Руководство TKScript

Разработчики:

Программное обеспечение:
ООО «КИГЛИ»
03150, г. Киев, ул. Боженко 11, оф. 507 т/ф (044) 227-87-23
E-Mail: info@uamap.net

Оглавление

Принцип работы модернизированной версии TKScript 9
Проекты 11
  Доступ к локальным файлам 11
  Исходные файлы скрипта (source files) 11
Модули и комментарии 12
  Архивы модуля 12
  Область видимости констант и классов 12
  Область видимости функций 12
  Модуль Main 13
  Программный код вне функций 13
  Комментарии 13
Идентификаторы, разделители, ключевые слова и константы 14
  ESC-последовательность 14
  Идентификаторы 14
  Разделители и операторы 15
  Зарезервированные ключевые слова 15
  Литералы 15
  Пользовательские константы 16
  Системные константы 17
Типы данных и объявление переменных 18
  Объявление переменных 19
  Область видимости 20
Указатели 21
  Типизированные указатели 21
  Нетипизированные указатели 21
  Удаление указателя 22
  Параметры указателей 22
  Управление памятью 22
Строки 23
  Буферизированные строки 23
  Оператор [ ] 23
  Файлы форматированного текста 25
Знаки операций (операторы) 26
  Операторные классы и приоритеты 26
  Операторы присваивания 26
  Унарные операторы 27
  Оператор с тремя операндами 28
  Потоковый оператор << 28
Знаки операций (операторы) встраиваемых классов 29
Выражения 31
  Константные и переменные выражения 31
  Выражения с массивами в качестве элементов 31
  Декремент/инкремент 31
  Логические выражения 32
  Выражения диапазона 32
  Выражение с тремя операндами “мини - if” 32
  Выражение new 33
  Выражение Value 33

Выражение списка {}.... 34
Обработка исключений.... 35
Операторы.... 36
Операторы if .. else.... 37
Оператор do .. while.... 37
Оператор while.... 38
Оператор switch.... 38
Оператор For.... 39
Оператор loop.... 40
Оператор foreach.... 40
Оператор prepare.... 41
Операторы clamp и wrap.... 42
Оператор use.... 43
Оператор define.... 43
Оператор function.... 43
Оператор class.... 43
Функции.... 44
Функции, определенные пользователем.... 44
Упреждающее объявление.... 44
Функции в других модулях.... 45
Вариации возвращаемых типов.... 45
TKS API функции.... 45
Классы.... 51
Классы cpp_factory.... 51
Члены класса.... 52
Классы скрипта в сравнении с API классами.... 52
Методы.... 52
Конструкторы и деструкторы TKScript.... 53
Конструкторы и деструкторы классов cpp_factory.... 54
Наследование и динамическое связывание (поздняя привязка).... 55
Массивы и списки массивов.... 57
Классы массивов C++.... 57
Оператор [ ].... 57
Изменение размеров массивов.... 57
Трекинг элементов и буфферизированные массивы.... 57
Списки массивов.... 58
Ассоциативные массивы (хэш-таблицы).... 58
Инициализаторы массивов.... 59
Многомерные массивы.... 59
Хэш хэшей.... 59
Пулы (накопители).... 60
Деревья.... 61
Стеки.... 61
Буффер.... 63
Классы массива.... 65
Конфигурация.... 66
Double.... 67
Envelope (диапазон).... 68
Событие.... 69
File.... 70

Float ... 73
Массив чисел с плавающей точкой (FloatArray) ... 74
Функция ... 76
Хэш-таблица ... 77
Целочисленный массив (IntArray) ... 79
Integer ... 82
Список (list) ... 83
Узел (элемент) списка ... 85
Массив объектов (ObjectArray) ... 87
PakFile ... 88
Массив указателей ... 90
Пул (Pool) ... 91
Скрипт ... 93
Стек ... 94
StdErrStream ... 95
StdInStream ... 97
StdOutStream ... 99
Поток ... 101
String ... 103
Строковый массив (StringArray) ... 105
Time ... 106
TKS ... 107
Узел дерева (TreeNode) ... 108
Value ... 110
Переменная ... 112
tks_engine_wrapper ... 113
type_t ... 115
member_func_t ... 116
member_var_t ... 117
static_var_t ... 118
static_func_t ... 119
func_param_t ... 120
type_agent_t ... 122
Различия между TK-скриптом и его модифицированной версией ... 123
testlocal.tks ... 124
fibonacci.tks ... 125
testfileio.tks ... 126
double.tks ... 135
md2.tks ... 154
tempscript.tks ... 162
tempscript2.tks ... 163
tgclso_hash.tks ... 164
tgclso_hash2.tks ... 165
list.tks ... 166
tgclso_lists.tks ... 167
pool.tks ... 169
preprocessor ... 170
testlang.tks ... 197
testxml.tks ... 199
testtime.tks ... 200

Как работать с этим руководством

Руководство TKScript состоит из отдельных глав. Как правило, каждая глава состоит из нескольких разделов.

Терминология

виртуальная машина выполнения скрипта	Механизм выполнения скрипта в рамках работы сервера приложений
TKX	Распространяемый, упакованный и gzip-ованный “выполняемый” формат, используемый TKScript.
TKP	Файл *TKS* проекта, описывающий отображение виртуальной файловой системы. Может быть скомпилирован в TKX-файл
TKS	Одиночные файл исходного кода TKScript
JIT	В этом случае компилятор, транслирующий файлы исходных кодов в собственный код, выполнит свои действия только после того, как приложение будет запущено. (JIT = Just In Time). В данной версии не поддерживается.

Принцип работы модернизированной версии TKScript

Сервер приложений

Любая программа, в которую встраивается TKScript, будет выступать в качестве сервера приложений, т.е. все скриптовые сценарии будут выполняться в рамках данной программы.

После старта программы будет инициализирован основной скриптовый модуль (main()). В рамках этого модуля будут запущены все скриптовые приложения.

Каждый скриптовый сценарий может быть привязан к конкретному действию. Такие сценарии после запуска сервера приложений будут хранить в памяти весь свой код вместе с данными и, при наступлении события, будут выполнены.

Упрощенная схема принципа выполнения скриптовых моделей изображена на рисунке 1.

Фабрика классов

Предназначена для регистрации классов, которые необходимы при выполнении того или иного скриптового сценария. Фабрика классов управляет регистрацией и продолжительностью жизни классов. Все объекты cpp_factory имеют тип.

cppf

Обеспечивает хранение библиотек дополнительных классов (таких, как vcl, win32 и др.) для возможности динамической загрузки и регистрации в Фабрике классов при выполнении скриптовых модулей.

cppf2tks

Это механизм выполнения скрипта, т.е. по сути сам TKScript. Дополнительная прослойка обеспечивает взаимодействие TKScript с Фабрикой классов.

debug_agent

Отладочный агент, осуществляющий связь между сервером приложений и внешними программами отладки (такими, как gdb.exe, make.exe, tks_console). Работает в рамках процесса работы сервера приложений.

gdb.exe

Внешняя программа-отладчик, которая при помощи внешней среды отладки (например, такой, как Dev-C++) отлаживает код скриптовых модулей.

make.exe

Внешняя программа, позволяющая собрать все скриптовые приложения после выполнения отладки.

tks_console

Внешняя программа, позволяющая отслеживать все сообщения, полученные в процессе отладки сценариев (например, сообщения об ошибках и т.п.).

Принцип выполнения скриптовых модулей

Рис 1

Проекты

TKS использует виртуальную файловую систему (VFS).

VFS только для чтения и все содержащиеся в ней файлы должны быть определены в файле проекта (TKP), который используется для отображения логических имен файлов в локальные. Файл проекта выполняется при тестировании проекта на протяжении разработки.

Доступ к локальным файлам

Для доступа к локальным файлам необходимо использовать класс File (см. раздел File). Класс PakFile (см. раздел PakFile) может быть использован для прозрачного отображения файлов локальной файловой системы. Для доступа с стандартному файловому потоку используйте классы StdInStream (см. раздел StdInStream), StdOutStream (см. раздел StdOutStream) (статические глобальные переменные).

Для того, что избежать проблем с различными видами представления путей в различных операционных системах, по возможности следует избегать использования абсолютного пути.

Символ “/”, который к счастью стал обычным даже для Microsoft Windows (традиционно использовавшего символ “\”) должен использоваться для разделения каталогов в представлении пути. Имена дисководов (их буквенное обозначение), а также точки монтирования не должны использоваться для переносимости.

Примечание:
С точки зрения безопасности, доступ к локальным файлам (кроме перечисленных в проекте) может быть полностью запрещен при использовании переключателя командной строки — nolocalfiles.

Исходные файлы скрипта (source files)

Скрипт располагается в директории (виртуальной), все файлы которой выполняются в указанном порядке, в зависимости от запуска приложения. Префикс "mod" подставляется вместо пути поиска текущего модуля.

Пожалуйста, обратите внимание на то, что исходные файлы скрипта (source files) будут включены в упакованный архив (для компилирования в двоичный код, при выполнении исходного кода, если исходные файлы приложения содержат компиляционные { /*...*/ } блоки).

Модули и комментарии

Модули используются для создания библиотек скриптов многоразового использования.

Модуль обеспечивает совокупность констант, переменных, функций и классов, которые обычно обслуживают определенные цели, например такие, как загрузка и отображение шрифтов (pixel-font.tks) или обеспечение средств отладки (debug.tks).

Вместе с C/C++ расширениями (плагинами) модули могут использоваться для определения интерфейса пользователя для плагина.

Примером может быть скриптовый менеджер голоса для аудио синтезатора, который использует исходные (родные) функции плагинов для генерации/воспроизведения колебаний.

Каждый модуль связан с TKS файлом исходного текста. Для обращения к определенным модулям им могут быть присвоены уникальные названия при помощи оператора `module`.

Пример:

```
module MMyModule;
```

Архивы модуля

Модуль обычно используется множеством приложений, которые предлагают использование общей директории. Путь к этой общей директории определяется также переменной среды:

```
TSK_MODULE_PATH
```

Директория проекта имеет более высокий приоритет, чем директория обычного модуля. Это означает, что если копия модуля создана и настроена для определенного приложения, а также размещена в директории проекта, она будет использоваться вместо архивного модуля.

Область видимости констант и классов

Классы и константы, декларируемые в одном модуле, также “видимы” во всех других модулях.

Область видимости функций

Функцию необходимо объявить до того, как она будет использована. Область видимости функции — модуль, в котором она была объявлена и реализована. Для вызова функций из других модулей, перед названием функции необходимо указать имя модуля, в котором эта функция была объявлена.

Например:

```
module Main;
MMyModule.MyFunction(); // assumes a MYModule.MyFunction function
```

Модуль Main

Модуль Main — это начальная точка приложения. Если модуль содержит функцию main(), эта функция будет вызвана только один раз, при успешной загрузке проекта.

Пример:

```
module Main;
function main()
{
trace "got " + Arguments.numElements + " argument(s).";
}
```

Если объявление модуля упущено, по умолчанию принимается объявление Main (кроме временных модулей, которые не имеют названий).

