str.appendch</code>	Добавить символ.
<code>str.clear</code>	Очистка строки.
<code>str.copy...</code>	Копирование.
<code>str.crc</code>	Вычисление контрольной суммы.
<code>str.del</code>	Удалить подстроку.
<code>str.dellast</code>	Удаление последнего символа.
<code>str.eqlen...</code>	Сравнение.
<code>str.fill...</code>	Дополнить строку.
<code>str.find...</code>	Найти символ в строке.
<code>str.hex...</code>	Конвертирование беззнакового целого в шестнадцатеричный вид.
<code>str.insert</code>	Вставка.
<code>str.islast</code>	Проверить последний символ.
<code>str.lines</code>	Конвертировать многостроковый текст в массив строк.
<code>str.lower</code>	Перевод в нижний регистр.
<code>str.out4</code>	Вывод 32-bit значения.
<code>str.print</code>	Вывод строки на консоль.
<code>str.printf</code>	Записать отформатированные данные в строку.
<code>str.read</code>	Чтение строки из файла.
<code>str.repeat</code>	Повтор строки.
<code>str.replace</code>	Замена в строке.

<code>str.replacech</code>	Замена символа.
<code>str.setlen</code>	Установить новый размер строки.
<code>str.split</code>	Разделение строки.
<code>str.substr</code>	Получить подстроку.
<code>str.trim ...</code>	Удалить крайние символы.
<code>str.upper</code>	Перевод в верхний регистр.
<code>str.write</code>	Запись строки в файл.
<code>str.writeappend</code>	Добавление строки к файлу.

Поиск в строке

<code>spattern</code>	Структура шаблона для поиска.
<code>spattern.init</code>	Создание шаблона поиска.
<code>spattern.search</code>	Поиск строки в строке.
<code>str.search</code>	Поиск подстроки.

* str

Получить длины строки.

```
operator uint * (  
    str left  
)
```

Возвращаемое значение

Длина строки.

Смотрите также

- [String](#)

str + str

Сложение двух строк с созданием результирующей строки.

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

Возвращаемое значение

Новая результирующая строка.

Смотрите также

- [String](#)

str = str

Скопировать строку.

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

Возвращаемое значение

Результирующая строка.

Смотрите также

- [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 +=\(str left, double val \)](#)

Добавить значение к строке. Добавить **str** к **str** => **str += str**.

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

Возвращаемое значение

Результирующая строка.

str += uint

Добавить **uint** к **str** => **str += uint**.

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

str += int

Добавить **int** к **str** => **str += int**.

```
operator str += (  
    str left,  
    int val  
)
```

str += float

Добавить **float** к **str** => **str += float**.

```
operator str += (  
    str left,  
    float val  
)
```

str += long

Добавить **long** к **str** => **str += long**.

```
operator str += (  
    str left,  
    long val  
)
```

str += ulong

Добавить **ulong** к **str** => **str += ulong**.

```
operator str += (  
    str left,  
    ulong val  
)
```

str += double

Добавить **double** к **str** => **str += double**.

```
operator str += (  
    str left,  
    double val  
)
```

Смотрите также

- [String](#)

str == str

- [operator uint ==\(str left, str right \)](#)
- [operator uint !=\(str left, str right \)](#)
- [operator uint %==\(str left, str right \)](#)
- [operator uint %!=\(str left, str right \)](#)

Сравнение на равенство.

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

Возвращаемое значение

Возвращает **1** если строки равны и **0** в противном случае.

str != str

Проверка на неравенство.

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

Возвращаемое значение

Возвращает **1** если строки не равны и **0** в противном случае.

str %== str

Сравнение без учета регистра.

```
operator uint %== (  
    str left,  
    str right  
)
```

Возвращаемое значение

Возвращает **1** если строки равны без учета регистра и **0** в противном случае.

str %!= str

Сравнение на неравенство без учета регистра.

```
operator uint %!= (  
    str left,  
    str right  
)
```

Возвращаемое значение

Возвращает **1** если строки не равны без учета регистра и **0** в противном случае.

Смотрите также

- [String](#)

str < str

- [operator uint <\(str left, str right \)](#)
- [operator uint <=\(str left, str right \)](#)
- [operator uint %<\(str left, str right \)](#)
- [operator uint %<=\(str left, str right \)](#)

Проверка на меньше.

```
operator uint < (  
    str left,  
    str right  
)
```

Возвращаемое значение

Возвращает **1** если первая строка меньше второй и **0** в противном случае.

str <= str

Проверка на меньше или равно.

```
operator uint <= (  
    str left,  
    str right  
)
```

Возвращаемое значение

Возвращает **1** если первая строка меньше или равна второй и **0** в противном случае.

str %< str

Проверка на меньше без учета регистра.

```
operator uint %< (  
    str left,  
    str right  
)
```

Возвращаемое значение

Возвращает **1** если первая строка меньше второй без учета регистра и **0** в противном случае.

str %<= str

Проверка на меньше или равно без учета регистра.

```
operator uint %<= (  
    str left,  
    str right  
)
```

Возвращаемое значение

Возвращает **1** если первая строка меньше или равна второй без учета регистра и **0** в противном случае.

Смотрите также

- [String](#)

str > str

- [operator uint >\(str left, str right \)](#)
- [operator uint >=\(str left, str right \)](#)
- [operator uint %>\(str left, str right \)](#)
- [operator uint %>=\(str left, str right \)](#)

Проверка на больше.

```
operator uint > (  
    str left,  
    str right  
)
```

Возвращаемое значение

Возвращает **1** если первая строка больше второй и **0** в противном случае.

str >= str

Проверка на больше или равно.

```
operator uint >= (  
    str left,  
    str right  
)
```

Возвращаемое значение

Возвращает **1** если первая строка больше или равна второй и **0** в противном случае.

str %> str

Проверка на больше без учета регистра.

```
operator uint %> (  
    str left,  
    str right  
)
```

Возвращаемое значение

Возвращает **1** если первая строка больше второй без учета регистра и **0** в противном случае.

str %>= str

Проверка на больше или равно без учета регистра.

```
operator uint %>= (  
    str left,  
    str right  
)
```

Возвращаемое значение

Возвращает **1** если первая строка больше или равна второй без учета регистра и **0** в противном случае.

Смотрите также

- [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>](#)

Конвертирование типов в str. Конвертировать int в str => str(int).

```
method str int.str < result >
```

Возвращаемое значение

Результирующая строка.

str(uint)

Конвертировать uint в str => str(uint).

```
method str uint.str < result >
```

str(float)

Конвертировать float в str => str(float).

```
method str float.str <result>
```

str(long)

Конвертировать long в str => str(long).

```
method str long.str <result>
```

str(ulong)

Конвертировать ulong в str => str(ulong).

```
method str ulong.str<result>
```

str(double)

Конвертировать double в str => str(double).

```
method str double.str <result>
```

Смотрите также

- [String](#)

type(str)

- [method int str.int](#)
- [method uint str.uint](#)
- [method float str.float](#)
- [method long str.long](#)
- [method double str.double](#)

Приведение строки к другим типам. Конвертировать **str** в **int => int(str)**.

```
method int str.int
```

Возвращаемое значение

Результирующее значение соответствующего типа.

uint(str)

Конвертировать **str** в **uint => uint(str)**.

```
method uint str.uint
```

float(str)

Конвертировать **str** в **float => float(str)**.

```
method float str.float
```

long(str)

Конвертировать **str** в **long => long(str)**.

```
method long str.long
```

double(str)

Конвертировать **str** в **double => double(str)**.

```
method double str.double
```

Смотрите также

- [String](#)

str.append

Добавить данные. Добавить данные к строке.

```
method str str.append (  
    uint src,  
    uint size  
)
```

Параметры

src Указатель на добавляемые данные.
size Размер добавляемых данных.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String](#)

str.appendch

Добавить символ.

```
method str str.appendch (  
    uint ch  
)
```

Параметры

ch Добавляемый символ.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String](#)

str.clear

Очистка строки.

```
method str str.clear()
```

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String](#)

str.copy...

- [method str str.copy\(uint ptr \)](#)
- [method str str.load\(uint ptr, uint len \)](#)

Копирование. Метод копирует данные в строку.

```
method str str.copy (
    uint ptr
)
```

Параметры

ptr Указатель на копируемые данные. Скопируются все данные до нулевого символа.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

str.load

Метод копирует данные в строку.

```
method str str.load (
    uint ptr,
    uint len
)
```

Параметры

src Указатель на копируемые данные. Если данные не заканчиваются нулем, то он добавится автоматически.

size Размер копируемых данных.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String](#)

str.crc

Вычисление контрольной суммы. Метод вычисляет контрольную сумму строки.

```
method uint str.crc()
```

Возвращаемое значение

Возвращается контрольная сумма строки.

Смотрите также

- [String](#)

str.del

Удалить подстроку.

```
method str str.del (  
  uint off,  
  uint len  
)
```

Параметры

off Смещение удаляемой подстроки.

len Размер удаляемой подстроки.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String](#)

str.dellast

Удаление последнего символа. Метод удаляет последний символ если он равен указанному параметру.

```
method str str.dellast (  
    uint ch  
)
```

Параметры

ch Проверяемый символ.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [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\(str src \)](#)

Сравнение. Сравнение строки с указанными данными. Сравнение происходит только по длине строки для которой вызывается метод.

```
method uint str.eqlen (
    uint ptr
)
```

Параметры

ptr Указатель на сравниваемые данные.

Возвращаемое значение

Возвращает 1 если есть равенство и 0 в противном случае.

str.eqlenign

Сравнение строки с указанными данными или строкой. Сравнение происходит только по длине строки для которой вызывается метод.

```
method uint str.eqlenign (
    uint ptr
)
```

Параметры

ptr Указатель на сравниваемые данные. Сравнение идет без учета регистра.

str.eqlen

Сравнение строки с указанной строкой. Сравнение происходит только по длине строки для которой вызывается метод.

```
method uint str.eqlen (
    str src
)
```

Параметры

src Указатель на сравниваемую строку.

str.eqlenign

Сравнение строки с указанной строкой. Сравнение происходит только по длине строки для которой вызывается метод.

```
method uint str.eqlenign (
    str src
)
```

Параметры

src Указатель на сравниваемую строку. Сравнение идет без учета регистра.

Смотрите также

- [String](#)

str.fill...

- [method str str.fill\(str val, uint count, uint flag \)](#)
- [method str str.fillspacel\(uint len \)](#)
- [method str str.fillspacer\(uint len \)](#)

Дополнить строку. Дополнение строки справа или слева.

```
method str str.fill (
  str val,
  uint count,
  uint flag
)
```

Параметры

val Строка которой будет происходить дополнение.
cou Количество добавлений.
nt
fla Флаги.
g

\$FILL_LEFT Заполнять с левой стороны.

\$FILL_LEN Параметр count указывает на конечный размер.

\$FILL_CUT Обрезать, если строка длиннее конечного размера. Использовать вместе с with FILL_LEN.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

str.fillspacel

Дополнить строку пробелами слева.

```
method str str.fillspacel (
  uint len
)
```

Параметры

len Конечный размер строки.

str.fillspacer

Дополнить строку пробелами справа.

```
method str str.fillspacer (
  uint len
)
```

Параметры

len Конечный размер строки.

Смотрите также

- [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\(uint symbol, uint i \)](#)

Найти символ в строке.

```
method uint str.findch (
    uint offset,
    uint symbol,
    uint fromend
)
```

Параметры

offset Смещение для начала поиска.
symbol Символ поиска.
fromend Если равен 1, то поиск будет идти с конца строки.

Возвращаемое значение

Смещение символа если он найден. Если символ не найден, то возвращается длина строки.

str.findch

Найти символ от начала строки.

```
method uint str.findch (
    uint symbol
)
```

Параметры

symbol Символ поиска.

str.findchr

Поиск символа от конца строки.

```
method uint str.findchr (
    uint symbol
)
```

Параметры

symbol Символ поиска.

str.findchfrom

Поиск символа от указанного смещения в строке.

```
method uint str.findchfrom (
    uint symbol,
    uint offset
)
```

Параметры

symbol Символ поиска.
offset Смещение с которого начинать поиск.

str.findchnum

Найти i-й символ в строке.

```
method uint str.findchnum (
    uint symbol,
    uint i
)
```

Параметры

symbol Символ поиска.
i Порядковый номер символа с 1.

Смотрите также

- [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>\(uint val \)](#)

Конвертирование беззнакового целого в шестнадцатеричный вид. Перевод в нижний регистр.

```
method str str.hexl (
    uint val
)
```

Параметры

val Беззнаковое целое для конвертации в строку.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

str.hexu

Конвертирование беззнакового целого в шестнадцатеричный вид. Перевод в верхний регистр.

```
method str str.hexu (
    uint val
)
```

Параметры

val Беззнаковое целое для конвертации в строку.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

hex2strl

Конвертирование беззнакового целого в шестнадцатеричный вид. Перевод в нижний регистр.

```
func str hex2strl<result> (
    uint val
)
```

Параметры

val Беззнаковое целое для конвертации в строку.