Программный код вне функций

Если отсутствует функция main(), выполнение приложения будет завершено после выполнения всех глобальных операторов. Это может быть использовано для инициализации массивов или выполнения различных предвычислительных операций (например, загрузки/формирования изображения).

Порядок выполнения определен в соответствии с порядком компиляции, указанном в файле проекта.

Комментарии

Для документационных целей рекомендуется указывать комментарии.

Ключевой символ “//” отмечает начало строки комментариев. Т.е. символы, следующие за данным ключевым словом, не интерпретируются.

Пример:

```
// ---- file : mymodule.tks
// ---- author: Bastian Spiegel
```

Ключевые символы /* и */ используются для исключения целого параграфа из интерпретации. В отличии от “//” эти комментарии могут распространяться на несколько строк подряд.

Не рекомендуется использовать вложенные комментарии, т.к. это путает режимы подсветки (и фактически не имеет смысла).

Например:

```
for(int i=0; /* это вложенный комментарий */ i++; i<10)
{
/* ..../*выполнение чего-то полезного*/... */
}
```

Идентификаторы, разделители, ключевые слова и константы

В основном, TKS использует обычную 7-битную ASCII кодировку символов, поддержка UTF-8 (unicode) пока не возможна (хотя, возможно, будет реализована в будущем).

ESC-последовательность

Для включения неотображаемых управляющих символов (кодов) ASCII (таких как перевод строки) также как и специальных символов, таких как “ ‘ ” или “ \” в строках или целых литералах, основанных на символах, поддерживается следующая последовательность символов:

Последовательность	Описание
\\	Обратный слеш
\’	Апостроф
\”	Двойная кавычка
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\f	Новая страница
\b	Возврат на один символ
\a	Звуковой сигнал
\e	ESC-символ (десятичное число 27, восьмеричные 033)

Последняя последовательность отмечает начало последовательностей ANSI. Наиболее подробный список последовательностей ANSI можно найти здесь http://astronomy.swin.edu.au/~pbourke/dataformats/ansi.html.

Примечание:
Обычный эмулятор консоли Windows (command.com) и, следовательно, все оболочки основанные на нем (такие как CygWin) к сожалению, не поддерживают последовательности (стандарта) ANSI (CygWin способен эмулировать ANSI последовательности, но Вам будет необходимо компилировать собственную CygWin версию tsk.exe)

Идентификаторы

Идентификаторы используются для определения уникальных имен переменным, классам, функциям и константам. Идентификаторы чувствительны к регистру символов, это означает, что, например, MyVariable и myvariable — различные названия.

Первый символ идентификатора должен быть буквой [a-zA-Z] или символом подчеркивания “_”. Идентификатор не должен содержать разделителей или операторов. Зарезервированные ключевые слова также не могут быть идентификаторами. Длина фактически не ограничена (память, 24бит), но она не должна превосходить 128 символов.

Разделители и операторы

Следующие символы разделяют слова, интерпретируемые индивидуально:

```
= > < == <= >= != , ! && || ++ -- + - * / & | ^ % << >> += -= *= /=
&= |= ^= %= <<= >>= ( ) { } [ ] ;
```

Значение этой, на первый взгляд загадочной, последовательности символов будет описано детально далее, но в основном они сохранили стандартные значения (подобно языкам JavaScript и/или C++).

Зарезервированные ключевые слова

Указанная ниже последовательность символов не должна использоваться в качестве идентификаторов:

```
abs acos argb asin asm break case clamp class compile cos default
#define define deref dtrace do else enum exp false float for func-
tion
frac if int ln loop module null Pointer prepare return rgb rnd
round sin String switch tag tan tcchar tcfloat tcobject tcstring
this trace true while wrap use
```

Эти ключевые слова используются для:

* вызова функций (abs, acos, argb, asin, clamp, cos, deref, dtrace, exp, frac, ln, rgb, rnd, round, sin, tan, tcchar, tcfloat, tcobject, tcstring, this, trace, wrap)
* управляющих структур/потоков данных (asm, class, compile, do, else, for, function, if, loop, module, prepare, return, switch, while, use)
* как системные константы (null, true, false, default).

Литералы

Следующие формы представления могут использоваться для описание константных значений:

10	десятичное целое число
10.25	(десятичное) число с плавающей точкой
$fedcba98	шестнадцатеричное целое число
#fedcba98	шестнадцатеричное целое число, использующее цветовой формат HTML (#aarrggbb, a=alpha, r=red, g=green, b=blue)
0b1101001	двоичное целое число
'!'	отдельный ASCII символ

'\n'	отдельная управляющая последовательность ASCII символов (например перевод строки)
"a \'string\'\n"	последовательность одного или нескольких ASCII символов
"\e[2J"	управляющая последовательность ASCII символов (например очистить экран)
String s= "hello, \n" "world"	многострочная символьная последовательность

Пример:

```
trace "hello, world.";

int i=0b1101001;
float f=10.25;

String s="a \'string\'

String s_clrscr="\e[2J";

String s_text="text*text*text*text*"
              "text*text*text*text*"
              "text*text*text*text*"
              "text*text*text*text*"
              "text*text*text*text*"
              "text*text*text*text*";
int c='!';
int c2=s_text[9];
```

Пользовательские константы

Для улучшения читаемости, рекомендуется назначать уникальные имена для часто используемых констант так, чтоб физическое значение константы было определено только в одном месте исходного кода (обычно, в заголовке).
Для этих целей предназначены выражения define (*define) и/или enum (enumeration).

Название константы должно начинаться с буквы [a-zA-Z] или знака подчеркивания “_”. Остальные символы могут быть буквами, знаком подчеркивания или цифрами [0-9].

Пример:

```
#define NUMLOOPS 42
loop(NUMLOOPS) { /* ... */ }
```

Пример:

```
#define DEG2RAD 0.00872664625997
trace "147.5 Grad im Bogenma?entsprechen "+(147.5*DEG2RAD);
```

Пример:

```
#define AUTHOR "Bastian Spiegel "
```

```
trace "This software was developed by " + AUTHOR ;
```

Пример:

```
enum { RED, GREEN, BLUE }; // => RED==0, GREEN==1, BLUE==2
```

Пример:

```
enum { RED, GREEN=4, BLUE }; // => RED==0, GREEN==4, BLUE==5
```

Константа всегда замещается только одним значением; программируемый макропроцессор, как в C, не доступен.

Системные константы

Ниже подан список предопределенных системных констант:

1PI	единица, деленная на число Пи
1SQRT2	единица, деленная на корень квадратный двойки
2PI	число Пи, умноженное на 2 (6.2..)
BIG_ENDIAN	прямой порядок байт, начиная со старшего
default	0 или 0.0 или false
E	математическая константа E
false	0
SEEK_BEG	потоковый режим поиска
SEEK_CUR	потоковый режим поиска
SEEK_CUR	потоковый режим поиска
IOS_OUT	потоковый режим ввода/вывода
IOS_IN	потоковый режим ввода/вывода
IOS_INOUT	потоковый режим ввода/вывода
null	0 объектный указатель
LITTLE_ENDIAN	обратный порядок байт, начиная с младшего
LN10	математическая константа ln(10)
LOG10	математическая константа log(10)
PI	математическая константа (число Пи)
PI2	число Пи, деленное на двойку (1.5..)
RAND_MAX	максимальное случайное значение, возвращаемое функцией
SQRT2	математическая константа корень квадратный 2
true	1

Типы данных и объявление переменных

Основные типы данных TKS:

* целое число (int)
* число с плавающей точкой (float)
* последовательность символов (String)
* (Object) указатель .

- Указатель — это ссылка на экземпляр класса C++ или скриптового класса и, таким образом, ключ к сложным структурам данных
- Строки (Strings) — это специальные объектные указатели

По причине простоты никакое различие между целыми числами со знаком и без знака не сделано (все целые числа со знаком).

TKS необходимо 32-разрядная схема адресации. Затраты памяти для хранения целого числа равны одному указателю (т.е. 4 байтам для 32-разрядной архитектуры и 8 байтам для 64-разрядной)

Язык скрипта автоматически конвертирует тип данных String, int и float.

Пример:

```
float f="1.23";
int i=3.14;
String s=42;
```

Каждый из ниже приведенных типов данных представлен классом C++; они всегда доступны так, как встроены в скрипт.

* Object - *базовый класс для всех API классов; создавать экземпляры этого класса нельзя.*
* Buffer (см. Буфер на стр. 63.) - *массив байт, полученный из потока*
* Class - *скриптовый класс, определенный пользователем*
* ClassArray (см. Классы массива на стр. 65.) - *массив классов, определенных пользователем*
* Envelope (см. Envelope (диапазон) на стр. 68.) - *массив чисел с плавающей точкой (время/ парные значения), поддерживающий интерполяцию*
* Event (см. Событие на стр. 69.) - *строка, основанная на значениях времени, в основном строка меток времени*. Этот класс считается устаревшим.
* File (см. File на стр. 70.) - *используется для доступа к файловой системе*
* Float (см. Float на стр. 73.) - *представление числа с плавающей точкой*
* FloatArray (см. Массив чисел с плавающей точкой (FloatArray) на стр. 74.) - *массив чисел с плавающей точкой (float)*
* Function (см. Функция на стр. 76.) - *ссылка на функцию.* Этот класс считается устаревшим.
* HashTable (см. Хэш-таблица на стр. 77.) - *ассоциативный массив (хэш-таблица)*
* IntArray (см. Целочисленный массив (IntArray) на стр. 79.) - *массив целых чисел (int)*
* Integer (см. Integer на стр. 82.) - *объектное представление целого числа (int)*
* ListNode (см. Узел (элемент) списка на стр. 85.) - *один узел (элемент) списка (или заголовок списка), полученный из Value*
* ObjectArray (см. Массив объектов (ObjectArray) на стр. 87.) - *массив объектов (однородных)*
* Pointer - *ссылка на (?неизвестный?) объект*
* PointerArray (см. Массив указателей на стр. 90.) - *массив объектов (возможно, “смешанных”)*

- Pool (см. Пул (Pool) на стр. 91.) - *неупорядоченный массив объектов подходящий для оптимизации управления памятью*
- Script (см. Скрипт на стр. 93.) - *динамически создаваемый TKScript модуль.* Этот класс считается устаревшим.
- Stack (см. Стек на стр. 94.) - *структура памяти LIFO (Last In First Out) для вызовов рекурсивных функций*
- Stream - *базовый класс для Files и Buffers*
- String (см. Поток на стр. 101.) - *буферизированная последовательность символов (имеет специальные fastpaths)*
- StringArray (см. Строковый массив (StringArray) на стр. 105.) - *массив последовательностей символов (строк)*
- Time (см. Time на стр. 106.) - *описывает точку во времени*
- TreeNode (см. Узел дерева (TreeNode) на стр. 108.) - *один узел дерева (или заголовок дерева), полученный из Value*
- Value (см. Value на стр. 110.) - *целочисленное, с плавающей точкой, строковое или объектное значение, представленное в виде объекта.*
- Variable (см. Переменная на стр. 112.) - *используются для доступа к переменным в динамически откомпилированных модулях скрипта*. Этот класс считается устаревшим.