Возвращаемое значение

Новая результирующая строка.

hex2stru

Конвертирование беззнакового целого в шестнадцатеричный вид. Перевод в верхний регистр.

```
func str hex2stru<result> (
    uint val
)
```

Параметры

val Беззнаковое целое для конвертации в строку.

Возвращаемое значение

Новая результирующая строка.

Смотрите также

- [String](#)

str.insert

Вставка. Метод вставляет одну строку в другую.

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

Параметры

offset Смещение куда будет вставляться строка.
value Вставляемая строка.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String](#)

str.islast

Проверить последний символ.

```
method uint str.islast (  
    uint symbol  
)
```

Параметры

symbol Проверяемый символ.

Возвращаемое значение

Возвращает 1 если последний символ в строке совпадает с указанным и 0 в противном случае.

Смотрите также

- [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>\(uint trim \)](#)

Конвертировать многостроковый текст в массив строк.

```
method arrstr str.lines (  
  arrstr ret,  
  uint trim,  
  arr offset  
)
```

Параметры

ret Результирующий массив строк.
trim Укажите 1 если вы хотите удалить крайние символы меньше или равные пробелу.
offset Массив для получения смещений подстрок. Может быть 0->>arr.

Возвращаемое значение

Результирующий массив строк.

str.lines

Конвертировать многостроковый текст в массив строк.

```
method arrstr str.lines (  
  arrstr ret,  
  uint trim  
)
```

Параметры

ret Результирующий массив строк.
trim Укажите 1 если вы хотите удалить крайние символы меньше или равные пробелу.

str.lines

Конвертировать многостроковый текст в массив строк.

```
method arrstr str.lines<result> (  
  uint trim  
)
```

Параметры

trim Укажите 1 если вы хотите удалить крайние символы меньше или равные пробелу.

Возвращаемое значение

Новый результирующий массив строк.

Смотрите также

- [String](#)

str.out4

- [method str str.out4\(str format, uint val \)](#)
- [method str str.out8\(str format, ulong val \)](#)

Вывод 32-bit значения. Значение будет сконвертировано и добавлено в конец строки.

```
method str str.out4 (
    str format,
    uint val
)
```

Параметры

format Формат конвертации. Совпадает с форматом в функции 'printf' в языке программирования C.

val 32-bit значение которое будет добавлено.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

str.out8

Вывод 64-bit значения. Значение будет сконвертировано и добавлено в конец строки.

```
method str str.out8 (
    str format,
    ulong val
)
```

Параметры

format Формат конвертации. Совпадает с форматом в функции 'printf' в языке программирования C.

val 64-bit значение которое будет добавлено.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String](#)

str.print

- [method str.print\(\)](#)
- [func print\(str output \)](#)

Вывод строки на консоль.

```
method str.print()
```

print

Функция выводит строку на консоль.

```
func print (  
    str output  
)
```

Параметры

output

Строка вывода.

Смотрите также

- [String](#)

str.printf

Записать отформатированные данные в строку. Метод форматирует и записывает значения в строку. Каждый аргумент конвертируется в соответствии со спецификацией C/C++ функции printf.

```
method str str.printf (  
  str format,  
  collection clt  
)
```

Параметры

<i>format</i>	Формат вывода.
<i>clt</i>	Передаваемые аргументы.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String](#)

str.read

Чтение строки из файла.

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

Параметры

filename

Имя файла.

Возвращаемое значение

Размер прочитанных данных.

Смотрите также

- [String](#)

str.repeat

Повтор строки. Повтор строки указанное количество раз.

```
method str str.repeat (  
  uint count  
)
```

Параметры

count Количество повторений. Результат запишется в эту самую строку.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [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\(str sold, str snew, uint flags \)](#)

Замена в строке. Метод заменяет данные в строке.

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

Параметры

offset Смещение заменяемых данных.
size Размер заменяемых данных.
value Вставляемая строка.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

str.replace

Этот метод ищет в строке строки из одного массива строк и заменяет их строками из другого.

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

Параметры

aold Массив строк которые ищутся.
anew Новые строки для замены.
flag Флаги.

<i>s</i>	\$QS_IGNORECASE	Игнорировать регистр при поиске.
	\$QS_WORD	Искать только целые слова.
	\$QS_BEGINWORD	Искать слова которые начинаются с указанного шаблона.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

str.replace

Метод заменяет в строке одну подстроку на другую.

```
method str str.replace (
    str sold,
    str snew,
    uint flags
)
```

Параметры

sold Строка, которая должна быть заменена.
snew Строка на которую заменяем.
flag Флаги.

<i>s</i>	\$QS_IGNORECASE	Игнорировать регистр при поиске.
	\$QS_WORD	Искать только целые слова.
	\$QS_BEGINWORD	Искать слова которые начинаются с указанного шаблона.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String](#)

str.replacech

Замена символа. Копирование исходной строки с заменой указанного символа на строку.

```
method str str.replacech (  
  str src,  
  uint from,  
  str to  
)
```

Параметры

src Исходная строка.
from Заменяемый символ.
to Строка на которую будет заменяться символ.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String](#)

str.setlen

- [method str str.setlen\(uint len \)](#)
- [method str str.setlenptr\(\)](#)

Установить новый размер строки. Метод не занимается резервированием места. Вы не можете устанавливать размер строки больше чем у вас зарезервировано места. В основном, эта функция используется для установки размера строки после записи в нее данных внешними функциями.

```
method str str.setlen (  
    uint len  
)
```

Параметры

len Новый размер строки.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

str.setlenptr

Пересчитать размер строки до нулевого символа. Функция может использоваться для установки размера строки после записи в нее данных другими функциями.

```
method str str.setlenptr()
```

Смотрите также

- [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\(uint symbol, str left, str right \)](#)

Разделение строки. Метод разбивает строку на подстроки с учетом указанного разделителя.

```
method arrstr str.split (  
    arrstr ret,  
    uint symbol,  
    uint flag  
)
```

Параметры

ret Массив строк куда будут записаны подстроки.
symb Символ-разделитель.
ol
flag Флаги.

\$\$SPLIT_EMPTY	Учитывать пустые строки.
\$\$SPLIT_NOSYS	Удалять символы меньше или равные пробелу справа и слева.
\$\$SPLIT_FIRST	Разделять только до первого разделителя.
\$\$SPLIT_QUOTE	Учитывать, что элементы могут быть заключены в одинарные или двойные кавычки.
\$\$SPLIT_APPEND	Добавлять строки. В противном случае, массив будет очищен перед добавлением.

Возвращаемое значение

Результирующий массив строк.

Метод разделяет строку и создает новый результирующий массив строк.

```
method arrstr str.split <result> (  
    uint symbol,  
    uint flag  
)
```

Параметры

symb Символ-разделитель.
ol
flag Флаги.

\$\$SPLIT_EMPTY	Учитывать пустые строки.
\$\$SPLIT_NOSYS	Удалять символы меньше или равные пробелу справа и слева.
\$\$SPLIT_FIRST	Разделять только до первого разделителя.
\$\$SPLIT_QUOTE	Учитывать, что элементы могут быть заключены в одинарные или двойные кавычки.
\$\$SPLIT_APPEND	Добавлять строки. В противном случае, массив будет очищен перед добавлением.

Возвращаемое значение

Новый результирующий массив строк.

str.split

Метод ищет первое вхождение символа *symbol* и делит строку на две части.

```
method uint str.split (  
    uint symbol,  
    str left,  
    str right  
)
```

Параметры

symbol Символ-разделитель.
left Левая подстрока до *symbol*.

right

Правая подстрока после *symbol*.

Возвращаемое значение

Возвращает 1, если разделитель был найден и 0 в противном случае.

Смотрите также

- [String](#)

str.substr

- [method str str.substr\(str src, uint off, uint len \)](#)
- [method str str.substr\(uint off, uint len \)](#)

Получить подстроку.

```
method str str.substr (
    str src,
    uint off,
    uint len
)
```

Параметры

<i>src</i>	Исходная строка.
<i>off</i>	Смещение подстроки.
<i>len</i>	Размер подстроки.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

str.substr

Получить подстроку. Подстрока будет записана поверх текущей строки.

```
method str str.substr (
    uint off,
    uint len
)
```

Параметры

<i>off</i>	Смещение подстроки.
<i>len</i>	Размер подстроки.

Смотрите также

- [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\(\)](#)

Удалить крайние символы. Удалить крайние пробелы и символы, которые меньше пробела.

```
method str str.trimsys()
```

Возвращаемое значение

Возвращается объект для которого был вызван метод.

str.trimrsys

Удалить справа пробелы и символы, которые меньше пробела.

```
method str str.trimrsys()
```

str.trim

Удалить указанный символ с любого края строки.

```
method str str.trim (
    uint symbol,
    uint flag
)
```

Параметры

symbol Удаляемый символ.

ol

flag Флаги.

\$TRIM_LEFT	Удалять с левой стороны.
\$TRIM_RIGHT	Удалять с правой стороны.
\$TRIM_ONE	Удалять только один символ.
\$TRIM_PAIR	Если удаляемый символ скобка, то смотреть на закрывающую скобку с правой стороны.
\$TRIM_SYS	Удалять символы меньше или равные пробелу.

str.trimrspace

Удалить правые пробелы.

```
method str str.trimrspace()
```

str.trimspace

Удалить крайние пробелы слева и справа.

```
method str str.trimspace()
```

Смотрите также

- [String](#)

str.write

Запись строки в файл.

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

Параметры

filename Имя файла для записи. Если файл существует, то он будет перезаписан.

Возвращаемое значение

Размер записанных данных.

Смотрите также

- [String](#)

spattern

Структура шаблона для поиска. Тип `spattern` используется для поиска в другой строке или буфере. Не изменяйте поля структуры. Переменная типа `spattern` должна быть инициализирована с помощью метода [spattern.init](#).

```
type spattern
{
    uint pattern
    uint size
    reserved shift[1024]
    uint flag
}
```

Поля типа

<code>pattern</code>	Внутренние данные.
<code>size</code>	Размер шаблона.
<code>shift[1024]</code>	Внутренние данные.
<code>flag</code>	Флаги поиска.

Смотрите также

- [String](#)

spattern.init

- [method spattern spattern.init\(buf pattern, uint flag \)](#)
- [method spattern spattern.init\(str pattern, uint flag \)](#)

Создание шаблона поиска. Перед началом поиска надо вызвать метод `spattern.init`, который позволит инициализировать строку поиска. Затем можно осуществлять поиск данного шаблона с помощью [spattern.search](#).

```
method spattern spattern.init (
    buf pattern,
    uint flag
)
```

Параметры

pattern Строка (шаблон) поиска.
flag Флаги поиска.

\$QS_IGNORECASE	Игнорировать регистр при поиске.
\$QS_WORD	Искать только целые слова.
\$QS_BEGINWORD	Искать слова которые начинаются с указанного шаблона.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

spattern.init

Создание шаблона поиска.

```
method spattern spattern.init (
    str pattern,
    uint flag
)
```

Параметры

pattern Строка (шаблон) поиска.
flag Флаги поиска.

\$QS_IGNORECASE	Игнорировать регистр при поиске.
\$QS_WORD	Искать только целые слова.
\$QS_BEGINWORD	Искать слова которые начинаются с указанного шаблона.

Смотрите также

- [String](#)

spattern.search

- [method uint spattern.search\(buf src, uint offset \)](#)
- [method uint spattern.search\(uint ptr, uint size \)](#)

Поиск строки в строке. Перед началом поиска надо вызвать метод [spattern.init](#), который позволит инициализировать шаблон поиска.

```
method uint spattern.search (
    buf src,
    uint offset
)
```

Параметры

src Строка в которой ищется указанная строка (шаблон поиска).
offset Смещение с которого продолжить поиск.

Возвращаемое значение

Смещение найденного фрагмента. Если равно размеру строки, то фрагмент не найден.

spattern.search

Поиск строки в строке.

```
method uint spattern.search (
    uint ptr,
    uint size
)
```

Параметры

src Строка в которой ищется указанная строка (шаблон поиска).
offset Смещение с которого продолжить поиск.

Возвращаемое значение

Смещение найденного фрагмента. Если равно размеру строки, то фрагмент не найден.

Смотрите также

- [String](#)

str.search

Поиск подстроки. Метод определяет вхождение одной строки в другую строку.

```
method uint str.search (  
    str pattern,  
    uint flag  
)
```

Параметры

pattern Строка (шаблон) поиска.
flag Флаги поиска.