Специальные классы Integer и Float, обеспечивают объект-оболочку для скаляров int и float, который приходится как нельзя кстати, когда нужно создать ссылку на эти базовые/основные типы данных.

Специальные классы Value и Float обеспечивают объект-оболочку для динамически введенных значений.

Объявление переменных

Пример:

```
vardecl ::= <type> <identifierlist> ;
type ::= int | float | String | Pointer | <Class>
identifierlist ::= <identifier>[=<expression>] |
<identifierlist>,<identifier>[=<expression>>]
identifier ::= <idchar1> | <idchar1>,<idcharseq>
identifierlist ::= <idchar> | <idcharseq><idchar>
idchar1 ::= [a-z,A-Z,_]
idchar ::= [a-z,A-Z,_,0-9]
expression ::= ...arbitrary expression, e.g. "(1+2)*3"...
```

Переменные могут быть объявлены везде, где ожидается выражение.

Пример:

```
int i=42;
float x,y=2.3,z; // объявление нескольких переменных одного типа
String s,t="hello, world.",u;
```

Пример:

```
for(int i=0; i<10; i++) trace i; // смотри выше
```

Пример:

```
function MyFunction() {
//объявление локальной переменной, область видимости которой в
//пределах функции
  String s="hello, world.";
  trace s;
}
```

Пример:

```
class MyClass {
  MyMethod();
}
MyClass::MyMethod {
  // ---- смотри выше ----
  String s="hello, world.";
  trace s;
}
```

Область видимости

Область видимости глобальных переменных — текущий модуль. Глобальные переменные других модулей доступны при указании перед название переменной имени модуля, в котором они объявлены (например, MMyModule.myvariable).

Переменные, объявленные в функции, видимы только в пределах функции, в которой они были объявлены. Переменные функции всегда локальны по умолчанию. Вы можете использовать атрибут “static” для объявления статичной переменной.

Смотри также testlocal.tks (см. на стр. 124.) , fibonacci.tks (см. fibonacci.tks на стр. 125.)

Указатели

Указатели используются для избежания траты времени и памяти на ресурсоемкие копии объектов, например, при передаче параметров функции или методу.

Множество объектных переменных могут указывать на один и тот же объект, но TKS во время выполнения заботится об управлении памятью. Т.е. одновременно только один указатель может быть владельцем объекта и таким образом иметь право удалять объект из памяти.

Специальный тип переменной Pointer представляет не типизированный объектный указатель (который технически указывает на объект YAC_Object C++)

Указатели не включают информацию о типе объекта, на который указывают; эта информация скорей получается от самого объекта, который находится в памяти и не является нулевым (null).

Присвоение несовместимого объектного типа данных типизированной объектной переменной заставит виртуальную машину выполнения скрипта прервать работу или вызвать ошибку, при использовании переменной в операторах или выражениях, которым необходим доступ к объекту (например, вызов метода или доступ к членам объекта)

Типизированные указатели

Пример:

```
// Назначение указателя, t2 и t будут указывать на одну и
// ту же область памяти после этого назначения.
// t будет владельцем, t2 удален перед назначением
Time t, t2 <= t;
t2.now();
```

Пример:

```
// Назначение указателя, нетипизированный
// указатель, возвращенный выражением "new",
// привязан к переменной (нетипизированной),
// предыдущее содержание которой удалено перед назначением
  Time t <= new Time;
  t.now();
```

Не типизированные указатели

Виртуальный тип данных Pointer используется для хранения ссылки на параметры объектов

Пример:

```
// нетипизированный указатель, возвращенный выражением "new"
// привязан к нетипизированной переменной
Pointer p <= new Time;
// теперь этот указатель разименован и привязан к переменной t
Time t<=deref p; t.now();
// переменная t становится владельцем объекта после назначения
// указателя (<=). С тех пор, как "p" разименован, следующее
```

```
// выражение не затрагивает "t":
p<=null;
```

Пример:

```
String s="hello, world.";
Pointer p <= s; // назначение нетипизированной объектной переменной
String t <= p;  // назначение типизированной объектной переменной
trace "t="+t;   // Вывод содержимого строки для отладки
trace "p="+p;   // Вывод адреса памяти и типа для отладки
```

Удаление указателя

Указатель удаляется присвоением 0 (null):

Пример:

```
String s="hello, world.";
s<=null; // удаление указателя; объект удален из памяти
// т.к. 's' владелец объекта
trace "s="+s; // теперь указатель <void> (0)
```

Параметры указателей

Параметры для указателей всегда передаются по ссылке (*call-by-reference*).

Параметры в запросах C/C++ всегда передаются по ссылке, т.е информация управления памятью (флаг deleteme) теряется.

Пример:

```
function MyFunction(String _s) {
// прямое изменение значения параметра
    _s.append(", world.");
    return _s;
}
trace MyFunction("Hello");
```

Управление памятью

Указатель <= назначает выражения только несвязанным изменяемым объектам и в любом случае создает не редактируемую ссылку данному объекту.

Выражения **tobject()** и **tcstring()** могут использоваться для создания копий не редактируемых объектов. Хранить таким образом безопаснее, чем в контейнерах (например, переменных, массивах, хэш-таблицах)

Значение выражения **#()** может использоваться для создания новых изменяемых объектов .

Список выражений **{}** может использоваться для создания новых списков изменяемых объектов.

Строки

Строка состоит из произвольного (ограниченного памятью) количества байт, закодированных по стандарту ASCII. Специальное значение “0” обозначает конец последовательности символов (ASCIIZ).
В настоящее время не поддерживаются строки в формате UNICODE, в будущих версиях это будет исправлено при помощи кодирования UTF-8.

Более подробное описание строкового класса (см. String на стр. 103.).

Буферизированные строки

`String` автоматически выделяет память при необходимости.

Пример:

```
String sbuf;
sbuf.alloc(1024);
sbuf.empty(); // очистка строки
sbuf="hello, ";
sbuf.append("world.");// не требуется выделение
// дополнительной памяти
```

Буфер объекта String может также указывать на константу, неизменяемую последовательность символов; при операции, вносящей изменения, сначала будет автоматически создана копия.

Пример:

```
String s<="hello"; // установка буфера на константную строку
//следующая операция автоматически создаст копию
//которая будет удалена после того, как выражение будет выполнено
trace s+", world";
```

Детальное описание всех поддерживаемых строковых операций можно найти по ссылке API; листинг представлен здесь:

operator !=(), operator &(), operator &&(), operator >(), operator >=(), operator <(), operator <<(), operator <=(), operator +(), operator ==(), operator [](), operator ||(), alloc(), append(), copy(), empty(), endsWith(), fixLength(), free(), freeStack(), getBufferLength(), getc(), getLength(), getWord(), indexOf(), insert(), isBlank(), lastIndexOf(), load(), loadLocal(), parseXML(), patternMatch(), print(), putc(), replace(), saveLocal(), split(), startsWith(), substring(), toLower(), toUpper(), trim(), words()

Оператор [ ]

Оператор [ ] используется для доступа к отдельному символу строки. При этом, необходимо помнить, что невозможно получить доступ к символу вне ASCII формата (т.е. 0..index<string.lenght).

Пример:

```
String s<="hello, world.";
trace "the 6th char of the string is:\'"+tcchar(s[5])+"\'.";
```

Функция **tcchar()** используется для конвертирования ASCII кода символа (s[5]=44=',') в пригодный для печати формат (2 char long, включая ASCIIZ) String (“,”).

Пример:

```
// Загружает локальный текстовый файл (true=ASCII режим,
// переводы каретки '\r' удалены) и разбивает его на строки
String t,s; s.loadLocal("test.txt", true);
s.split('\n');
foreach t in s trace "line="+t;
s.freeStack();

foreach t in s.splitChar('\n') trace "line="+t;
```

Пример:

```
// Разбивает на слова, не принимая во внимания вложенные строки:
String t,s<;="abc def ghi jkl \". . .\""; s.words(false);
foreach t in s trace "t="+t;
s.freeStack();

foreach t in s.splitSpace(false) trace "t="+t;
```

Пример:

```
// Разбивает на слова, принимая во внимание вложенные строки
// Эти строки интерпретируются, как одно слово:
String t,s<;="abc def ghi jkl \". . .\"";
s.words(true);
foreach t in s trace "t="+t;
s.freeStack();

foreach t in s.splitSpace(true) trace "t="+t;
```

Пример:

```
String s<="one two three"; s.words(1);
trace "the 2. word is "+s.getWord(1);
s.freeStack();

trace "the 3. word is "+(s.splitSpace(0)[2]);
```

Список объекта String после использования должен быть освобожден вручную. Иначе последовательные вызовы split() или words() не будут иметь желаемого результата.

Файлы форматированного текста

Зачастую, текстовые файлы часто сохраняются в машинно-читаемой форме с учетом автоматической обработку. XML (Extensible Markup Language) используется, как стандартный формат для этих целей.

XML определяет основную структуру и синтаксис текстового файла. Обычно, каждый XML файл начинается со ссылки на DTD (Document Type Description), который синтаксический анализатор (парсер) XML должен использовать для проверки правильности структурной организации документа. DTD описывает набор элементов, их атрибуты и возможные значения, а также каким образом эти элементы могут быть вложены для формирования сложных структур данных.

TKS использует упрощенный синтаксический анализатор (парсер) XML, который выполняет только основную проверку синтаксиса. Проверка потока текста между элементами и правильность DTD не поддерживаются.

Метод String.parseXML() делит строку на Л/П дерево; Левые элементы ссылаются на узлы того же самого уровня; правые узлы ведут к поддеревьям структуры элемента. Списки атрибута элементов преобразованы в хэш-таблицу (ассоциативный массив). Структура элемента документа преобразована в TreeNodes, хранящего элемент хэш-таблицы, для того, чтоб сделать ее доступной из скриптов.

Пример:

```
String s<="<test><body><text value=\" \'. . .\' <test>\"/></
body></test>";
TreeNode t<=s.parseXML();
TreeNode u<=t.right;
trace "u.name="+u.name; u<=u.right;
HashTable r<=u.object;
trace "text=\""+r["value"]+"\"";
```

Знаки операций (операторы) TKScript

Операторы играют важную роль в операциях присваивания и выражениях; сложные выражения создаются при помощи комбинирования двух или более выражений с использованием операторов.

Операторные классы и приоритеты

* Арифметические и битовые операторы (+ - * / % | ^ & << >> ~)
* Логические операторы (! || && ^^)
* Операторы отношения (== <= != >= < >)
* После- и До- инкремент/декремент (++ --)
* Операторы присваивания (= *= /= %= &= += -= |= ^= >>= <<=)

- Высокий приоритет : ! ~ (унарное отрицание)
- Средний приоритет : * / % &
- Низкий приоритет : + - | ^ || && ^^ == <= != >= < > >> <<

Пример:

```
// Вначале вычисляется выражение 3*2, затем от 20 отнимается
// результат (6), добавляется 10. Таким образом, результат = 24
int i = 10 + 20 - 3 * 2;
```

Для самостоятельного создания подвыражений, например для задания приоритетов, в выражение могут быть добавлены скобки ():

Пример:

```
//Вначале вычисляется выражение (20-3), затем результат умножается
//на 2 и в конце добавляется 10. Таким образом, результат = 44.
int i = 10 + (20-3) * 2;
```

Операторы присваивания

Операторы присваивания используются для хранения в памяти значения результата выражения, например глобальные и локальные переменные, массивы, элементы хэш-таблиц и экземпляры классов.

Если символу “=” предшествует арифметическое выражение, то предыдущее значение, хранящееся в памяти, может быть принято во внимание при вычислении его нового значения.

Следовательно, знаки операций присваивания — краткие формы для следующих операций присваивания:

```
l *= <выражение>; равносильно l = l * <выражение>;
l /= <выражение>; равносильно l = l / <выражение>;
l %= <выражение>; равносильно l = l % <выражение>;
l &= <выражение>; равносильно l = l & <выражение>;
l += <выражение>; равносильно l = l + <выражение>;
l -= <выражение>; равносильно l = l - <выражение>;
l |= <выражение>; равносильно l = l | <выражение>;
```

```
l ^= <выражение>; равносильно l = l ^ <выражение>;
l >>= <выражение>; равносильно l = l >> <выражение>;
l <<= <выражение>; равносильно l = l << <выражение>;
```

Замечание:

Целью присваивания не должно быть сложное выражение, a.b.c.d=e; например не допускается. В настоящее время возможны следующие цели присвоения:

```
myvar <assignop> <expr>;
MyModule.myvar <assignop> <expr>;
myclassinstance.member <assignop> <expr>;
myarrayvar[] <assignop> <expr>;
myhashvar[] <assignop> <expr>;
myapiclassinstance.mymember <assignop> <expr>;
myclassinstance.mymember <assignop> <expr>;
myclassinstance.myarrayvar[] <assignop> <expr>;
myclassinstance.myhashvar[] <assignop> <expr>;
```

Порядок вычисления подвыражений с одинаковым приоритетом (вычисления) не определяется, а остаётся на усмотрение используемой выполняющей среды (исполняющего модуля), например интерпретатора.

Вычисление по цепочке не поддерживается, т.е. все подвыражения вычисляются независимо от того, мог ли результат целого выражения быть определен уже после вычисления его первого подвыражения.

Пример:

```
if(i && j && k) { /* ... */ }
```

Запись может быть оптимизирована следующим образом:

Пример:

```
if i if j if k { /* ... */ }
```

Унарные операторы

Унарные операторы имеют наивысший вычислительный приоритет. В отличии от C++ или Java, здесь нет унарного оператора приведения типов. Эта функциональность скорее обеспечивается встроенными запросами (tcint (), tcfloat (), tcchar (), tcstring ()).

Пример:

```
int i = -1; // отрицание, результат -1.
```

Пример:

```
int i = ~1; // поразрядное отрицание, результат -2
            //(0xFFFFFFFE в 32битовой архитектуре)
```

Пример:

```
int i = !1; // логическое отрицание, результат 0
```

Оператор с тремя операндами

Этот оператор используется для выбора между двумя подвыражениями в зависимости от результата условного выражения. Пожалуйста, более детально см. п. 31.

Пример:

```
String s = rnd(2) ? "true" : "false";
```

Потоковый оператор <<

Оператор << используется для преобразования объектов из/в потоковые объекты ввода/вывода.

Пример:

```
String s="hello, world.";

Buffer b; b.size=512; // обработка буфера(Поток)

b << s; // преобразование строки в буфер (поток)

s="..."; trace s; // сброс строки

b.offset=0; // смещение строки сброса

s << b; // обратное преобразование строки из буфера (потока)

trace "s="+s;
```

Пример:

Пожалуйста, смотри testfileio.tks.html (см. testfileio.tks на стр. 126.)

Знаки операций встраиваемых классов

Для классов, встраиваемых в TKScript, поддерживаются следующие знаки операторы:

operator= + - * / % == != < > <= >= & | && || ^ ! ~ <=

Все операторы поддерживаются скриптом не явно. Напрямую они не вызываются.

operator= для неинициализированных объектов (для которых не вызывается конструктор). Для внешних классов (кроме `any`), при использовании этого оператора вызывается конструктор, передающий значение, стоящее справа от символа “=”. А для класса `any` будет создана копия объекта справа от оператора “=”.

Пример:

```
double d=3.4; //d будет инициализировано значением 3.4
```

Для остальных внешних классов, определяемых пользователем, если справа стоит такой же тип и этот тип указатель — конструктор вызываться не буде. Указатель будет скопирован.

Пример:

```
function create_form()
{
  TForm fm;
   fm.pConstructor(mainForm);
   fm.Width=400;
   fm.Height=300;
   return fm;
}

function main()
{
  TForm fm=create_form();
}
 // Позволяет скопировать указатель
```

Если операторы определены вне класса, то они не будут выполняться.

Пример:

```
class a_t
{
  ...
  a_t operator+ (const a_t & rhs);
  ...
}
a_t operator+ (const a_t & rhs);
   a_t a,b,c;
c=a+b;
// Вызов оператора вне класса выполняться не будет.
```

Оператор <=

Пример:

```
any   a=f();
TForm a=f();
// Вызов конструктора объекта a. Этот объект инициализируется
// значением, расположенным с правой стороны от знака "=".

any   a<=f();
// Если с левой стороны указывается any — объект,
// расположенный справа от оператора, просто копируется.
// Если с левой стороны указывается внешний тип — для
// объекта выполняется оператор статического
// приведения типов (static_cast).
```

Выражения

Выражения используются для комбинирования одного, двух или трех значений (подвыражений) для получения результата с использованием операторов. Наиболее простой пример выражения — константа (значение).

Константные и переменные выражения

Пример:

```
int i=42; // константное выражение ("42")
int j=i;  // переменное выражение ("i")
```

Выражения с массивами в качестве элементов

С технической точки зрения результатом каждого выражения может быть только одно значение. Так, как это выражение может являться ссылкой на массив или другую сложную структуру данных, в выражении также могут эффективно производиться вычисления с массивами значений.

Пример:

```
// результат выражения — указатель на массив целых чисел (int),
// который опять составлен из произвольного числа значений
function ReturnN {
   return [1,2,3,4,5,6,7,8];
}
trace "element[7]="+ReturnN()[7];
```

Декремент/инкремент

Переменное значение также может содержать оператор до/после инкремент/декремент (++v, v++, --v, v--), который используется для изменения переменной без вычисления и присваивания. Переменная инкрементируется или декрементируется в зависимости от положения оператора, до или после того как выполнено более позднее вычисление. Работает как с целыми числами, так и с числами с плавающей точкой (+1.0, -1.0).

Пример:

```
int i=42, j=i++; // j равно 42, i стало 43.
```

Пример:

```
int i=42, j=--i; // j равно 41, i стало 41.
```

Это также может использоваться для членов модулей и классов:

Пример:

```
int j = MMyModule.myvariable++;
```

Пример:

```
int j = --myobj.mymember;
```

Эти операторы также могут использоваться в выражениях:

Пример:

```
int i=42; i++; i--;
```

Логические выражения

Логические выражения вычисляются, как истина (!=0) или ложь (==0). Обычно логические выражения используются для проверки условия выхода их цикла или для сигнализирования состояния, используя флаги (или маски). Литералы true и false могут быть использованы для замещения значений 1 или 0.

Пример:

```
if(true) trace "it's true";
```

Выражения диапазона

Диапазонные выражения тестируют попадание целого числа или числа с плавающей точкой в указанный диапазон.

Пример:

```
int i=5; i= (1 < i < 10);
float f=5.23; i= (1.1 < f < 10.10);
```

Выражение с тремя операндами “мини - if”

Это выражение проверяет условие в пределах выражения и выбирает между двумя альтернативными результирующими выражениями, в зависимости от результата проведенной проверки, какое из них может быть истинным или ложным.

Пример:

```
// значение выражения с тремя операндами определено
// значением переменной i
int i=rnd(10),j=rnd(10),k=i>5?(i+j):(j-i);
```

Вышеуказанный пример равносилен следующей программе в терминах функциональных возможностей:

```
int i=rnd(10), j=rnd(10), k;
if(j>5)
k=i+j;
else
k=j-i;
```

Выражение new

Выражение new используется для создания новых объектов.

В качестве параметра, передающегося в это выражение, используется название скриптового класса, определенного пользователем или название C++ API класса.

Пример:

```
function NewString {
String s<=new String; // размещение новой строки
return deref s;       // реализация привязки к переменной s
}
String mynewstring<=NewString();
```

Пример:

```
class AbstractFactory {
newObject() {/*trace "Abstract::newObject, never called";*/}
}
class ConcreteStringFactory : AbstractFactory {
newObject() { return new String; }
}
class ConcreteFloatFactory : AbstractFactory {
newObject() { return new Float; }
}
```

Выражение Value #()

Это выражение используется для того, чтобы зафиксировать возвращаемое значение и тип выражения, а также представить его в виде объекта класса Value. Пожалуйста, смотрите также Value (см. Value на стр. 110.) в API Документации.

Пример:

```
    function ReturnSomething(int _what) {
      switch(_what)
      {
        case 0:
           return 0;
        case 1:
           return 42;
        case 2:
           return PI;
        case 3:
           return new Time;
        case 4:
           return "hello, world.";
      }
    }
Value vi<=#(ReturnSomething(1)); // -возвращает целое
Value vf<=#(ReturnSomething(2));//-возвращает число с плавающей
точкой
```

```
Value vo<=#(ReturnSomething(3)); // -возвращает объект
Value vs<=#(ReturnSomething(4)); // -возвращает строку
trace "vi.value="+vi.value;
trace "vf.value="+vf.value;
trace "vo.value="+vo.value;
trace "vs.value="+vs.value;
```

Выражение списка {}

Выражения Списка используются для создания двунаправленного списка произвольной длины в процессе работы. Пожалуйста, также смотрите ListNode (см. Узел (элемент) списка на стр. 85.) документации API.

Пример:

```
//
// ----    file: list.tks
// ----  author: bastian spiegel
// ---- created: 14-Jan-2004
//

ListNode l<={1,2.2,"hallo"};
ListNode c;
c<=l.tail; c.appendValue(
c<=l.tail; c.appendValue(
c<=l.tail; c.appendValue(

Value v;
foreach v in {1,2.2,"hi",tcobject(42),
    trace "v.type="+v.type+" v.string="+v.stringValue;
trace l;
l.print();

ListNode l2,l3;

function evalListExpr() {
    trace "--------------------------------";
    l2<={rnd(1),rnd(2),rnd(3),
    l3<=l2.copy;
    c<=l3.tail; c.appendValue(
    trace l3;
    l3.print();
}

loop(1)
   evalListExpr();
```

Обработка исключений

Синтаксис скрипта не позволяет отслеживать возникновение исключений в ходе выполнения программы.

Источниками ошибок может являться как сам TKScript, так и различные внешние исключения. При возникновении исключения работа TKScript будет завершена, а сообщение о произошедшей ошибке будет отображено в консоли. Также в окне консоли отображаются сообщения о выполнении программы (см. рис.1).