<code>\$QS_IGNORECASE</code>	Игнорировать регистр при поиске.
<code>\$QS_WORD</code>	Искать только целые слова.
<code>\$QS_BEGINWORD</code>	Искать слова которые начинаются с указанного шаблона.

Возвращаемое значение

Возвращается 1 если подстрока найдена и 0 в противном случае.

Смотрите также

- [String](#)

String - Filename

Работа с именами файлов. Методы для работы с именами файла и директорий.

<code>str.faddname</code>	Добавление имени.
<code>str.fappendslash</code>	Добавить слэш.
<code>str.fdelslash</code>	Удалить конечный слэш.
<code>str.ffullname</code>	Получить полное имя.
<code>str.fgetdir</code>	Получить путь родительской директории.
<code>str.fgetdrive</code>	Получить имя диска.
<code>str.fgettext</code>	Получить расширение.
<code>str.fgetparts</code>	Получить составляющие имени.
<code>str.fnameext</code>	Получить имя файла.
<code>str.fsetext</code>	Изменить расширение.
<code>str.fsetname</code>	Изменить текущее имя файла.
<code>str.fsetparts</code>	Составить или изменить имя.
<code>str.fsplitt</code>	Получить директорию и имя файла.
<code>str.fwildcard</code>	Проверить по маске.

str.faddname

Добавление имени. Добавить имя файла или директории к текущему пути.

```
method str str.faddname (  
    str name  
)
```

Параметры

name The name being added. It will be added after a slash.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Filename](#)

str.fappendslash

Добавить слэш. Добавляет '\' в конец строки если он там отсутствует.

```
method str str.fappendslash()
```

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Filename](#)

str.fdelslash

Удалить конечный слэш. Удалить конечный '\' если он есть.

```
method str str.fdelslash()
```

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Filename](#)

str.ffullname

Получить полное имя. Получить полный путь и имя файла.

```
method str str.ffullname (  
    str name  
)
```

Параметры

name Исходное имя файла.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Filename](#)

str.fgetdir

Получить путь родительской директории. Метод убирает последнее имя файла или директории.

```
method str str.fgetdir (  
    str name  
)
```

Параметры

name Исходное имя файла.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Filename](#)

str.fgetdrive

Получить имя диска. Получить сетевое имя (\\computer\share) или имя диска (c:).

```
method str str.fgetdrive (  
    str name  
)
```

Параметры

name Исходное имя файла.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Filename](#)

str.fgetext

Получить расширение. Метод создает и возвращает новую строку с текущим расширением файла.

```
method str str.fgetext< result >
```

Возвращаемое значение

Результирующая строка с расширением.

Смотрите также

- [String - Filename](#)

str.fgetparts

Получить составляющие имени. Получить директорию, имя и расширение файла.

```
method str.fgetparts (  
  str dir,  
  str fname,  
  str ext  
)
```

Параметры

dir Строка для получения директории. Может быть 0->str.
fname Строка для получения имени файла. Может быть 0->str.
ext Строка для получения расширения файла. Может быть 0->str.

Смотрите также

- [String - Filename](#)

str.fnameext

Получить имя файла. Получить из полного пути имя последнего файла или директории.

```
method str str.fnameext (  
    str name  
)
```

Параметры

name Исходное имя файла.

Смотрите также

- [String - Filename](#)

str.fsetext

- [method str str.fsetext\(str name, str ext \)](#)
- [method str str.fsetext\(str ext \)](#)

Изменить расширение. Получить имя файла с новым расширением.

```
method str str.fsetext (  
    str name,  
    str ext  
)
```

Параметры

name Исходное имя файла.
ext Расширение файла.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

str.fsetext

Изменить расширение в имени файла.

```
method str str.fsetext (  
    str ext  
)
```

Параметры

ext Расширение файла.

Смотрите также

- [String - Filename](#)

str.fsetname

Изменить текущее имя файла.

```
method str str.fsetname (  
    str filename  
)
```

Параметры

filename Новое имя файла.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Filename](#)

str.fsetparts

Составить или изменить имя. Составить имя файла из пути, имени и расширения. Эта функция также может быть использована для изменения пути, имени или расширения. В этом случае если какое-то из составляющих равно 0, то оно остается без изменений.

```
method str str.fsetparts (  
    str dir,  
    str fname,  
    str ext  
)
```

Параметры

<i>dir</i>	Директория.
<i>fname</i>	Имя файла.
<i>ext</i>	Расширение файла.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Filename](#)

str.fspllit

Получить директорию и имя файла. Разделить полный путь на имя конечного файла или директории и остальной путь.

```
method str.fspllit (  
    str dir,  
    str name  
)
```

Параметры

dir Строка для получения директории.

name Строка для получения имени файла или директории.

Смотрите также

- [String - Filename](#)

str.fwildcard

Проверить по маске. Проверить строку на совпадение с указанной маской.

```
method uint str.fwildcard (  
    str wildcard  
)
```

Параметры

wildcard Проверяемая маска. Может содержать '?' (один символ) и '*' (любые символы).

Возвращаемое значение

Возвращает 1 в случае совпадения строки с маской.

Смотрите также

- [String - Filename](#)

String - Unicode

Юникодные строки. Вы можете использовать переменные типа **ustr** для работы с юникодными строками. Тип **ustr** наследуется от типа **buf**, поэтому вы можете также использовать [методы типа buf](#).

- [Операторы](#)
- [Методы](#)

Операторы

<code>* ustr</code>	Получить длину юникодной строки.
<code>ustr[i]</code>	Получение [i] символа в юникодной строке.
<code>ustr + ustr</code>	Сложить две строки.
<code>ustr = type</code>	Присвоить тип юникодной строке.
<code>str = ustr</code>	Копировать юникодную строку в обычную.
<code>ustr += type</code>	Добавить тип к юникодной строке.
<code>str == ustr</code>	Сравнение на равенство.
<code>ustr < ustr</code>	Проверка на меньше.
<code>ustr > ustr</code>	Проверка на больше.
<code>ustr(str)</code>	Конвертировать строку в юникодную строку <code>ustr(str)</code> .
<code>str(ustr)</code>	Конвертировать юникодную строку в строку <code>str(ustr)</code> .

Методы

<code>ustr.clear</code>	Очистить юникодную строку.
<code>ustr.copy</code>	Копирование.
<code>ustr.del</code>	Удалить подстроку.
<code>ustr.findch</code>	Найти символ в юникодной строке.
<code>ustr.fromutf8</code>	Конвертировать строку UTF-8 в юникодную строку.
<code>ustr.insert</code>	Вставка.
<code>ustr.lines</code>	Конвертировать многостроковый юникодный текст в массив юникодных строк.
<code>ustr.read</code>	Чтение юникодной строки из файла.
<code>ustr.replace</code>	Замена в юникодной строке.
<code>ustr.reserve</code>	Резервирование памяти.
<code>ustr.setlen</code>	Установить новый размер строки.
<code>ustr.split</code>	Разделение юникодной строки.
<code>ustr.substr</code>	Получить подстроку из юникодной строки.
<code>ustr.toutf8</code>	Конвертировать юникодную строку в строку UTF-8.
<code>ustr.trim ...</code>	Удаление крайних символов в юникодной строке.
<code>ustr.write</code>	Запись юникодной строки в файл.

* `ustr`

Получить длину юникодной строки.

```
operator uint * (  
    ustr left  
)
```

Возвращаемое значение

Длина юникодной строки.

Смотрите также

- [String - Unicode](#)

ustr[i]

Получение [i] символа в юникодной строке.

```
method uint ustr.index (  
    uint id  
)
```

Возвращаемое значение

[i] символ ushort юникодной строки.

Смотрите также

- [String - Unicode](#)

ustr + ustr

- [operator ustr +<result> \(ustr left, ustr right \)](#)
- [operator ustr +<result>\(ustr left, str right \)](#)

Сложить две строки. Сложение двух юникодных строк и создание новой юникодной строки.

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

Возвращаемое значение

Новая результирующая юникодная строка.

ustr + str

Сложить юникодную и простую строку.

```
operator ustr +<result> (  
    ustr left,  
    str right  
)
```

Возвращаемое значение

Новая результирующая юникодная строка.

Смотрите также

- [String - Unicode](#)

ustr = type

- [operator ustr =\(ustr left, str right \)](#)
- [operator ustr =\(ustr left, ustr right \)](#)

Присвоить тип юникодной строке. Скопировать обычную строку в юникодную **ustr = str**.

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

Возвращаемое значение

Результирующая юникодная строка.

ustr = ustr

Копировать юникодную строку в другую юникодную строку.

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

Смотрите также

- [String - Unicode](#)

str = ustr

Копировать юникодную строку в обычную.

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

Возвращаемое значение

Результирующая строка.

Смотрите также

- [String - Unicode](#)

ustr += type

- [operator ustr +=\(ustr left, ustr right \)](#)
- [operator ustr +=\(ustr left, str right \)](#)

Добавить тип к юникодной строке. Добавить **ustr** к **ustr** => **ustr += ustr**.

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

Возвращаемое значение

Результирующая юникодная строка.

ustr += str

Добавить **str** к **ustr** => **ustr += str**.

```
operator ustr += (  
    ustr left,  
    str right  
)
```

Смотрите также

- [String - Unicode](#)

str == ustr

- [operator uint ==\(str left, ustr right \)](#)
- [operator uint ==\(ustr left, str right \)](#)

Сравнение на равенство.

```
operator uint == (  
    str left,  
    ustr right  
)
```

Возвращаемое значение

Возвращает **1** если юникодные строки равны и **0** в противном случае.

ustr == str

Сравнение на равенство юникодной и простой строки.

```
operator uint == (  
    ustr left,  
    str right  
)
```

Возвращаемое значение

Возвращает **1** если строки равны и **0** в противном случае.

Смотрите также

- [String - Unicode](#)

ustr < ustr

- [operator uint <\(ustr left, ustr right \)](#)
- [operator uint <=\(ustr left, ustr right \)](#)

Проверка на меньше.

```
operator uint < (  
    ustr left,  
    ustr right  
)
```

Возвращаемое значение

Возвращает **1** если первая строка меньше второй и **0** в противном случае.

ustr <= ustr

Проверка на меньше или равно.

```
operator uint <= (  
    ustr left,  
    ustr right  
)
```

Возвращаемое значение

Возвращает **1** если первая юникодная строка меньше или равна второй и **0** в противном случае.

Смотрите также

- [String - Unicode](#)

ustr > ustr

- [operator uint >\(ustr left, ustr right \)](#)
- [operator uint >=\(ustr left, ustr right \)](#)

Проверка на больше.

```
operator uint > (  
    ustr left,  
    ustr right  
)
```

Возвращаемое значение

Возвращает **1** если юникодная первая строка больше второй и **0** в противном случае.

ustr >= ustr

Проверка на больше или равно.

```
operator uint >= (  
    ustr left,  
    ustr right  
)
```

Возвращаемое значение

Возвращает **1** если первая юникодная строка больше или равна второй и **0** в противном случае.

Смотрите также

- [String - Unicode](#)

ustr(str)

Конвертировать строку в юникодную строку **ustr(str)**.

```
method ustr str.ustr<result> (  
)
```

Возвращаемое значение

Результирующая юникодная строка.

Смотрите также

- [String - Unicode](#)

str(ustr)

Конвертировать юникодную строку в строку **str(ustr)**.

```
method str ustr.str<result> (  
)
```

Возвращаемое значение

Результирующая строка.

Смотрите также

- [String - Unicode](#)

ustr.clear

Очистить юникодную строку.

```
method ustr ustr.clear
```

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Unicode](#)

ustr.copy

- [method ustr ustr.copy\(uint ptr, uint size \)](#)
- [method ustr ustr.copy\(uint ptr \)](#)

Копирование. Метод копирует данные указанного размера в стьюникодную строку.

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

Параметры

ptr Указатель на копируемые данные.
size Количество копируемых символов.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

ustr.copy

Метод копирует данные в юникодную строку.

```
method ustr ustr.copy( uint ptr )
```

Параметры

ptr Указатель на копируемые данные. Нулевой символ ushort будет добавлен автоматически.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Unicode](#)

ustr.del

Удалить подстроку.

```
method ustr ustr.del (  
    uint off,  
    uint len  
)
```

Параметры

off Смещение удаляемой подстроки.
len Размер удаляемой подстроки.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Unicode](#)

ustr.fromutf8

Конвертировать строку UTF-8 в юникодную строку.

```
method ustr ustr.fromutf8 (  
  str src  
)
```

Параметры

src Исходная строка UTF-8.

Возвращаемое значение

Возвращается объект для которого был вызван метод..

Смотрите также

- [String - Unicode](#)

ustr.insert

Вставка. Метод вставляет одну юникодную строку в другую.

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

Параметры

offset Смещение куда будет вставляться строка.
value Вставляемая юникодная строка.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [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>\(\)](#)

Конвертировать многостроковый юникодный текст в массив юникодных строк.

```
method arrustr ustr.lines (  
    arrustr ret,  
    uint flag  
)
```

Параметры

re Результирующий массив юникодных строк.

t

fl Флаги.

ag

\$SPLIT_EMPTY Учитывать пустые строки.

\$SPLIT_NOSYS Удалять символы меньше или равные пробелу справа и слева.

\$SPLIT_FIRST Разделять только до первого разделителя.

\$SPLIT_QUOTE Учитывать, что элементы могут быть заключены в одинарные или двойные кавычки.

\$SPLIT_APPEND Добавлять строки. В противном случае, массив будет очищен перед добавлением.

Возвращаемое значение

Результирующий массив юникодных строк.

ustr.lines

Конвертировать многостроковый юникодный текст в массив юникодных строк.

```
method arrustr ustr.lines<result> (  
    uint trim  
)
```

Параметры

trim Укажите 1 если вы хотите удалить крайние символы меньше или равные пробелу.

Возвращаемое значение

Новый результирующий массив юникодных строк.

ustr.lines

Конвертировать многостроковый юникодный текст в массив юникодных строк.

```
method arrustr ustr.lines (  
    arrustr ret  
)
```

Параметры

ret Результирующий массив юникодных строк.

ustr.lines

Конвертировать многостроковый юникодный текст в массив юникодных строк.

```
method arrustr ustr.lines<result>()
```

Возвращаемое значение

Новый результирующий массив юникодных строк.

Смотрите также

- [String - Unicode](#)

ustr.replace

Замена в юникодной строке. Метод заменяет данные в юникодной строке.

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

Параметры

<i>offset</i>	Смещение заменяемых данных.
<i>size</i>	Размер заменяемых данных.
<i>value</i>	Вставляемая юникодная строка.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Unicode](#)

ustr.reserve

Резервирование памяти. Метод увеличивает размер отведенной памяти в юникодной строке.

```
method ustr.reserve (  
    uint len  
)
```

Параметры

len Суммарный запрашиваемый размер памяти. Если он меньше текущего размера, то ничего не происходит. При увеличении размера текущие данные в строке сохраняются.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Unicode](#)

ustr.setlen

- [method ustr ustr.setlen\(uint len \)](#)
- [method ustr ustr.setlenptr](#)

Установить новый размер строки. Метод не занимается резервированием места. Вы не можете устанавливать размер строки больше чем у вас зарезервировано места. В основном, эта функция используется для установки размера строки после записи в нее данных внешними функциями.

```
method ustr ustr.setlen (
    uint len
)
```

Параметры

len Новый размер строки.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

ustr.setlenptr

Пересчитать размер юникодной строки до нулевого символа. Функция может использоваться для установки размера строки после записи в нее данных другими функциями.

```
method ustr ustr.setlenptr
```

Смотрите также

- [String - Unicode](#)

ustr.split

- [method arrustr ustr.split\(arrustr ret, ushort symbol, uint flag \)](#)
- [method arrustr ustr.split <result> \(uint symbol, uint flag \)](#)

Разделение юникодной строки. Метод разбивает строку на подстроки с учетом указанного разделителя.

```
method arrustr ustr.split (  
    arrustr ret,  
    ushort symbol,  
    uint flag  
)
```

Параметры

ret Массив юникодных строк куда будут записаны подстроки.
symb Символ-разделитель.
ol
flag Флаги.

\$SPLIT_EMPTY	Учитывать пустые строки.
\$SPLIT_NOSYS	Удалять символы меньше или равные пробелу справа и слева.
\$SPLIT_FIRST	Разделять только до первого разделителя.
\$SPLIT_QUOTE	Учитывать, что элементы могут быть заключены в одинарные или двойные кавычки.
\$SPLIT_APPEND	Добавлять строки. В противном случае, массив будет очищен перед добавлением.

Возвращаемое значение

Результирующий массив юникодных строк.

Метод разбивает строку на подстроки с учетом указанного разделителя и создает результирующий массив строк.

```
method arrustr ustr.split <result> (  
    uint symbol,  
    uint flag  
)
```

Параметры

symb Символ-разделитель.
ol
flag Флаги.

\$SPLIT_EMPTY	Учитывать пустые строки.
\$SPLIT_NOSYS	Удалять символы меньше или равные пробелу справа и слева.
\$SPLIT_FIRST	Разделять только до первого разделителя.
\$SPLIT_QUOTE	Учитывать, что элементы могут быть заключены в одинарные или двойные кавычки.
\$SPLIT_APPEND	Добавлять строки. В противном случае, массив будет очищен перед добавлением.

Возвращаемое значение

Новый результирующий массив юникодных строк.

Смотрите также

- [String - Unicode](#)

ustr.substr

Получить подстроку из юникодной строки.

```
method ustr ustr.substr (  
    ustr src,  
    uint start,  
    uint len  
)
```

Параметры

<i>src</i>	Исходная юникодная строка.
<i>start</i>	Смещение подстроки.
<i>len</i>	Размер подстроки.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [String - Unicode](#)

ustr.toutf8

Конвертировать юникодную строку в строку UTF-8.

```
method str ustr.toutf8 (  
  str dest  
)
```

Параметры

dest Конечная строка.

Возвращаемое значение

Параметр *dest*.

Смотрите также

- [String - Unicode](#)

ustr.trim...

- [method ustr ustr.trim\(uint symbol, uint flag \)](#)
- [method ustr ustr.trimrspace\(\)](#)
- [method ustr ustr.trimspace\(\)](#)

Удаление крайних символов в юникодной строке.

```
method ustr ustr.trim (
    uint symbol,
    uint flag
)
```

Параметры

symbol Удаляемый символ.

flag

Флаги.

\$TRIM_LEFT	Удалять с левой стороны.
\$TRIM_RIGHT	Удалять с правой стороны.
\$TRIM_ONE	Удалять только один символ.
\$TRIM_PAIR	Если удаляемый символ скобка, то смотреть на закрывающую скобку с правой стороны.
\$TRIM_SYS	Удалять символы меньше или равные пробелу.

Возвращаемое значение

Возвращается объект для которого был вызван метод.

ustr.trimrspace

Удаление пробелов справа.

```
method ustr ustr.trimrspace ()
```

ustr.trimspace

Удаление пробелов с обеих сторон.

```
method ustr ustr.trimspace ()
```

Смотрите также

- [String - Unicode](#)

ustr.write

Запись юникодной строки в файл.

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

Параметры

filename Имя файла для записи. Если файл существует, то он будет перезаписан.

Возвращаемое значение

Размер записанных данных.

Смотрите также

- [String - Unicode](#)

System

Системные функции.

- [Callback и поиск](#)
- [Функции для типов](#)

<code>max</code>	Определить максимальное из двух чисел.
<code>min</code>	Определить минимальное из двух чисел.

Callback и поиск

<code>callback</code>	Создание callback функции.
<code>freecallback</code>	Освободить созданную callback функцию.
<code>getid</code>	Получить код объекта по имени.

Функции для типов

<code>destroy</code>	Удаление объекта.
<code>new</code>	Создание объекта.
<code>sizeof</code>	Получить размер типа.
<code>type_delete</code>	Уничтожить объект определенный указателем.
<code>type_hasdelete</code>	Требуется ли уничтожение объекта.
<code>type_hasinit</code>	Требуется ли инициализация объекта.
<code>type_init</code>	Инициализировать объект определенным указателем.

max

- [func uint max\(uint left, uint right \)](#)
- [func uint max\(int left, int right \)](#)

Определить максимальное из двух чисел.

```
func uint max (
    uint left,
    uint right
)
```

Параметры

left Первое сравниваемое число типа uint.
right Второе сравниваемое число типа uint.

Возвращаемое значение

Функция возвращает максимальное из двух чисел.

max

Определить максимальное из двух целых чисел.

```
func uint max (
    int left,
    int right
)
```

Параметры

left Первое сравниваемое число типа int.
right Второе сравниваемое число типа int.

Возвращаемое значение

Функция возвращает максимальное из двух чисел.

Смотрите также

- [System](#)

min

- [func uint min\(uint left, uint right \)](#)
- [func uint min\(int left, int right \)](#)

Определить минимальное из двух чисел.

```
func uint min (  
    uint left,  
    uint right  
)
```

Параметры

left Первое сравниваемое число типа uint.
right Второе сравниваемое число типа uint.

Возвращаемое значение

Функция возвращает минимальное из двух чисел.

min

Определить минимальное из двух целых чисел.

```
func uint min (  
    int left,  
    int right  
)
```

Параметры

left Первое сравниваемое число типа int.
right Второе сравниваемое число типа int.

Возвращаемое значение

Функция возвращает минимальное из двух чисел.

Смотрите также

- [System](#)

callback

Создание callback функции. Эта функция делает возможным указание gntee функции в качестве callback функции. Например, gntee функция может быть указана в качестве функции обработчика событий для окон.

```
func uint callback (  
    uint idfunc,  
    uint parsize  
)
```

Параметры

idfunc Идентификатор (адрес) gntee функции, которая будет использоваться в качестве callback функции.

parsize Суммарный размер занимаемый параметрами, измеряется в uint. Если функция имеет 1 параметр uint, то указывайте 1 (uint = 1). uint + uint = 2, uint + long = 3.

Возвращаемое значение

Адрес, который можно передавать в качестве callback функции. Его необходимо освободить с помощью функции [freecallback](#) когда gntee функция не будет больше использоваться.

Смотрите также

- [System](#)

freecallback

Освободить созданную callback функцию.

```
func freecallback (  
    uint ptr  
)
```

Параметры

ptr Указатель, который был возвращен [callback](#) функцией.

Смотрите также

- [System](#)

getid

Получить код объекта по имени. Функция возвращает код объекта (функции, метода, оператора, типа) по его имени и параметрам.

```
func uint getid (  
    str name,  
    uint flags,  
    collection idparams  
)
```

Параметры

name Имя объекта (функции, метода, оператора).

flags Флаги.

\$GETID_METHOD	Искать метод. Укажите главный тип метода первым параметром в коллекции.
\$GETID_OPERATOR	Искать оператор. Вы можете указывать оператор как есть в параметре name. Например, +=.
\$GETID_OFTYPE	Укажите этот флаг если вы хотите описать параметры вместе с типом элементов (of тип). В этом случае коллекция должна содержать пары - главный тип и тип элементов.

idparams Типы параметров.

ms

Возвращаемое значение

Код (идентификатор) найденного объекта или **0** если объект не найден.

Смотрите также

- [System](#)

destroy

Удаление объекта. Удаление объекта созданного функцией [new](#).

```
func destroy (  
    uint obj  
)
```

Параметры

obj Указатель на удаляемый объект.

Смотрите также

- [System](#)

new

- [func uint new \(uint objtype \)](#)
- [func uint new \(uint objtype, uint oftype, uint count \)](#)

Создание объекта. Функция создает объект указанного типа.

```
func uint new (  
    uint objtype  
)
```

Параметры

objtype Идентификатор или имя типа.

Возвращаемое значение

Функция возвращает указатель на созданный объект.

new

Функция создает объект указанного типа, определяет тип дочерних элементов и если надо, то создает указанное количество элементов.

```
func uint new (  
    uint objtype,  
    uint oftype,  
    uint count  
)
```

Параметры

objtype Идентификатор или имя типа.

oftype Тип дочерних элементов.

count Первоначальное количество создаваемых элементов.

Возвращаемое значение

Функция возвращает указатель на созданный объект.

Смотрите также

- [System](#)

sizeof

Получить размер типа.

```
func uint sizeof (  
    uint idtype  
)
```

Параметры

idtype Идентификатор или имя типа. Компилятор заменит имя типа его идентификатором.

Возвращаемое значение

Размер типа в байтах.

Смотрите также

- [System](#)

type_delete

Уничтожить объект определенный указателем. Gentee удаляет объекты автоматически. Используйте эту функцию только для отведенных вами в памяти переменных.

```
func type_delete (  
    pubyte ptr,  
    uint idtype  
)
```

Параметры

ptr Указатель на область памяти где расположен удаляемый объект.
idtype Тип объекта.

Смотрите также

- [System](#)

type_hasdelete

Требуется ли уничтожение объекта. Указывает на необходимость вызова функции [type_delete](#) для удаления объекта данного типа.

```
func uint type_hasdelete (  
    uint idtype  
)
```

Параметры

idtype Тип объекта.

Возвращаемое значение

Возвращается **1** при необходимости вызова [type_delete](#) и **0** в противном случае.