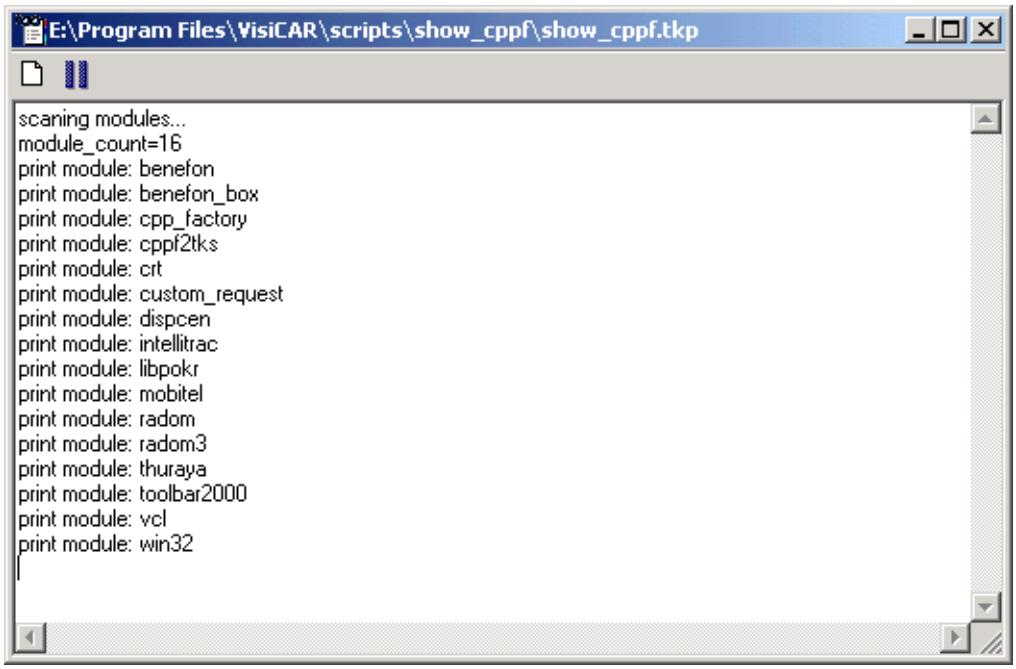

Рис1

Операторы

Операторы используются для объявления переменных, констант, классов, функций и модулей. Оператор не возвращает значений, а скорей используются как для сохранения результирующих значений содержащегося в нём выражения (-ий) в памяти, так и в качестве параметра для управляющей конструкции. TKS поддерживает следующие конструкции:

```
do..while
  if..else
  for
  foreach
  loop
  switch..case
  while
```

Другие поддерживаемые операторы:

```
clamp
   class
   define
   enum
   function
   module
   prepare
   use
   wrap

   <вызовы встроенных функций>
   <вызовы метода класса>
   <переменная/массив/хэш/присваивание члену>
   <объявление переменных>
   <вызовы функций, определенных пользователем>
```

Операторы часто группируются в блоки (последовательности операторов), которые заключаются в фигурные скобки {}.

Для каждого исходного файла, верхний уровень последовательности операторов создан неявно, что делает возможным написание TKS скрипта, состоящего, например, всего лишь из единственного оператора.

Операторы вне тела функции или метода выполняются после успешной компиляции всех модулей, определенных в файле проекта. Порядок выполнения точно такой, как порядок возникновения файлов в проекте. Этот механизм может быть использован для инициализации объектов, размещения массивов и/или инициализации переменных.

Пример:

```
trace "hello, world.";
```

— представляет собой полную работоспособную TKS программу

В отличии от C, блоки операторов не могут непосредственно содержать другие блоки операторов:

Пример:

```
if(i==42) { { /* ..это не работает.. */ } }
```

Пример:

```
if(i==42) { if(j==1) { /* ..это работает.. */ } }
```

Операторы if .. else

Этот оператор используется для выполнения перехода по условию, в зависимости от результата логического (условного) выражения.

Если логическое выражение равносильно истине (!=0), тогда оператор после **if** будет выполнен, иначе управляющий поток выполнит переход по оператору **else**, если возможно.

Пример:

```
if(i==42) j=1;
```

Пример:

```
if(i==42) j=1; else j=2;
```

Пример:

```
if(i==42) { j=1; k=2; } else { j=2; k=3; }
```

Примечание:
Круглые скобки () не требуются; На самом деле, они часть выражения:

Пример:

```
if i==42 j=1;
```

Пример:

```
if i==42 j=1; else j=2;
```

Пример:

```
if i==42 { j=1; k=2; } else { j=2; k=3; }
```

Оператор do .. while

Этот оператор используется для циклического выполнения оператора (последовательности операторов) до тех пор, пока логическое выражение после **while** не станет ложным, т.е. false (0).

Пример:

```
int i=0; do i++; while (i<10);
```

Пример:

```
int i=0,j=0; do { i++; j++; } while (i+j)<10;
```

Логическое условие будет проверяться каждый раз, после завершения итерации цикла, т.е. тело цикла do ... while будет выполнено хотя бы раз.

Подобно оператору **if**, круглые скобки () вокруг условного выражения являются дополнительными и фактически являются частью выражения.

Оператор while

Этот оператор используется для повторения последовательности операторов до тех пор, пока логическое выражение (после while), будет оставаться истинным, т.е. true (!=0).

Пример:

```
int i=0; while(i<10) i++;
```

Пример:

```
int i=0,j=0; while(i+j)<10 { i++; j++; }
```

Оператор switch

Этот оператор используется для выбора одного из нескольких альтернативных путей выполнения программы. Если условие найдено, ассоциированная с ним последовательность операторов будет выполнена. Под условие default попадают все остальные условия, которые не перечислены в списке явных.

Каждая последовательность операторов должна завершаться ключевым словом **break**, иначе последовательность условий следующего условия будет выполнена, не смотря на то, была ли найдена соответствующая метка в этой последовательности или нет.

Пример:

```
int i=rnd(10);
    switch(i) {
        case 1:
                trace "i is 1.";
                break;
        case 2:
                trace "i is 2.";
                break;
        case 3:
        case 4:
                trace "i is 3 or 4";
                break;
        default:
                trace "i is neither 1,2,3 nor 4.";
                break;
    }
```

В отличии от C, в качестве условных выражений позволяется использовать строки и числа с плавающей точкой; более того, условие выбора (case) не обязательно должно быть константой, оно может содержать даже вызов функции.

Пример:

```
function MyFunction() {
        return 10;
    }

    String s=Arguments[0];

    switch(s) {
        case "-h":
        case "--help":
            /* ... */
            break;
        case MyFunction(): // case 10
            /* ... */
            break;
}
```

Оператор For

Этот оператор используется для повторения тела цикла до тех пор, пока условие цикла (логическое выражение) остается истинным. Таким образом, режим работы цикла **for** подобен циклу **while**, хотя может быть обеспечена дополнительная инициализация и оператор модификации. Пустой оператор (;) используется для избежания инициализации; точка с запятой после оператора модификации не требуется.

Оператор инициализации вызывается перед стартом первой итерации цикла, оператор модификации вызывается каждый раз, когда итерация заканчивается. Перед выполнением каждой итерации выполняется проверка условие цикла. Если условие истинно — тело цикла выполняется и начинается выполнение следующей итерации.

Пример:

```
for(int i=0; i<10; i++) trace "i="+i;
```

* trace "i="+i; — тело цикла
* int i=0; — инициализация цикла
* i<10; — условие цикла
* i++ — оператор модификации
* i — счетчик цикла

Пример:

```
int i=0;
for(;i<10;) trace "i="+i++; //избежание инициализации +
модификационный оператор
```

Пример:

```
int n = 16; //<i>http://www.bagley.org/~doug/shootout/nestedloop</
i>
    int x = 0;
    for (int a=0; a<n; a++)
        for (int b=0; b<n; b++)
            for (int c=0; c<n; c++)
                for (int d=0; d<n; d++)
                    for (int e=0; e<n; e++)
                        for (int f=0; f<n; f++)
                            x++;
```

Оператор loop

Этот оператор напоминает оператор **for** и оператор **while**. Фактически, этот оператор представляет собой часто используемый частный случай оператора **for** и **while**.

Выражение, которое следует за ключевым словом **loop,** обозначает количество итераций. Это целочисленное выражение вычисляется только один раз, перед началом первой итерации. Нет возможности получить доступ к внутреннему счетчику. Счетчик будет декрементироваться до тех пор, пока не станет = 0.

Скобки вокруг целочисленного выражения, точно так же, как в операторах do...while, while и for, являются частью выражения.

Пример:

```
loop 10 trace "loop.";
```

Пример:

```
int i=0;
loop(10) trace "i="+i++;
```

Пример:

```
int n = 16; // http://www.bagley.org/~doug/shootout/nestedloop
    int x = 0;
    loop(n)
        loop(n)
            loop(n)
                loop(n)
                    loop(n)
                        loop(n)
                            x++;
```

Оператор foreach

Этот оператор используется для итерирования “контейнерных” объектов, например таких, как списки строк, массивы, хэш-таблицы или пулы (pool). Тело цикла будет выполнено для каждого элемента данного “контейнерного” объекта.

Для выхода из цикла foreach, переменной цикла может быть присвоено значение -1 в том случае, если переменная цикла — целое число. Если переменная цикла является указателем — значение 0 (null) будет сигнализировать виртуальной машине выполнения скрипта о необходимости выхода из цикла.

Пример:

```
String t,s="one \"two and a half\" three";
foreach t in s.splitSpace(true) trace "t="+t;
```

Пример:

```
int i; foreach i in [1,2,4,7,9,11] trace "i="+i;
```

Пример:

```
int i=0; //http://www.bagley.org/~doug/shootout/bench/wordfreq/
String s,k;

    HashTable words; words.alloc(20000);
    s.loadLocal("wordfreq.txt", true);
    foreach k in s.splitSpace(false) {
        k.toLower();
        words[k]=tcint(words[k])+1;
    }

    StringArray sta; sta.alloc(words.numElements);
    IntArray ita;    ita.alloc(words.numElements);
    i=0;
    foreach k in words {
        ita[i]=i;
          Integer io; io.value=words[k]; sta[i]=io.printf("%7d")+"
"+k;          i++;
    }

    sta.sortByValue(ita, false);
    i=words.numElements;
    loop(--i)
        trace sta[ita[i--]];

    trace "words.numElements="+words.numElements;
```

Оператор prepare

Этот оператор используется для старта одноразовой инициализации в контексте функции. Последовательность операторов, следующая за ключевым словом **prepare**, будет выполнена только один раз на протяжении работы приложения.

Пример:

```
function FLookUp(int _i) {
        IntArray lut;
        prepare {
```

```
            lut.alloc(512);
            int i=0;
            loop(256) lut.add(i++);
            loop(256) lut.add(255);
        }
         return lut[_i];
    }
    trace FLookUp(184); trace FLookUp(384);
```

Операторы clamp и wrap

Эти операторы используются для ограничения значения переменной (целой, с плавающей точкой или Объектной ссылкой) в пределах определенного диапазона.