Смотрите также

- [System](#)

type_init

Инициализировать объект определенным указателем. Gentee инициализирует объекты автоматически. Используйте эту функцию только для отведенных вами в памяти переменных.

```
func uint type_init (  
    pubyte ptr,  
    uint idtype  
)
```

Параметры

ptr Указатель на область памяти где расположен создаваемый объект.
idtype Тип объекта.

Возвращаемое значение

Возвращается указатель на объект.

Смотрите также

- [System](#)

Thread

Данная библиотека позволяет создавать потоки и работать с ними. Описанные ниже методы применяются к переменным типа **thread**. Для использования библиотеки необходимо с помощью команды `include` указать файл `thread.g`, который находится в поддиректории `lib\thread`.

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

- [Методы](#)
- [Функции](#)

Методы

<code>thread.create</code>	Создать поток.
<code>thread.getexitcode</code>	Получить код окончания потока.
<code>thread.isactive</code>	Проверить активность потока.
<code>thread.resume</code>	Возобновить работу потока.
<code>thread.suspend</code>	Приостановить работу потока.
<code>thread.terminate</code>	Уничтожить поток.
<code>thread.wait</code>	Дождаться окончания данного потока.

Функции

<code>exitthread</code>	Прекратить работу текущего потока.
<code>sleep</code>	Приостановить работу текущего потока на указанное время.

thread.create

Создать поток.

```
method uint thread.create (  
    uint idfunc,  
    uint param  
)
```

Параметры

idfunc Указатель на функцию, которая вызовется как новый поток. Функция должна иметь один параметр. Получить указатель можно с помощью оператора &.

param Дополнительный передаваемый параметр.

Возвращаемое значение

Возвращается дескриптор созданного потока. В случае ошибки возвращается 0.

Смотрите также

- [Thread](#)

thread.getexitcode

Получить код окончания потока.

```
method uint thread.getexitcode (  
    uint result  
)
```

Параметры

result Указатель на переменную типа uint куда будет записан код окончания потока. Если поток еще выполняется, то запишется значение \$STILL_ACTIVE.

Возвращаемое значение

В случае успешного завершения возвращается **1**, в противном случае возвращается **0**.

Смотрите также

- [Thread](#)

thread.isactive

Проверить активность потока.

```
method uint thread.isactive()
```

Возвращаемое значение

Возвращается 1 если поток существует и 0 в противном случае.

Смотрите также

- [Thread](#)

thread.resume

Возобновить работу потока. Возобновить работу потока который был приостановлен методом [thread.suspend](#).

```
method uint thread.resume ()
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Thread](#)

thread.suspend

Приостановить работу потока.

```
method uint thread.suspend()
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Thread](#)

thread.terminate

Уничтожить поток.

```
method uint thread.terminate (  
    uint code  
)
```

Параметры

code Код окончания потока.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Thread](#)

thread.wait

Дождаться окончания данного потока.

```
method uint thread.wait()
```

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Thread](#)

exitthread

Прекратить работу текущего потока.

```
func exitthread (  
    uint code  
)
```

Параметры

code Код окончания потока.

Смотрите также

- [Thread](#)

sleep

Приостановить работу текущего потока на указанное время.

```
func sleep (  
    uint msec  
)
```

Параметры

msec Время остановки в миллисекундах.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Thread](#)

Tree

Объект дерева. Каждый элемент объекта может иметь дочерние элементы. Необходимо подключать файл tree.g.

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

- [Операторы](#)
- [Методы](#)
- [Методы элементов treeitem](#)

Операторы

tree of type	Указание типа элементов.
* tree	Получить количество элементов в дереве.
* treeitem	Получить количество дочерних элементов.
foreach var,treeitem	Оператор foreach.

Методы

tree.clear	Удалить все элементы из дерева.
tree.del	Удалить элемент.
tree.leaf	Добавить "лист".
tree.node	Добавить "узел".
tree.root	Получение корневого элемента.

Методы элементов treeitem

treeitem.changenode	Сменить узел-хозяин у данного элемента
treeitem.child	Получить первый дочерний элемент у объекта.
treeitem.data	Получить указатель на данные хранящиеся в объекте.
treeitem.getnext	Получить следующий элемент дерева.
treeitem.getprev	Получить предыдущий элемент дерева.
treeitem.isleaf	Проверка на "лист".
treeitem.isnode	Проверка на "узел".
treeitem.isroot	Проверка на "корень".
treeitem.lastchild	Получить последний дочерний элемент у объекта.
treeitem.move	Переместить элемент.
treeitem.parent	Получить хозяина объекта.

tree of type

Указание типа элементов. Вы можете определять тип элементов дерева с помощью оператора **of** когда вы описываете переменную типа **tree**. По умолчанию, элементы дерева имеют тип **uint**.

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

Смотрите также

- [Tree](#)

* tree

Получить количество элементов в дереве.

```
operator uint * (  
    tree itree  
)
```

Возвращаемое значение

Количество элементов в дереве.

Смотрите также

- [Tree](#)

* `treeitem`

Получить количество дочерних элементов.

```
operator uint * (  
    treeitem treei  
)
```

Возвращаемое значение

Количество дочерних элементов у данного узла.

Смотрите также

- [Tree](#)

foreach var,treeitem

Оператор foreach. Вы можете использовать оператор **foreach** для перебора всех элементов у данного элемента дерева. **Переменная** содержит очередной дочерний элемент.

```
foreach variable,treeitem {...}
```

Смотрите также

- [Tree](#)

tree.clear

Удалить все элементы из дерева.

```
method tree tree.clear (  
)
```

Возвращаемое значение

Возвращается объект для которого был вызван метод.

Смотрите также

- [Tree](#)

tree.del

- [method tree.del\(treeitem item, uint funcdel \)](#)
- [method tree.del\(treeitem item \)](#)

Удалить элемент. Удаление элемента вместе со всеми потомками.

```
method tree.del (  
    treeitem item,  
    uint funcdel  
)
```

Параметры

item Удаляемый элемент.

funcdel Функция которая будет вызываться перед удалением каждого элемента. Может быть 0.

tree.del

Удаление элемента вместе со всеми потомками.

```
method tree.del (  
    treeitem item  
)
```

Параметры

item Удаляемый элемент.

Смотрите также

- [Tree](#)

tree.leaf

- [method treeitem tree.leaf\(treeitem parent, treeitem after \)](#)
- [method treeitem tree.leaf\(treeitem parent \)](#)

Добавить "лист". Добавить "лист" к указанному узлу. "Лист" является конечным элементом.

```
method treeitem tree.leaf (  
    treeitem parent,  
    treeitem after  
)
```

Параметры

parent Узел-хозяин. Если 0->treeitem, то элемент будет добавлен в корень.

after Элемент после которого необходимо вставлять. Если указан 0->treeitem то элемент будет первым дочерним элементом.

Возвращаемое значение

Добавленный элемент или 0 в случае ошибки.

tree.leaf

Добавить "лист" к указанному узлу. Элемент будем последним дочерним элементом.

```
method treeitem tree.leaf (  
    treeitem parent  
)
```

Параметры

parent Узел-хозяин. Если 0->treeitem, то элемент будет добавлен в корень.

Возвращаемое значение

Добавленный элемент или 0 в случае ошибки.

Смотрите также

- [Tree](#)

tree.node

- [method treeitem tree.node\(treeitem parent, treeitem after \)](#)
- [method treeitem tree.node\(treeitem parent \)](#)

Добавить "узел". Добавить "узел" к указанному узлу. "Узел" позволяет добавлять к себе элементы.

```
method treeitem tree.node (
  treeitem parent,
  treeitem after
)
```

Параметры

parent Узел-хозяин. Если 0->treeitem, то элемент будет добавлен в корень.

after Элемент после которого необходимо вставлять. Если указан 0->treeitem то элемент будет первым дочерним элементом.

Возвращаемое значение

Добавленный элемент или 0 в случае ошибки.

tree.node

Добавить "узел" к указанному узлу. "Узел" позволяет добавлять к себе элементы.

```
method treeitem tree.node (
  treeitem parent
)
```

Параметры

parent Узел-хозяин. Если 0->treeitem, то элемент будет добавлен в корень.

Возвращаемое значение

Добавленный элемент или 0 в случае ошибки.

Смотрите также

- [Tree](#)

tree.root

- [method treeitem tree.root](#)
- [method treeitem treeitem.getroot\(\)](#)

Получение корневого элемента.

```
method treeitem tree.root
```

Возвращаемое значение

Корневой элемент дерева.

treeitem.getroot

Получение корневого элемента.

```
method treeitem treeitem.getroot()
```

Возвращаемое значение

Возвращает корневой элемент дерева.

Смотрите также

- [Tree](#)

treeitem.child

Получить первый дочерний элемент у объекта.

```
method treeitem treeitem.child()
```

Возвращаемое значение

Возвращает первый дочерний элемент или 0 если его нет.

Смотрите также

- [Tree](#)

treeitem.data

Получить указатель на данные хранящиеся в объекте.

```
method uint treeitem.data()
```

Возвращаемое значение

Возвращает указатель на данные.

Смотрите также

- [Tree](#)

treeitem.getnext

Получить следующий элемент дерева.

```
method treeitem treeitem.getnext()
```

Возвращаемое значение

Возвращает следующий элемент дерева.

Смотрите также

- [Tree](#)

treeitem.getprev

Получить предыдущий элемент дерева.

```
method treeitem treeitem.getprev()
```

Возвращаемое значение

Возвращает предыдущий элемент дерева.

Смотрите также

- [Tree](#)

treeitem.isleaf

Проверка на "лист". Метод проверяет является ли "листом" (не может иметь потомков) данный объект.

```
method uint treeitem.isleaf
```

Возвращаемое значение

Возвращает 1 если данный элемент является "листом" дерева и 0 в противном случае.

Смотрите также

- [Tree](#)

treeitem.isnode

Проверка на "узел". Метод проверяет может ли данный объект иметь потомков.

```
method uint treeitem.isnode
```

Возвращаемое значение

Возвращает 1 если данный элемент является "узлом" дерева и 0 в противном случае.

Смотрите также

- [Tree](#)

treeitem.isroot

Проверка на "корень". Метод проверяет является ли данный объект корневым.

```
method uint treeitem.isroot
```

Возвращаемое значение

Возвращает 1 если данный элемент является корневым и 0 в противном случае.

Смотрите также

- [Tree](#)

treeitem.lastchild

Получить последний дочерний элемент у объекта.

```
method treeitem treeitem.lastchild()
```

Возвращаемое значение

Возвращает последний дочерний элемент или 0 если его нет.

Смотрите также

- [Tree](#)

treeitem.move

- [method uint treeitem.move\(treeitem after \)](#)
- [method uint treeitem.move\(treeitem target, uint flag \)](#)

Переместить элемент.

```
method uint treeitem.move (  
    treeitem after  
)
```

Параметры

after Узел, после которого вставить элемент. Укажите 0, если необходимо сделать первым элементом.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

treeitem.move

Переместить элемент.

```
method uint treeitem.move (  
    treeitem target,  
    uint flag  
)
```

Параметры

target Элемент перед или после которого вставлять в зависимости от значения флага.
flag Флаги перемещения.

<code>\$TREE_FIRST</code>	Первый элемент с таким же родителем.
<code>\$TREE_LAST</code>	Последний элемент с таким же родителем.
<code>\$TREE_AFTER</code>	После этого элемента.
<code>\$TREE_BEFORE</code>	Перед этим элементом.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [Tree](#)

treeitem.parent

Получить хозяина объекта.

```
method treeitem treeitem.parent()
```

Возвращаемое значение

Возвращает хозяина данного элемента.

Смотрите также

- [Tree](#)

XML

Обработка XML файлов. Данная библиотека предназначена для обработки XML файла и построения дерева XML. В текущей версии не поддерживается многобайтная кодировка, также игнорируется описание структуры документа `<!DOCTYPE .....`. Для использования библиотеки необходимо с помощью команды `include` указать файл `xml.g`, который находится в поддиректории `libxml`.

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

- [Операторы](#)
- [Методы](#)
- [Методы для элементов дерева XML](#)

Описание XML	Краткое описание XML библиотеки.
Операторы	
<code>foreach var,xml item</code>	Оператор <code>foreach</code> .
Методы	
<code>xml.addentity</code>	Добавить определение сущности (entity).
<code>xml.getroot</code>	Получить корневой элемент дерева XML документа.
<code>xml.procfile</code>	Обработать XML файл.
<code>xml.procstr</code>	Обработать XML строку.
Методы для элементов дерева XML	
<code>xmlitem.chtag</code>	Метод получает элемент-тэг по его "пути" в дереве XML.
<code>xmlitem.findtag</code>	Найти элемент-тэг по имени.
<code>xmlitem.getattrib</code>	Метод получает значение атрибута элемента-тэга.
<code>xmlitem.getchild</code>	Получить первый дочерний элемент текущего элемента.
<code>xmlitem.getchildtag</code>	Метод получает первый дочерний элемент-тэг.
<code>xmlitem.getchildtext</code>	Метод получает первый дочерний элемент-текст.
<code>xmlitem.getname</code>	Получить имя элемента XML.
<code>xmlitem.getnext</code>	Получить следующий элемент.
<code>xmlitem.getnexttag</code>	Получить следующий элемент-тэг.
<code>xmlitem.getnexttext</code>	Получить следующий элемент-текст.
<code>xmlitem.getparent</code>	Получить родительский элемент текущего элемента.
<code>xmlitem.gettext</code>	Получить текст текущего элемента.
<code>xmlitem isemptytag</code>	Имеются ли дочерние элементы.
<code>xmlitem.ispitag</code>	Определить является ли элемент тэгом инструкции обработки.
<code>xmlitem.isitag</code>	Определить является ли элемент тэгом.
<code>xmlitem.istext</code>	Определить является ли элемент текстом.

Описание XML

Краткое описание XML библиотеки. Переменные типа **xml** и **xmlitem** (элемент дерева XML) обеспечивают работу с XML документом. Элементом дерева XML, может быть **элемент-текст** и **элемент-тэг**. Элемент-тэг имеет несколько типов:

- элемент-тэг, который может содержать в себе другие элементы `<tag ...>.....</tag>`;
- элемент-тэг не содержащий в себе других элементов `<tag .../>`;
- элемент-тэг инструкции обработки `<?tag ...?>`.

Элемент-тэг может содержать в себе атрибуты.

Порядок работы с XML документом:

- обрабатывается документ (строится дерево XML) с помощью методов [xml.procfiler](#) или [xml.procfiler](#);
- добавляются определения сущностей, если необходимо, с помощью метода [xml.addentity](#);
- осуществляется поиск необходимых элементов в дереве XML, методы: [xml.getroot](#), [xmlitem.htag](#), [xmlitem.findtag](#), [xmlitem.getnext](#), и т.д.;
- для обработки однородных элементов можно использовать операцию **foreach**;
- доступ к атрибутам тэгов осуществляется методом [xmlitem.getattrib](#), получение текста - [xmlitem.gettext](#).

Смотрите также

- [XML](#)

foreach var,xmlitem

Оператор foreach. Просмотр всех элементов с помощью оператора **foreach**. Требуется описание дополнительной переменной типа **xmltags**. Оператор foreach применяется для переменных типа **xmlitem** и перебирает все дочерние элементы-теги у исходного.

```
xmltags xtags
xmlitem curtag
...
foreach xmlitem cur, curtag.tags( xtags )
{
    ...
}
foreach variable,xmlitem.tags( xmltags ) {...}
```

Смотрите также

- [XML](#)

xml.addentity

Добавить определение сущности (entity). Метод добавляет определение сущности (entity), сущность должна быть определена до использования метода `xmlitem.gettext`. По умолчанию определены следующие сущности:

& - &;
" - ";
' - ';
> - >;
< - <;

```
method xml.addentity (  
    str key,  
    str value  
)
```

Параметры

key Ключ (имя сущности - **&entity_name**;).
value Значение сущности, строка которая будет вставляться в текст.

Смотрите также

- [XML](#)

xml.getroot

Получить корневой элемент дерева XML документа. Корневой элемент не несет в себе никакой информации, он только содержит в себе все элементы дерева XML документа.

```
method xmlitem xml.getroot()
```

Возвращаемое значение

Возвращает корневой элемент.

Смотрите также

- [XML](#)

xml.procstr

Обработать XML строку. Метод обрабатывает строку содержащую XML документ.

```
method uint xml.procstr (  
    str src  
)
```

Параметры

src Строка с XML данными.

Возвращаемое значение

В случае успешного завершения возвращается 1, в противном случае возвращается 0.

Смотрите также

- [XML](#)

xmlitem.htag

Метод получает элемент-тэг по его "пути" в дереве XML. "Путь" состоит из имен тэгов разделённых символом '/', если путь начинается с символа '/', то просмотр начинается с корня дерева, иначе с текущего элемента.

```
method xmlitem xmlitem.htag (  
    str path  
)
```

Параметры

path Путь к элементу.

Возвращаемое значение

Возвращает найденный элемент или 0, если элемент не найден.

Смотрите также

- [XML](#)

xmlitem.findtag

Найти элемент-тэг по имени. Поиск ведётся рекурсивно по всем дочерним элементам.

```
method xmlitem xmlitem.findtag (  
    str name  
)
```

Параметры

name Имя искомого тэга.

Возвращаемое значение

Возвращает найденный элемент или 0, если элемент не найден.

Смотрите также

- [XML](#)

xmlitem.getattrib

Метод получает значение атрибута элемента-тега.

```
method str xmlitem.getattrib (  
    str name,  
    str result  
)
```

Параметры

name Имя атрибута.
result Строка для получения результата.

Возвращаемое значение

Возвращает строку - значение атрибута. Если атрибут не был найден, возвращается пустая строка.

Смотрите также

- [XML](#)

xmlitem.getchild

Получить первый дочерний элемент текущего элемента.

```
method xmlitem xmlitem.getchild()
```

Возвращаемое значение

Возвращает дочерний элемент или 0, если не имеется дочерних элементов.

Смотрите также

- [XML](#)

xmlitem.getchildtag

Метод получает первый дочерний элемент-тэг. Метод аналогичен [xmlitem.getchild](#), но в данном случае проверяется является ли элемент тэгом, если нет, то ищется первый элемент-тэг среди дочерних элементов.

```
method xmlitem xmlitem.getchildtag()
```

Возвращаемое значение

Возвращает дочерний элемент-тэг или 0, если не имеется элементов-тэгов.

Смотрите также

- [XML](#)

xmlitem.getchildtext

Метод получает первый дочерний элемент-текст. Метод аналогичен [xmlitem.getchild](#), но в данном случае проверяется является ли элемент текстом, если нет, то ищется первый элемент-текст среди дочерних элементов.

```
method xmlitem xmlitem.getchildtext()
```

Возвращаемое значение

Возвращает дочерний элемент-текст или 0, если не имеется элементов-текстов.

Смотрите также

- [XML](#)

xmlitem.getname

Получить имя элемента XML.

```
method str xmlitem.getname (  
    str res  
)
```

Параметры

res Результирующая строка.

Возвращаемое значение

Возвращается параметр *res*.

Смотрите также

- [XML](#)

xmlitem.getnext

Получить следующий элемент. Метод получает следующий элемент дерева XML, поиск следующего элемента ведётся среди элементов имеющих один и тоже родитетельский элемент.

```
method xmlitem xmlitem.getnext()
```

Возвращаемое значение

Возвращает следующий элемент или 0, если элемент последний.

Смотрите также

- [XML](#)

xmlitem.getnexttag

Получить следующий элемент-тэг. Метод получит следующий элемент-тэг. Метод аналогичен [xmlitem.getnext](#), но в данном случае проверяется является ли элемент тэгом, если нет, то процесс повторяется.

```
method xmlitem xmlitem.getnexttag
```

Возвращаемое значение

Возвращает следующий элемент-тэг или 0, если элемент последний.

Смотрите также

- [XML](#)

xmlitem.getnexttext

Получить следующий элемент-текст. Метод получит следующий элемент-текст. Метод аналогичен [xmlitem.getnext](#), но в данном случае проверяется является ли элемент текстом, если нет, то процесс повторяется.

```
method xmlitem xmlitem.getnexttext()
```

Возвращаемое значение

Возвращает следующий элемент-текст или 0, если элемент последний.

Смотрите также

- [XML](#)

xmlitem.getparent

Получить родительский элемент текущего элемента.

```
method xmlitem xmlitem.getparent()
```

Возвращаемое значение

Возвращает родительский элемент или 0, если текущий элемент корневой.

Смотрите также

- [XML](#)

xmlitem.gettext

Получить текст текущего элемента. Данный метод можно применять как к элементу-тексту, так и к элементу-тэгу, в этом случае текст берется из дочернего элемента-текста.

```
method str xmlitem.gettext (  
    str result  
)
```

Параметры

result Результирующая строка.

Возвращаемое значение

Возвращает строку содержащую текст элемента. Если элемент не содержит текста возвращается пустая строка.

Смотрите также

- [XML](#)

xmlitem.isemptytag

Имеются ли дочерние элементы. Метод определяет является ли текущий элемент дерева XML тэгом не имеющим дочерних элементов `<tag .../>`;

```
method uint xmlitem.isemptytag()
```

Возвращаемое значение

Возвращает число отличное от нуля если элемент является тэгом не имеющим дочерних элементов, иначе 0.

Смотрите также

- [XML](#)

xmlitem.ispitag

Определить является ли элемент тэгом инструкции обработки. Метод определяет является ли текущий элемент дерева XML тэгом инструкции обработки `<?tag ...?>`.

```
method uint xmlitem.ispitag()
```

Возвращаемое значение

Возвращает число отличное от нуля если элемент является тэгом инструкции обработки, иначе 0.

Смотрите также

- [XML](#)

xmlitem.istag

Определить является ли элемент тэгом. Метод определяет является ли текущий элемент дерева XML тэгом.

```
method uint xmlitem.istag()
```

Возвращаемое значение

Возвращает число отличное от нуля если элемент является тэгом, иначе 0.

Смотрите также

- [XML](#)

xmlitem.istext

Определить является ли элемент текстом. Метод определяет является ли текущий элемент дерева XML текстом.

```
method uint xmlitem.istext()
```

Возвращаемое значение

Возвращает число отличное от нуля если элемент является текстом, иначе 0.

Смотрите также

- [XML](#)

Примеры

Мы рады, что Вы решили поглубже познакомиться с языком программирования Gentee. Здесь мы рассмотрим примеры его практического применения.

Мы писали эти уроки в расчете на любой базовый уровень. Неважно, умеете Вы программировать и знаете какие-то языки или нет. Все уроки расположены в порядке возрастания сложности. Если урок слишком прост для Вас, просто пропустите его. Если же Вы новичек, то мы рекомендуем проходить уроки по порядку.

Мы не будем рассматривать синтаксис и семантику языка программирования Gentee. Это все описано в документации. Здесь мы пытаемся дать какие-то практические навыки и возможности и особенности языка описываются только по мере необходимости.

В каждом уроке мы решаем какую-то задачу. Кроме этого, в конце урока дается упражнение для самостоятельного выполнения. Программированию можно обучиться только на практике, поэтому мы рекомендуем Вам выполнять эти упражнения.

Для удобства, имеются исходные тексты программ для всех встречающихся задач и упражнений. Таким образом, в случае затруднения при решении упражнений, Вы всегда можете воспользоваться данными текстами в качестве подсказки. Все исходные тексты включены в дистрибутив Gentee и находятся в поддиректории **Samples**.

hello	Простой пример вывода строки на консоль.
square	Подсчет площади и периметра круга и прямоугольника.
easymath	Нахождение НОД, факториала и чисел Фиобоначи.
primenumbers	Нахождение простых чисел с помощью "решета Эратостена".
fileattrib	Установка атрибутов файлов.
runini	Использование INI файлов
easyhtml	Вывод палитры цветов в виде HTML, которые наиболее часто используются при создании html страниц.
calendar	Составление календаря в виде HTML на любой год.
samefiles	Нахождение одинаковых файлов на диске.

Вы можете присылать нам задачи, решения которых могут быть полезны для многих людей. Если они покажутся нам интересными и мы сами сможем их решить, то мы обязательно разберем и опубликуем их здесь.

hello

Как правило, первой программой при знакомстве с языком программирования является вывод строки "Hello, World!". Мы тоже не будем оригинальными и попробуем сделать это же самое.

Пример 1

```
func hello <main>
```

```
{  
    print( "Hello, World!" )  
    getch()  
}
```

Разберем этот код более подробно. Есть такое понятие как функция - это набор операций производящих какое-то действие. Функции можно вызывать из других функций. В Gentee функции описываются ключевым словом **func**. Выше мы видим функцию с именем **hello** и с атрибутом **main**, который сообщает о том, что данная функция должна быть запущена после загрузки программы.

```
print( "Hello, World!" )
```

Это вызов функции **print**, которая отвечает за вывод указанной строки. После этого мы вызываем функцию **getch** для ожидания нажатия клавиши.

Есть очень много встроенных функций в языке Gentee. Их описание можно найти в документации. Здесь и далее мы будем вводить новые функции и методы только по мере необходимости.