Оператор **clamp** осуществлен при помощи следующего С-кода:

```
if(var->pointer.float_val<min.value.float_val)
   var->pointer.float_val=min.value.float_val;
else
  if(var->pointer.float_val>max.value.float_val)
    var->pointer.float_val=max.value.float_val;
```

Оператор **wrap** осуществлен при помощи следующего С-кода:

```
if(var->pointer.float_val<min.value.float_val)
var->pointer.float_val      =      (var->pointer.float_val      -
min.value.float_val) + max.value.float_val;
else if(var->pointer.float_val>=max.value.float_val)
var->pointer.float_val      =      min.value.float_val+      (var-
>pointer.float_val- max.value.float_val);
```

Пример:

```
float f=32; clamp f 0 20; // f это 20
```

Пример:

```
int i=32; wrap i 0 20; // i это 12
```

В случае, если данная переменная типа Объект — ограничения должны быть того же типа, что и переменная. Следовательно, этот механизм может быть использован для проверки корректности диапазона сложных типов данных, импортированных с использованием интерфейса плагина C++.

Пример:

```
use tkopengl;
Vector v<=vector(2,3,4);
wrap v vector(0,0,0) vector(1,2,3); // v is (1,1,1)
```

Оператор define

Оператор, использующийся для определения константных значений. Он анализируется при сканировании исходного кода. Генерируется таблица соответствия, используемая при синтаксическом анализе текста (после сканирования исходного текста).

Пожалуйста, также (см. Пользовательские константы на стр. 16.) .

Оператор function

Этот оператор используется для декларации новых функций. Для более подробной информации (см. Функция на стр. 76.).

Оператор class

Этот оператор используется для декларации новых классов. Для более подробной информации (см. Классы TKScript на стр. 51.).

Функции

Функции используются для группировки часто используемых последовательностей операторов, для избежания избыточности кода. Эти операторные блоки могут быть параметризированы до 255(скриптовых) или 16(собственных) аргументов. Т.о. функции поддерживают модуляризацию программы и, более того, помогают разбить проблему на подпроблемы (Разделяй и властвуй!)

Функции, определенные пользователем

Значения, возвращаемые функциями, определенными пользователем, динамичны и могут изменяться, в зависимости от установленных параметров. Каждая функция может вернуть не более одного значения. Если необходимо вернуть множество значений — могут использоваться такие объекты, как хэш-таблицы или массивы.

Пример:

```
function Deg2Rad(float _degrees) {
return _degrees*2PI/360.0;
}

trace Deg2Rad(90);
```

Пример:

```
function SplitFloat(float _f) {
float ffrac=frac(_f);
return [_f-ffrac, ffrac];
}

float f;
foreach f in SplitFloat(1.23)
trace f;
```

Упреждающее объявление

В том случае, если выполнение функции не следует сразу за ее объявлением — такое объявление будем называть **упреждающим объявлением**. Список параметров функции может не повторяться при выполнении заранее объявленной функции.

Скорость выполнения функции может быть слегка увеличена, так, как вызов функции может быть осуществлен напрямую, т.е. без динамического связывания.

Пример:

```
function HTMLSpan(String _s); // "Упреждающее объявление"
    HTMLSpan("test");
function HTMLSpan {
    // Выполнение, список параметров не повторяется
    return ""+_s+"";  }
```

Пример:

```
function PrintHello(MyClass _m) {
    trace "hello"+_m.string;
}
class MyClass {
    String string;
}
PrintHello(m);
```

Функции в других модулях

К функциям в других модулях можно выполнить обращение, указав при вызове имя соответствующего модуля, в котором эти функции объявлены.

Пример:

```
MMyModule.MyFunction();
```

Вариации возвращаемых типов функций TKScript

Возвращаемый тип функции не фиксированный и может зависеть от текущих аргументов.

Пример:

```
HashTable values<=

function vreturn(String _arg) {
     if(values.exists(_arg))
        return values[_arg];
     else
        return "err: no such element "+_arg+".";
}
```

TKS API функции

Ниже перечислены функции, которые обеспечиваются TKS основной API.

int	2n(int)	возвращает n-ю степень двойки (два в n-й степени)
int|float	abs(int|float)	возвращает абсолютное (положительное) значение
float	acos(float)	возвращает арккосинус
int	argb(int, int, int, int)	возвращает пакетированное 32-битное значение цвета
float	asin(float)	возвращает арксинус
float	cos(int|float)	возвращает косинус

float	deg(float)	преобразовывает радианы в градусы
float	frac(float)	возвращает дробную часть числа
String	getenv(String)	возвращает значение переменной среды
float	mathAbsMaxf(float, float)	возвращает большее из двух абсолютных значений
int	mathAbsMaxi(int, int)	возвращает большее из двух абсолютных значений
float	mathAbsMinf(float, float)	возвращает меньшее из двух абсолютных значений
int	mathAbsMini(int, int)	возвращает меньшее из двух абсолютных значений
float	mathMaxf(float, float)	возвращает большее из двух значений
int	mathMaxi(int, int)	возвращает большее из двух значений
float	mathMinf(float, float)	возвращает меньшее из двух значений
int	mathMini(int, int)	возвращает меньшее из двух чисел
float	mathPowerf(float, float)	возвращает значение в степени b
int	mathPoweri(int, int)	возвращает значение в степени b
	print(String)	вывод строки в STDOUT. Добавляя новую строку в случае необходимости.
	putenv(String, String)	задание значений переменной среды
float	rad(float)	преобразование градусов в радианы
int	rgb(int, int, int)	возвращает пакетированное 32-битное значение цвета (alpha=255)
int|float	rnd(int|float)	возвращает случайное число из заданного диапазона
float	round(float)	округление к ближайшему целому числу
float	sin(int|float)	возвращает синус
float	sqrt(int|float)	возвращает корень квадратный
	stderr(String)	вывод строки в STDERR
	stdout(String)	вывод строки в STDOUT
int	system(String)	выполнение системной команды*
float	tan(int|float)	возвращает тангенс
String	tcchar(int)	преобразование целого числа в один строковый символ строки в формате ASCII
float	tcfloat(int|float|String)	преобразовывает значение в формат числа с плавающей точкой
int	tcint(int|float|String)	преобразовывает значение в формат целого числа
Object	tcobject(int|float|String)	преобразовывает значение в формат объекта. Создает копию объекта, если он только для чтения.

String	tcstring(int|float|String)	преобразовывает значение в формат строки. Создает копию строки, если строка только для чтения.
	trace(String)	вывод строки в консоль отладки (по умолчанию STDOUT). Добавляет новую строку, если необходимо.

Замечания:

* Для улучшения скорости, большинство используемых функций являются встроенными
* В целях безопасности, вызов system() работает только в том случае, если константа DX_SYSEXEC определена на этапе компиляции. Иначе будет возвращено -1. (*)

Встроенные статические функции

void registerSelfTask()	Регистрирует вызывающий скрипт в пуле скриптов. Это позволяет сохранить его состояние.
void unregisterSelfTask()	Удаляет скрипт из пула. Вызов из события пока не возможен. Если скрипт был выполнен и вышел, то удалить его нельзя. Он останется в памяти до закрытия программы.
refFrom	Возвращает ссылку на переданный ей объект (ссылка в понятиях C++). В качестве параметра этой функции передается любой объект фабрики классов (cpp_factory). См. Пример1 после таблицы.
crefFrom	Возвращает константную ссылку на переданный ей объект. См. Пример1 после таблицы.
ptrFrom	Возвращает ссылку на переданный в функцию объект класса `any`. См. Пример1 после таблицы.
cptrFrom	Возвращает константную ссылку на переданный в функцию объект класса `any`. См. Пример1 после таблицы.
setEventHandler	Устанавливает обработчик события. Эта функция может быть вызвана, как член класса или глобальная. Ей может быть передано 2 или 3 параметра. См. Пример2 после таблицы.
static_cast	Статическое приведение типов.
getStaticVariable	Возвращает значение переменной. В качестве параметра передается строка с название переменной.
callFunction	Вызов глобальной функции. В качестве параметра необходимо передать полное имя функции, а также необходимые параметры (переменное количество).
getProperty	Получение свойства члена класса. В качестве параметра передается имя члена.

setProperty	Установка свойства члена класса. В качестве параметров передается имя члена класса, а также значение, которое необходимо установить этому свойству.
Constructor	Создает объект.
CastedConstructor	Создает объект типизированных классов или класса `any`.
pConstructor	Создает указатель на объект.
pCastedConstructor	Создает указатель на объект типизированного класса или класса `any`.
Destructor	Разрушает объект.
typeid(<выражение>)	Служит для динамического определения типа выражения. Возвращает объект типа type_agent_t (см. на стр. 56.), характеризующий этот тип.

Пример1:

Сравнение получения ссылок и указателей в TKScript и C++.

Описание	TKScript	C++
Получение ссылки	any rd = refFrom(d);	double & rd = d;
Получение константной ссылки	any crd = crefFrom(d);	const double & crd = d;
Получение указателя	any pd = ptrFrom(d);	double * pd = &d;
Получение указателя на константное значение	any cpd = ptrFrom(d);	const double * cpd = &d

Пример2:

```
bottom.setEventHandler("OnClick","on_btn_click");
// используется для функций внутри скрипта



class TButton
{
...
  __property TNotifyEvent OnClick={...};
...
};
setEventHandler(bottom.OnClick,"on_btn_click");

// При таком объявлении выражении bottom.OnClick вернет
// значение OnClick, а не ссылку. Т.е. не будет корректно работать
```

```
class TButton
{
  TNotifyEvent OnClick;
};
setEventHandler(bottom.OnClick,"on_btn_click");

// При таком объявлении выражение bottom.OnClick вернет
// ссылку на OnClick. Корректная работа



setEventHandler(bottom.OnClick,form,"OnBtnClick");
// используется для внешних функций
```

Вызов функций

Для вызова функции необходимо соблюдение синтаксиса TKScript. При этом функция будет вызвана корректно.
Внутренние функции скрипта не регистрируются.

При использовании точечной нотации может быть доступен вызов внешних функций, а также свойства для значений, ссылок и указателей первого уровня.

Пример:

Доступны:

TForm TForm & TForm *
const TForm & const TForm *

Не доступны:

TForm ** и т.п.

Механизм передачи параметров во внешние функции

Рассмотрим сравнение передачи параметров в функции в TKScript и C++:

TKScript	C++
int	int
float	float
Integer	int &
Float	float &
Double	double &
String	const char*

T (cpp_factory)	T &

Где T — все остальные типы

Механизм получения значений из внешних функций

Рассмотрим сравнение получения значений из внешних функций в TKScript и C++

C++	TKScript
int	int
float	float
const char*	String
char*	String
T	any

Где T — все остальные типы

Классы

Классы TKScript

Класс TKScript — это структура данных, состоящая из целых чисел, чисел с плавающей точкой и объектов, т.е. экземпляров C++ или определенных пользователем скриптовых классов.

Классы используются для расширения приложения с помощью новых типов данных. Подобно таким основным типам данных, как int и float, прежде необходимо создать экземпляры классов, перед тем, как они могут использоваться. Такие экземпляры называются объектами.