Теперь поговорим немного о строках. Строки в Gentee заключаются в двойные кавычки. Имеется служебный символ `\`, который в зависимости от следующих символов производит какие-то действия. Вот некоторые из них:

`\n` обозначает перевод строки.

`\\` вывод символа `\`.

Кроме этого, Gentee сохраняет все переносы внутри строки. Две строки ниже эквивалентны.

```
"Hello, World!
```

```
Hello, World!"
```

```
"Hello, World!\nHello, World!"
```

Упражнение 2

Напишите программу вывода "Hello, World!" с предложением нажать любую клавишу.

square

В этом уроке мы начнем знакомство с числами. Давайте напишем программу вычисления площади прямоугольника и круга. При вычислениях будем использовать числа с двойной точностью `double`. Для начала сделаем каркас нашей функции.

```
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")
        }
    }
}
```

Вы видите здесь два новых оператора: **while** и **switch**.

Оператор **while** осуществляет выполнение тела цикла пока условное выражение не равно 0. В нашем случае указана единица, что означает бесконечный цикл и выход из цикла будет осуществляться с помощью команды **break**, которую Вы встретите ниже.

Оператор **switch** вычисляет выражение и ищет данное значение в значениях **case**. Программа ждет когда пользователь нажмет клавишу и смотрит что делать дальше. Остановимся на строке

```
case '3', 27 : break
```

Мы видим, что в **case** можно перечислять через запятую возможные значения. 27 - это код клавиши **Esc**. Что же касается ':', то оно означает включение следующей строки в фигурные скобки. То есть данный фрагмент эквивалентен следующему:

```
case '3', 27 { break }
```

Дело в том, что Gentee почти везде требует использование фигурных скобок и использование ':' в простейших случаях помогает избежать лишнего их нагромождения.

Определимся что нам требуется для вычислений: переменная типа строка для получения значений от пользователя и две переменные типа `double` для хранения величин. Добавим перед циклом

```
str    input
double width height
```

Переменные одного типа перечисляются через запятую или пробел.

Сейчас займемся получением данных и вычислениями. Вот как будет выглядеть подсчет площади прямоугольника.

```
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")
```

Функция **conread** считывает данные введенные пользователем. Операция `\(...)` внутри строки вычисляет выражение в скобках и вставляет результат в строку.

Аналогично делаем для вычисления площади круга

```
print("Specify the radius of the circle: ")
width = double( conread( input ))
print("The area of the circle: \( 3.1415 * width * width )\n\n")
```

Упражнение 2

Напишите программу вычисления периметра прямоугольника и длины окружности. Подсчет периметра и длины окружности оформите в виде двух функций.

easymath

Пример 1

Необходимо найти наибольший общий делитель (НОД) двух чисел.

Для решения этой задачи воспользуемся алгоритмом Евклида. Он звучит следующим образом.

$\text{НОД}(x, y) = x$ если y равно 0 и

$\text{НОД}(x, y) = \text{НОД}(y, x \text{ MOD } y)$ если y не равно 0.

$x \text{ MOD } y$ - это получение остатка от деления.

То есть, мы находим остаток от деления двух чисел и если он не равен 0, то рассматриваем уже второе число и полученный остаток от деления и т.д.

В этом алгоритме мы видим явный пример рекурсии - вызов функции самой себя. Выглядеть эта функция будет вот так.

```
func uint gcd( uint first second )
{
    if !second : return first
    return gcd( second, first % second )
}
```

% - операция получения остатка от деления.

uint - тип обозначающий положительное целое число.

if - условный оператор, который в полном представлении имеет следующий вид.

```
if condition {
}
elif condition {
}
else {
}
```

Блоков **elif** может быть сколько угодно. Если условие выполнено, то будут исполняться следующие за условием операторы в фигурных скобках.

Сейчас осталось написать главную функцию, которая будет получать данные от пользователя и вызывать **gcd**. Она может содержать следующий цикл.

```
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 - функция стандартной библиотеки, которая выводит текст и получает данные от пользователя.

Пример 2

Вычислить факториал $n!$ для n от 1 до 12. Факториал - это произведение всех чисел до данного числа включительно.

Вот какая программа может быть написана в качестве решения.

```
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")
}

for num n = 1, 13
{
    print("\(n)! = \(factorial( n ))\n")
}
```

```
    }  
    getch ()  
}
```

Цикл **for** выполняется пока переменная-счетчик (в нашем случае n) меньше значения второго выражения. На каждом шаге переменная-счетчик увеличивается на 1. **for** - частный случай более общего оператора цикла **for**, с которым мы много раз встретимся в дальнейшем.

```
for counter = expression, expression, change of the value of counter  
{  
}
```

Упражнение 3

Выведите числа Фибоначи пока очередное число не превысит 2000000000. Для вычисления используйте рекурсивную функцию. Каждое из чисел Фибоначи равно сумме двух предыдущих.

$X_0 = 1$

$X_1 = 1$

...

$X_n = X_{n-1} + X_{n-2}$

Упражнение 4

Решите предыдущее упражнение без использования рекурсии.

primenumber

Пример 1

Нахождение простых чисел с помощью "решета Эратостфена".

Конкретизируем задачу. Простые числа - это такие числа, которые имеют только два множителя: 1 и это число. Другими словами, простое число делится только на 1 и на само себя. Метод нахождения простых чисел с помощью "решета Эратостфена" заключается в следующем. Выписываем все числа от двойки до заданного числа. Два - простое число. Зачеркиваем все остальные четные числа. Переходим к тройке и зачеркиваем после нее каждое третье число. Находим следующее незачеркнутое. Это пять и зачеркиваем далее каждое пятое и т.д.. Таким образом в конце все незачеркнутые числа будут простыми.

Разобьем задачу на два шага. На первом шаге, мы непосредственно реализуем этот алгоритм, а на втором займемся выводом результатов.

```
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

for num i = 2, high/2 + 1
{
    if !sieve[ i ]
    {
        j = i + i
        while j <= high
        {
            sieve[ j ] = 1
            j += i
        }
    }
}
```

В начале мы просим пользователя ввести число до которого мы будем находить простые числа. Ограничим это число 100000 чтобы не занимать много ресурсов.

```
arr  sieve[ high + 1 ] of byte
```

Это описание массива байтов *sieve*. Нумерация элементов массива начинается с нуля поэтому увеличиваем на 1 его размер. Все массивы и переменные при создании инициализируются нулями. Таким образом, решим, что если элемент массива равен 0, то соответствующее число не зачеркнуто. Если зачеркиваем число, то ставим там 1.

В цикле **for num** мы проходим только по половине чисел. Этого достаточно, а почему - подумайте сами. Далее мы реализуем сам алгоритм. Как видите, это занимает несколько строк. Мы предлагаем Вам разобрать его работу в качестве упражнения.

```
j += i
```

Это увеличение переменной *j* на *i*. Есть аналогичные операции для умножение, деления и вычитания.

Все лишние числа зачеркнули - пора приступить ко второму шагу.

Можно конечно вывести все эти числа на экран, но давайте попробуем сохранить их в файле.

```
j = 0
input.setlen( 0 )

for num i = 2, high + 1
{
    if !sieve[ i ]
    {
        input.out4( "%8u", i )
        if ++j == 10
        {
            j = 0
            input += "\n"
        }
    }
}
```

```
    }  
  }  
}
```

```
input.write( "prime.txt" )  
shell( "prime.txt" )
```

Описание метода **out4** можно найти в документации. В данном случае мы каждое число дополняем пробелами до ширины в 8 символов. Кроме этого переходим на новую строку после вывода каждых 10 чисел. За это отвечает переменная *j*.

В текстовых файлах для обозначения новой строки обычно используется комбинация возврата каретки '\r' и перевода строки '\n'. В Gentoo для этого служит одна команда '\l'.

Метод **write** записывает строку в файл, а функция **shell** открывает указанный файл в соответствующем для данного файла приложении.

fileattrib

В этом уроке мы поработаем с файлами.

Пример 1

Необходимо у файлов установить или снять атрибут **только чтение**. Должна присутствовать возможность указания параметров в командной строке. Файлы могут быть определены в виде маски с использованием '*' и '?'. '*' - определяет любую последовательность символов, '?' - любой символ.

Так

```
c:\temp\*. * - все файлы в директории c:\temp
c:\temp\*.exe - все файлы с расширением exe в директории c:\temp
c:\temp\*.a.* - все файлы начинающиеся с 'a' в директории c:\temp
```

Начнем с разбора командной строки. Здесь нет ничего сложного, так как имеется две функции: **argc** - получить количество аргументов, **argv** - получить конкретный параметр. Первым параметром должно быть слово **on** или **off** для установки или снятия атрибута и вторым параметром маска для обрабатываемых файлов. Вот что у нас вышло.

```
if argc() > 1
{
    if argv( temp, 1 ) %== "on" : mode = 1
    elif argv( temp, 1 ) %== "off" : mode = 2
    argv( path, 2 )
}
```

Оператор '%==' производит сравнение строк без учета регистра. В этом случае вы можете писать и **ON** и **Off**.

Если параметры не были указаны при запуске программы или были указаны неверно, то надо дать возможность ввести необходимую информацию в консоли.

```
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 )
}
```

Здесь предлагается ввести **1** для установки атрибута, **2** для его снятия и **3** для выхода из программы. Функция **conrequest** ожидает от пользователя нажатия клавиши и возвращает номер выбранного варианта с 0.

Например,

```
conrequest( "Press #'Y#' or #'N#'", "Yy|Nn" )
```

Сейчас можно приступить непосредственно к реализации задания. Для поиска файлов существует структура **ffind**. Опишем переменную **fd** типа **ffind** и инициализируем ее.

```
fd.init( path, $FIND_FILE | $FIND_RECURSE )
$FIND_FILE - указывает на то, что ищем файлы
$FIND_RECURSE - будем искать файлы во всех поддиректориях.
```

Например,

```
fd.init( "c:\\temp.txt", $FIND_FILE | $FIND_RECURSE )
с указанным флагом $FIND_RECURSE будет искать файл temp.txt на всем диске C:.
```

Для перебора файлов используем оператор **foreach**

```
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" )
}
```

info - тип хранящий информацию о файле. Его описание можно найти в помощи.

cur - переменная данного типа в которой будет храниться информация об очередном найденном файле.

То, что находится внутри цикла прокомментирую вкратце. Получаем текущие атрибуты файла

```
attrib = getfileattrib( cur.fullname )
```

В зависимости от режима устанавливаем или сбрасываем атрибут **только чтение**. Все остальные атрибуты сохраняются.

```
if mode == 1 : attrib |= $FILE_ATTRIBUTE_READONLY
```

```
else : attrib &= ~$FILE_ATTRIBUTE_READONLY
```

Записываем измененные атрибуты файла

```
setfileattrib( cur.fullname, attrib )
```

runini

Давайте попробуем сейчас заняться более серьезной автоматизацией. Согласитесь, что не очень удобно пользоваться программой ge2exe из командной строки. Попробуем как-то облегчить этот процесс.

Пример 1

Пусть есть некий ini-файл, которые содержит информацию о программах на языке Gntee. Необходимо дать возможность пользователю выбрать программу из списка, откомпилировать ее и если необходимо создать EXE файл.

Опишем формат ini-файла с которым будем работать. Каждая секция будет обозначать одну программу и может иметь следующие поля:

Name - имя программы.

Src - .g файл программы.

Exe - создавать или нет exe файл (1 или 0).

Run - запускать или нет после компиляции.

Output - Если вы хотите дать конечному файлу другое имя или сохранить его в другой директории, то укажите его имя с путем здесь.

Обязательным является только поле **Src**.

```
[ID2]
Name = Square
Src = ..\square\square.1.g
Exe = 0
```

Вы можете сами изменять этот ini-файл как Вам угодно. В том числе, добавлять новые элементы. В качестве примера мы будем использовать файл **runini.ini** в поддиректории **samples\runini**.

Для работы с ini-файлом нам потребуется библиотека **ini.g**. Подключим ее с помощью команды **include**. Для упрощения мы подразумеваем, что наши примеры находятся в поддиректории **samples** и поэтому используем относительные пути. Если вы захотите перенести пример в другое место, то изменяйте пути на абсолютные.

```
include : $"..\..\lib\ini\ini.g"
```

Строка начинающаяся со знака '\$' не имеет служебных символов, но распознает макросы. Такие строки удобно использовать для описания путей к файлам потому что не надо удваивать знак '\

Создадим две вспомогательные функции.

```
func uint openini( ini retini )
```

Функция **openini** будет считывать runini.ini файл и выдавать сообщение об ошибке в случае его отсутствия. Код ее можете посмотреть в исходном тексте.

```
func uint idaction( ini retini, str section )
```

Эта функция более важная. Она будет вызывать программы компиляции и создания exe файла. Первый параметр - объект типа ini, второй - имя секции-программы которую надо запустить.

Следующие операторы считывают значения полей.