Объекты

В каждом скрипте явно создан объект **cppf** типа **tks_engine_wrapper** (см. раздел tks_engine_wrapper на стр. 113.). Этот объект может использоваться в служебных целях для управления скриптом (загрузка библиотек, запуск файлов, обновление данных о функциях и переменных и т.п.).

Классы cpp_factory

Классы, регистрирующиеся через Фабрику классов, имеют общий механизм диспетчеризации функций и методов. Этот механизм достаточно отличается от механизма классов, встроенных в TKScript и является более гибким. Внешние классы регистрируются с учетом ограничений, накладываемых синтаксисом TKScript.

Выделяют 3 разновидности классов — это **Встроенные типы C++** (double, assigne и т.д); **Классы, определенные пользователем** и **Класс any** (любые возвращаемые значения, присваивания и т.д.).

Следует отметить, что класс `any` значительно уступает по скорости другим классам. Так происходит потому, что обращение к функциям, переменным и т.п. класса `any` осуществляется на этапе выполнения скриптового модуля, тогда как у других классов — на этапе компиляции. Токим образом, обращение к функциям и перемнным выполняться не один раз, что замедляет работу программы.

Рекомендуется как можно реже использовать класс `any` (не только из-за проиграша в скорости, но и для улучшения читаемости).

Классы Фабрики классов поддерживают перегруженные функции (т.е. функции с одинаковыми именами, но различным количеством параметров). Для этих функций возможно задание строки параметров.

В классах Фабрики классов конструкторы можно вызывать несколько раз. При повторном вызове конструктора старый объект будет удален и создан новый с указанными параметрами.

При вызове метода для еще не созданного объекта будет вызвано исключение и программа завершит свою работу.

Члены класса

Член класса — это переменная, определенная в области имени класса. Каждый раз при создании экземпляра класса, память для его членов выделяется также. Члены класса видимы в методах класса; также они доступны из внешнего кода при использовании синтаксиса “objname.member”. Объявление члена класса может также содержать инициализаторы (например: int i=42; или int ia2[16];)

Пример:

```
class MyClass
{
  int        i;
  float      f;
  String     s;
  int      ia[];  //тоже, что и целочисленный массив ia;.
  //Размер массива, заданный при инициализации=0.

  int        ia2[16]; //Размер, заданный при инициализации=16
  FloatArray fa;      //тоже, что и float fa[]; . Размер,
  //заданный при инициализации=0.

  float      fa2[32]; //Размер, заданный при инициализации=32
  HashTable  ht;      // Размер, заданный при инициализации=57
}
  MyClass obj;

  trace "obj.f="+obj.f;
```

Классы TKScript в сравнении с API классами

В то время, как скриптовые классы объявлены определяемым пользователем скриптом, C++ API классы либо встроены в механизм скрипта (основные классы API) либо импортируются через интерфейс плагина YAC.

Ниже перечисленные классы — это часть ядра TKS API:

Buffer, ClassArray, Configuration, Envelope, Event, File, Float, FloatArray, HashTable, IntArray, Integer, ListNode, ObjectArray, PointerArray, Pool, Script, Stack, Stream, String, StringArray, StringIterator, Time, TKS, TreeNode, Value, Variable.

Пожалуйста, смотрите документацию API для изучения подробного описания всех основных классов.

Методы

Методы используются для объединения часто используемых операторов, которые работают с объектами. Метод похож на функцию, за исключением использования ключевого слова “function” и объявления в пределах класса.

Пример:

```
class MyClass {
          myMethod() { trace "myMethod called."; }
      }
      MyClass c;
      c.myMethod();
```

Метод может быть реализован непосредственно в пределах объявления класса (как показано в вышеприведенном примере). Также можно использовать упреждающее объявление, в том случае, если реализация будет осуществляться позже (рекомендуется для больших операторных блоков):

Пример:

```
class MyClass {
     int i;
     float f;

     myMethod(int _i, float _f); // упреждающее объявление
 }

 MyClass::myMethod { // выполнение
     i=_i;
     f=_f;
     trace "myMethod called. i="+i+" f="+f;
 }

 MyClass c;
 c.myMethod();
```

Методы суппер классов (т.е. основных классов) могут вызываться, используя основное имя класса:

Пример:

```
 class BaseClass {
     sayHello() { stdout "hi!
 }
 class MyClass : BaseClass {
myMethod() { stdout "myMethod says" + BaseClass::sayHello(); }
 }

 MyClass c;
 c.myMethod();
```

Конструкторы и деструкторы классов TKScript

Для инициализации элементов класса после создания экземпляра объекта, класс может обеспечить подходящие конструкторы и деструкторы.

Конструктор — это метод, который носит то же самое имя, что и класс; это также применимо к деструктору с той разницей, что его имени предшествует символ ~.

В TKS в конструкторы и деструкторы параметры передаваться не могут; Вместо этого можно использовать подходящие методы init() или exit().

Конструкторы вызываются, когда объект создан. Если класс был получен из базового класса, сначала вызывается конструктор базового класса.

При удалении объекта — деструкторы вызываются в обратном порядке, т.е. деструктор базового класса вызывается в последнюю очередь.

Пример:

```
class Telephone {
     float fVolume; // член (свойство)

     Telephone() { // конструктор
         trace "constructing Telephone.";
         fVolume=1.0;
     }

     ~Telephone(); // упреждающее объявление деструктора

     ring() { stdout "Telephone::ring


//---- определение интерфейса пользователя для установки звука---
    setVolume(float _f) { fVolume=_f; }
 }

 Telephone::~Telephone { // выполнение деструктора
     trace "deleting Telephone.";
 }

//----создание экземпляра объекта Telephone, неявный
//вызов конструктора----
 Telephone t;

 t.fVolume=0.74; // установка свойств fVolume

// ---- установка свойств с использованием метода, определяемого
// интерфейсом пользователя ----
 t.setVolume(0.74);

 t.ring(); // вызов метода
 t<=null;  // явно удаляет объект, неявный вызов деструктора
```

Конструкторы и деструкторы классов cpp_factory

Для классов, регистрирующихся через Фабрику классов, применимы несколько другие конструкторы.

Для **Встроенных типов C++** для вызова конструктора используются такие ключевые слова, как `Constructor` или `pConstructor`.
При использование первого значения — конструктор создаст объект. При выходе этого объекта за пределы области видимости, объект будет разрушен.
Во втором случае — конструктор создаст указатель на объект, при выходе за пределы видимости указателя — объект не будет разрушен.

Для **классов, определенных пользователем** и **класса any** используются такие ключевые слова, как `CastedConstructor` или `pCastedConstructor`. Они действуют аналогично конструкторам для **Встроенных типов C++**.

При использовании ключевого слова **Destructor** — разрушается объект или указатель (если в значении храниться указатель).

Наследование и динамическое связывание

В целях усовершенствования, специализации и расширения, класс может быть получен из одного или более базовых классов. Полученный класс наследует всех членов и методы того основного класса (классов), из которого он получен. TKS поддерживает множественное наследование, т.е. каждый класс может иметь до 64 базовых классов.

Методы базового класса (классов) могут быть перегружены с новой реализацией, до тех пор, пока сигнатура их параметров (т.е. их количество и тип) остается неизменной.

При вызове перегруженного метода, механизм динамического связывания сначала находит, какой именно метод нужно вызвать. Этот выбор зависит от типа самого объекта, а не ссылки (на этот объект), т.к. объекты, имеющие один или несколько базовых классов, могут быть привязаны к ссылке на любой базовый класс. В C++ этот процесс называется виртуальным вызовом.

Пример:

```
class BaseClass {
        exec() { trace "BaseClass::exec"; }
}
class ExtClass : BaseClass {
        exec() { trace "ExtClass::exec"; }
}
BaseClass bc <= new ExtClass; // приведение к базовому классу
// ---- механизм динамического связывания выбирает
//метод ExtClass::exec ----
bc.exec();
```

Пример:

```
class C1 {
    int i;
}
class C2 {
    float f;
}
class C3 : C1, C2 {
    C3() {
```

```
            i=42; f=10;
            trace "C3::C3()";
        }
    }
    C3 c;
```

Пример:

```
    class CClass {
    // определение "интерфейса".
    outputHTML() { return "*ill*"; }
   }
   class CLink : CClass {
            String target;
            String label;
            getLabel() { return label; }
            outputHTML()
            {
                   return "<a href=\""+target+"\">"+getLabel()+"</a>";
            }
    }
    class CImage : CClass {
         String img_source;
         String img_alt;
         outputHTML() {
           return "<img alt=\""+img_alt+"\" src=\""+img_source+"\">";
         }
    }
    class CImageLink : CImage, CLink {
            getLabel() { return CImage::outputHTML(); }
            outputHTML() { return CLink::outputHTML(); }
    }
    CImageLink il;
    il.target="http://tkscript.de";
    il.img_source="images/test.png";
    il.img_alt="test";
    trace il.outputHTML();
```

Массивы и списки массивов

Массив — это последовательность полей целых чисел, чисел с плавающей точкой или объектов.

Классы массивов C++

С технической точки зрения, массив также является экземпляром класса API C++, т.е. специализированного класса массива для элементов различных типов.

```
int -> IntArray
float -> FloatArray
String -> StringArray
Pointer -> PointerArray, ObjectArray
<uclass> -> ClassArray
```

Оператор [ ]

Каждый из вышеперечисленных классов поддерживает оператор [], который используется для считывания или модификации отдельных элементов массива.

Пример:

```
int ia[10];   // создание целочисленного массива с 10 элементами
ia[2]=3;      // задание элемента
trace ia[2];  // печать элемента массива
```

Изменение размеров массивов

Методы alloc(), realloc() и free() используются для определения, изменения размеров массива или удаления его элементов.

Если размер массива изменен с использованием метода realloc(), его старое содержание сохраняется, если это возможно. Если новый размер массива меньше, чем количество ранее используемых элементов, то последние элементы массива в количестве, равном (oldNumElements-newMaxElements), будут отброшены.

Пример:

```
IntArray ia; ia.alloc(10); // создание целочисленного массива с 10
элементами
ia[5]=42;      // установка элемента 6. Также изменение numElements
ia.realloc(20); // изменение размеров массива
trace ia[5];    // считывание элемента
```

Трекинг элементов и буферизированные массивы

Каждый массив отслеживает как фактическое количество используемых элементов массива (numElements), так и максимальное количество доступных элементов (maxElements).

После создания массива, свойство maxElements инициализируется значением, переданным методу alloc(), numElements будет установлено в 0. Метод empty() используется для того, чтобы сбросить numElements фактически не удаляя элементы.

Списки массивов

Один элемент может быть добавлен или удален при использовании таких методов, как add(), insert(), delete(). Этот режим работы напоминает списки, отсюда следует термин Список массива. В отличии от настоящих списков, максимальная длина ограничена числом выделенных элементов.