```
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, "" )
```

Последний параметр в функциях **getvalue** и **getnum** определяет значение в случае отсутствия данного поля в ini-файле.

Код ниже формирует командные строки для запуска компилятора и ge2exe в зависимости от полученных опция из ini-файла. Запуск программ происходит с помощью функции **process**. Указание "." в качестве второго параметра при вызове **process** означает, что текущая директория будет рабочей директорией для gentee.exe и ge2exe.exe.

```
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 )

```

Сейчас рассмотрим тело главной функции, которая будет выводить список возможных программ и получать от пользователя имя выбранной программы.

```

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 )
}

```

В начале считываем ini-файл и получаем список секций в виде массива строк. Далее выводим список программ в окно и запрашиваем у пользователя имя секции-программы. После этого вызываем функцию **idaction** с выбранным именем секции.

Более подробно разберем следующую строчку.

```
print( "\("cur)".fillspacer( 20 ) + name + "\n" )
```

Метод **fillspacer** дополняет строку пробелами до необходимой длины. Вы видите, что мы применяем метод не к переменной типа строка, а к строке в двойных кавычках. Это можно делать потому, что строка в двойных кавычках такой же объект как и переменная типа строка. Более того, мы можем применять методы и к функциям или методам возвращающим строки.

Например, данное выражение дополнит нашу строку десятью пробелами слева и доведет общую длину строки до 30 символов.

```
"ID: \("cur)".fillspacer( 20 ).fillspacel( 30 )
```

Упражнение 2

Напишите программу, которая используя `gunini.1.g` может принимать имя секции-программы из командной строки и производить ее запуск. Если параметр командной строки не указан, то она должна работать как вышеразобранная программа.

У меня эта программа заняла 14 строк. Можете открыть `gunini.2.g` и убедиться сами.

easyhtml

В этом уроке мы познакомимся с одним интересным классом функций. Это **text** функции. Как видно из названия, эти функции ориентированы на работу с текстом. В отличие от обычных функций основу их составляет текст в который может быть встроен код.

Пример 1

Выведем палитру цветов, которые наиболее часто используются при создании html страниц. Результат работы сохраним в виде html файла.

Для начала определим количество выводимых цветов в одной строке с помощью команды `define`.

```
define {  
    lcount = 12  
}
```

Такие константные величины называются макросами и при обращении к ним впереди имени должен стоять знак '\$'. Разбор кода начнем с конца

```
func color< main >  
{  
    str out  
  
    out @ colorhtm()  
    out.write( "color.htm" )  
    shell( "color.htm" )  
}
```

```
out @ colorhtm()
```

Здесь мы осуществляем вызов **text** функции **colorhtm** с записью результата в строку `out`. Следующими командами мы записываем полученную строку в файл и открываем его в браузере.

```
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 = 0xFFFFFFFF, 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()  
}  
...
```

\!

Я заменил многоточием вывод заголовка и окончания html файла. Для вставки кода в текст используется команда `{...}`.
Ниже мы определяем подфункцию **outitem**.

```
rgb.out4( "%06X", vrgb )  
@ item( rgb )
```

Здесь на основе локальной переменной **vrgb** создается строка в 16-ом представлении и вызывается другая **text** функция **item** для вывода ячейки с указанным цветом. Унарная операция **@** осуществляет вывод в текущую строку вывода или на консоль, если такая строка отсутствует. Далее проверяем количество выведенных ячеек в строке и если необходимо начинаем новую строку в в нашей таблице.

Для перебора возможных значений используем три вложенных цикла. Каждый цикл отвечает за изменение красной, зеленой или синей составляющей. Компонуем эти составляющие в переменной **vrgb** и вызываем описанную выше подфункцию.

Четыре следующих цикла выводят дополнительные палитры для серого, красного, зеленого и синего цветов.
Команда **!** указывает на окончание **text** функции. По умолчанию, **text** функция может идти до конца файла.

Рассмотрим **text** функцию вывода ячейки.

```
text item( str rgb )  
<TD ALIGN=CENTER><TABLE BGCOLOR=#\ (rgb) WIDTH=60><TR><TD> </TD></TR></TABLE>  
<FONT FACE="Courier">\ (rgb) </FONT>  
</TD>  
\!
```

Как видите это текст на языке HTML с выводом параметра **rgb** содержащего цвет. Мы вставляем его в двух местах: как фон таблицы и для вывода его значения под ячейкой.

Упражнение 2

Создайте HTML файл с таблицей умножения.

calendar

Закрепим наше знакомство с text функциями.

Пример 1

Создать календарь в виде HTML файла на указанный пользователем год.

Вещь полезная, так что приступим. Главную функцию возьмем из предыдущего примера и сделаем небольшие изменения, а именно запросим год на который создавать календарь.

```
congetstr( "Enter a year: ", year )
out @ calendar( uint( year ) )
out.write( "calendar.htm" )
shell( "calendar.htm" )
```

В text функции **calendar** описываем переменную типа **datetime** и устанавливаем ей дату 1 января указанного года. Далее выводим заголовок HTML файла и приступаем к формированию календаря.

```
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>
```

Календарь разделим на три столбца и четыре строки. Переменная **firstday** будет хранить номер первого дня недели для текущего пользователя. **dayofweek** будет содержать номер дня недели для текущей даты. Названия месяцев на языке пользователя получим с помощью функции **nameofmonth**.

```
firstday = firstdayofweek()
dayofweek = stime.dayofweek
fornum i = 0, 4
{
    @"\1<TR>"
    fornum j = 1, 4
    {
        month = i * 3 + j
        @"\1<TD>\ (nameofmonth( stemp, month ))
<PRE>"
        ...
    }
}
```

Для получения сокращенных названий дней недели воспользуемся функцией **abbrnameofday**. Так как для календаря используем вывод с одинаковой шириной всех символов, то нам необходимо добавлять недостающие пробелы. На каждый день отведем ширину в четыре символа.

```
fornum k = firstday, firstday + 7
{
    @" \ ( abbrnameofday( stemp, k ).setlen( 2 ))"
}
@" \1"
@" ".repeat( ( 7 + dayofweek - firstday ) % 7 )
```

При выводе дней месяца выделяем красным воскресные дни - это когда **dayofweek** равен 0. Также следим за переносом строки после вывода последнего дня недели и в переменной **lines** сохраняем количество выведенных строк.

```
uint day = 1
```

```

uint lines
while day <= daysinmonth( year, month )
{
    if !dayofweek : @"<FONT COLOR=red>"
    @str( day++ ).fillspace1( 4 )
    if !dayofweek : @"</FONT>"

    dayofweek = ( dayofweek + 1 ) % 7
    if dayofweek == firstday
    {
        @" \1"
        lines++
    }
}

```

В заключении последнюю строку дополняем пробелами и выводим недостающие строки для того, чтобы все месяцы у нас имели одинаковую высоту.

```
@" ".repeat( ( 7 + firstday - dayofweek ) % 7 )
```

```

while lines++ < 7 : @" \1"
@"</PRE>"

```

Пример вышел немного трудноватым для понимания из-за большого количества текста на языке HTML и дополнительного форматирования. С другой стороны, результат программы получился вполне приемлемый.

samefiles

В этом уроке предлагаю поработать с файлами. Давайте попробуем найти одинаковые файлы на Вашем компьютере. Причем сделаем задачу более интересной и полезной - будем находить одинаковые файлы не по их названию, а по содержанию. Я думаю, что результаты работы программы вас сильно удивят.

Пример 1

Необходимо найти в указанной папке или на диске все одинаковые по содержанию файлы.

Задача на первый взгляд будет выполняться очень длительное время. Попробуем придумать алгоритм, который упростит выполнение задания. Если у файлов разные размеры, то они не могут быть равными. Исходя из этого предположения можно в начале получить имена и размеры в всех сравниваемых файлов, затем отсортировать их по размеру и после этого сравнивать только файлы с одинаковым размером.

С помощью команды **type** опишем структуру для хранения необходимой информации. Для экономии памяти будем хранить не весь путь, а только имя файла. Вместо полного имени в поле `ow` `per` будет храниться индекс родительской директории в массиве директорий.

```
type finf
{
    str   name
    uint  size
    uint  owner
}
```

Вот какие глобальные переменные нам понадобятся:

```
global
{
    arr dirs of finf
    arr files of finf
    arr sizes of uint
    str output
}
```

dirs - массив обработанных директорий.

files - массив файлов.

sizes - массив индексов для `files`, который мы будем сортировать.

output - строка для вывода результатов.

Следующие две функции отвечают за добавление в массивы директорий и файлов.

```
func uint newdir( str name, uint owner )
func uint newfile( str name, uint size owner )
```

С их кодом можно познакомиться в исходнике. Там добавляется один элемент к массиву и заполняются его поля.

Функция **scanfolder** находит все директории и файлы по указанному пути. В случае нахождения директории, добавляется элемент в массив **dirs**, он становится родительским и заново вызывается функция сканирования. Если находим файл, то добавляем элемент в массив файлов. Для упрощения задачи я исключаю из рассмотрения файлы больше 4GB, условие `!cur.sizehi` служит как раз для этого.

```
func scanfolder( str wildcard, uint owner )
{
    ...
    if cur.attrib & $FILE_ATTRIBUTE_DIRECTORY
    {
        scanfolder( cur.fullname + "\\*.*", newdir( cur.name, owner ))
    }
    ...
}
```

Функция **scaninit** на основании начального пути сканирования формирует вызов **scanfolder**. Модифицируя эти функции вы можете использовать различные маски для поиска файлов, а так же указывать верхние и нижние границы размера сравниваемых файлов.

```
func scaninit( str folder )
{
    str wildcard

    folder.fdelslash()
```

```

    @"Scanning \( folder )\n"
    scanfolder( (wildcard = folder ).faddname( "*" .*" ), newdir( folder, 0 ))
}

```

Второй этап заключается в сортировке полученных данных. Конечно, можно сортировать массив `files`, но для ускорения лучше создать отдельный массив с индексами на `files` и сортировать его. Сортировка, как правило, требует много перемещений элементов и чем меньше размер одного элемента массива, тем лучше.

```

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 )
}

```

В функции `sortfiles` заполняем массив `sizes` индексами на `files`. Первоначально индекс совпадает с порядковым номером. После этого сортируем массив `sizes` используя функцию сортировки `sortsize`. Параметры `left` и `right` являются указателями на данные. Если бы у нас в массиве были структуры, то мы бы использовали их как объекты, но так как у нас массив `sizes` состоит из `uint`, то мы берем значение элемента с помощью `->uint`. `files[ index ].size` возвращает размер соответствующего файла. Сравнивая размеры возвращаем значение меньше, равное или больше нуля, если левый сравниваемый файл меньше, равен или больше правого.

Функции `getdir` и `getfile` восстанавливают полное имя файла по значению поля `ow` `peg`. `getdir` доходит рекурсивно до первой родительской директории и возвращаясь обратно создает полный путь.

```

func str getdir( uint id, str ret )
func str getfile( uint id, str ret )

```

Сейчас приступим к разбору главной функции сравнения. Проходим в цикле по всем отсортированным файлам начиная с минимального размера.

```

func compare
{
    ...
    fornum i, *sizes - 1
    {
        id = sizes[ i ]

        if !*files[ id ].name : continue

        found = 0
        next = sizes[ j = i + 1 ]

        while files[ id ].size == files[ next ].size
        {

```

В данном цикле мы сравниваем текущий файл со всеми следующими файлами имеющими такой же размер. При этом пропускаем уже определенные файлы-дубликаты. Сравнение производим с помощью функции `isequalfiles` из стандартной библиотеки. При обнаружении совпадения выводим информацию в строку `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 ) @"\\1"

        found = 1
        files[ next ].name.clear()
    }
    if ++j == *sizes : break
    next = sizes[ j ]
}

```

Этот фрагмент служит для вывода промежуточных результатов. `i & 0x3F` определяет вывод результата после каждого 64-го файла.

```

if i && !( i & 0x3F )
{
    @ "\\rApproved files: \%(i) Found the same files: \%(count)"
}
}
...
}

```

В функциях

```

func init
func search
func main<main

```

нет ничего сложного. Так как количество файлов и директорий может быть большим, то мы в функции `init` заранее резервируем место для некоторого начального количества элементов. И кроме этого создаем один пустой родительский элемент в массиве `dirs`, чтобы нумерация директорий начиналась с 1. Дело в том, что мы считаем поле `ow` не равным нулю в том случае, если выше родительской директории нет. То есть директория не может иметь нулевой индекс.

Упражнение 2

Написать программу нахождения одинаковых файлов на всех локальных жестких дисках компьютера.