Пример:

```
float fa[100]; loop(fa.maxElements) fa.add( rnd(1.0) );
fa.realloc(200);
loop(fa.MaxElements-fa.numElements) fa.add( rnd(1.0) );
trace "fa.numElements=" + fa.numElements;
fa.empty();
trace "empty()";
trace "fa.numElements =" + fa.numElements
 + "fa.maxElements =" + fa.maxElements;
```

Ассоциативные массивы (хэш-таблицы)

В отличие от обычных массивов, хэш-таблицы последовательно не индексированы. Вместо этого, поименованные ключи (строки) используются для вычисления фактического (внутреннего) индекса, используя так называемую хэш-функцию.

Кроме того, каждый элемент хэш-таблицы может иметь индивидуальный тип, который определяется присвоенным ему результатом выражения. Если значение этого результата представлено нетипизированным указателем, то явное присвоение указателя не требуется для связывания этого указателя с элементом хэш-таблицы.

Пример:

```
HashTable ht; ht.alloc(113);
ht["myint"]=42;
ht["myfloat"]=1.23;
ht["mystr"]="hello, world."; // привязка константного указателя
ht["myobj"]=new IntArray; //привязка неконстантного указателя к
хэш-таблице
if ht.exists("myobj")
        trace "found slot \"myobj\"";

 ht.delete("myobj");
```

Пример:

```
int n=150; // http://www.bagley.org/~doug/shootout/bench/hash2
HashTable hash1; hash1.alloc(10000);
HashTable hash2; hash2.alloc(10000);
int i=0;
for(i=0; i<10000; i++)
```

```
    hash1["foo_" + i] = i;
String k;
loop(n)
    foreach k in hash1
        hash2[k] += hash1[k];
```

Инициализаторы массивов

Значения элементов массива могут быть установлены либо присвоением по одному значению, либо с использованием нижеприведенных выражений инициализации массива:

Пример:

```
StringArray str <= [ "one", "two", "three"];
FloatArray  flt <= [ 1.1, 2.2, 3.3 ];
IntArray    ia  <= [1, 2, 3];
HashTable   ht  <= #["myint"=42, "myfloat"=1.23,
                     "mystr"="hello, world.", "myobj"=new IntArray];
```

Первый элемент инициализатора массива определяет тип создающегося массива. Инициализаторы хэш-таблиц предваряются символом #.

Замечание:
Возвращаемое значение (т.е. объект массива) инициализатора массива — константа, т.е. только для чтения!

Однако, выражение инициализации вызывают не один раз, списки выражения вычисляются снова и элементы массива обновляются.

Многомерные массивы

В TKS, многомерные массивы созданы с использованием массивов указателей на массивы целых чисел, массивы с плавающей точкой или массивы указателей.

Пример:

```
int i;
PointerArray mda <= [ [1,2,3], [4,5,6], [7,8,9] ];

IntArray ia<= mda[0];
i=ia[2];
trace "ia[2]=mda[0][2]="+i;

i=mda[2][1];
trace "mda[2][1]="+i;
```

Хэш хэшей

Для создания печально известного “хэша хэшей”, указатели массивов не требуются. Класс “хэш-таблица” (см. Хэш-таблица на стр. 77.) может хранить произвольные типы данных (включая указатели), поэтому объявление и обращение к многомерным хэш-таблицам является непосредственным (прямым):

Пример:

```
HashTable mht <= #[
   "ht1"=#["one"=1, "two"=2.2],
   "ht2"=#["three"=3, "four"=4.4]
];

HashTable ht <= mht["ht2"];

trace mht["ht1"]["two"];
```

Замечание:
Подобно инициализаторам массивов, `#[]` инициализатор хэш-таблиц возвращает константные значения. Однако, если выражение инициализатора вызывается более чем один раз, список выражений вычисляется снова и элементы хэш таблицы обновляются.

Пулы (накопители)

Пул (pool) — это контейнер для однородных объектов, к которым не обращаются последовательно, а скорее используя так называемые имена-идентификаторы.

Основное преимущество остается в таких быстрых методах, как qAlloc() и qFree(), последовательный перебор для пустых элементов не требуется даже после многих вложенных запросов alloc/free.

Пример:

```
Pool p; p.template=String; p.alloc(10);
  int nameid; loop(10) {
  nameid=p.qAlloc();
  p[nameid]="test"+rnd(1024);
}
foreach nameid in p trace "p["+nameid+"]="+p[nameid];
```

Пример:

```
class MyPoolEntry {
         String name;
 }

 function main() {
         Pool pool;
         int i;
         int id;
         MyPoolEntry ce;
         String s;

         pool.template=MyPoolEntry;
         pool.alloc(32);
         loop(10)
         {
                 id=pool.qAlloc();
```

```
            ce<=pool[id];
            ce.name="poolentry_"+id;
        }
        i=0;
        foreach id in pool
        {
            trace("foreach qFree i=="+i);
            if(++i<5)
            {
                pool.qFree(id);
            }
            else
                id=-1; // конец foreach
    }
    foreach id in pool
    {
            ce<=pool[id];
            s<=ce.name;
            trace("foreach entry \""+s+"\"

    }
    pool.free();
    }
```

Деревья

Дерево — это, в основном, специальная форма двунаправленного списка, состоящая из Узлов дерева (TreeNodes), которые связаны со своим предшественником (родителем), с узлами на том же уровне (“left”) или отклоняющимися поддеревьями (“right”).

Кроме того, каждый узел дерева может хранить объектный указатель, чтобы сослаться на произвольные данные.

Стеки

TKS не поддерживает вызов рекурсивных функций, так как и глобальные и локальные переменные статичны, т.е. размещены в памяти по фиксированному адресу. Так, как рекурсия делает программирование более удобным в некоторых случаях, например при записи препроцессора HTML, может быть использована следующая уловка:

Перемещая все локальные переменные в определенный пользователем класс (stackframe), который при каждом вызове функции создает свой экземпляр, разрешается проблема невозможности использования переменных более одного раза одновременно. Объекты stack-frame-а управляются классом Stac, который обеспечивает необходимые методы push() и pop().

Пример:

```
class FibStackFrame {
    int i;
}

 Stack stFib;
```

```
stFib.init(FibStackFrame, 8);

function Fib(int _i) {
        int r;
        if(_i==0)
                return 0;
        else
                if(_i==1)
                        return 1;
                else
                {
            // ---- получение новой функции stackframe ----
                        FibStackFrame st<=stFib.push(); st.i=_i;

            // Fib - запросы перезаписывают переменную функции _i
                        r= Fib(st.i-1) + Fib(st.i-2);

           // восстановление старой функции stackframe
                        st<=stFib.pop();

                        return r;
                }
}

trace("fib(9)="+Fib(9));
```

Буфер

* универсальный бинарный буфер
* поддерживает оператор преобразования в последовательную форму <<
* прозрачная конверсия порядка байт
* может быть доступно во внешних C++ классах (например, использоваться для организации TCP/IP сети)

Наследование

```
Object -> Stream -> Buffer
```

(см. Поток на стр. 101.)

Свойства

size - (чтение-запись), сохраняет текущий размер буфера в байтах
byteOrder - BIG_ENDIAN или LITTLE_ENDIAN
offset - текущее смещение в байтах (интерфейс потока)

Методы

	fillZero()	быстрое заполнение нулевыми элементами, с использованием memset
	free()	удаление содержимого буфера, сбрасывание размера и смещения.
int	getByteOrder()	возвращает текущий порядок байт (BIG_ENDIAN/ LITTLE_ENDIAN)
int	getOffset()	возвращает текущее смещение чтения/записи
int	getSize()	возвращает размер буфера
String	getString(int _off, int _len)	извлекает строку из буфера. Если _len - 0, символы копируются, пока ASCIIZ не будет найден
int	gunzip(Buffer _src, int _off, int _clen, int _ulen)	сжимает буфер. корректировка размера, сброс смещения.
int	gzip(Buffer _src, int _off, int _ulen, int _level)	распаковывает буфер. корректировка размера, сброс смещения.
int	peekI8(int _off)	считывание байт из _off
int	peekI16(int _off)	считывание 16 битного числа из _off. Изменение порядка байт, при необходимости.
int	peekI32(int _off)	считывание 32 битного числа из _off. Изменение порядка байт, при необходимости.
int	peekF32(int _off)	считывание IEEE float из _off.
	pokeI8(int _off, int _val)	записывает байт в _off.

	pokeI16(int _off, int _val)	запись 16 битного числа в _off. Конвертирование порядка байт, при необходимости.
	pokeI32(int _off, int _val)	запись 32 битного числа в _off. Конвертирование порядка байт , при необходимости.
	pokeF32(int _off, int _val)	запись значения типа float в _off.
	resize(int _size)	изменяет размер буфера. Сохраняет элементы (если _size>size).
	setByteOrder(int _type)	устанавливает порядок байт в буфере (BIG_ENDIAN/ LITTLE_ENDIAN).
	setOffset(int _off)	установить текущее смещение чтения /записи
	setSize(int _size)	освобождение буфера/отбрасывание элементов и размещение нового буфера
	int setString(int _off, String _s)	запись строки _s по смещению _off в буфер

<Методы потока> (см. Методы на стр. 101.)

Примеры:

```
Buffer b; b.size=4096;
b.i8=1;
b.i16=24;
b.i32=0xdeadbeef;
b.f32=3.1415;
print "b.offset="+b.offset;
StdOutStream.writeBuffer(b, 0, b.offset);
```

Классы массива

* однородный массив классов, определенных пользователем
* может быть доступно во внешних C++ классах

Наследование

```
Object -> ClassArray
```

Свойства

`numElements` - возвращает/устанавливает количество используемых элементов
`maxElements` - возвращает количество доступных элементов
`nextFree` - возвращает указатель на следующий свободный элемент

Методы

int	alloc(int _num)	очищает текущий массив/отбрасывает элементы и размещает _num новых элементов. Сбрасывает numElements.
	empty()	сбрасывает numElements. (Ни один элемент не будет фактически удален)
	free()	удаление элементов
Class	getNextFree()	возвращает ссылку на элементы [numElements++]
int	getNumElements()	возвращает numElements
int	getMaxElements()	возвращает maxElements
int	realloc(int _size)	изменение размещения массива (с указанием нового размера), пытаясь сохранить элементы если _size>maxElements.
	reverse()	изменение порядка элементов в массиве на противоположный
	setNumElements(int _num)	установка числа использующихся в данный момент элементов массива

Конфигурация

Наследование

```
Object -> Configuration
```

Свойства

debugLevel - возвращает/устанавливает текущий уровень отладки (0 .. 99)
forceInt - разрешает/запрещает JIT-компиляцию

Методы

	`setDebugLevel(int _level)`	(0 .. 99)
`int`	`getDebugLevel()`	возвращает текущий уровень отладки
`int`	`setForceInt(int _bool)`	вызывает интерпретируемое выполнение (т.е. запрещает JIT-компиляцию)

Double

* класс-оболочка для одного IEEE 64-битного числа с плавающей точкой
* поддержка форматированной печати
* смотри также классы Integer (см. Integer на стр. 82.) и Float (см. Float на стр. 73.)

Наследование

```
Object -> YAC_Double -> Double
```

Свойства

value - чтение/запись, текущее значение числа с плавающей точкой (сокращается до 32бит)

Методы

float	getValue ()	
	setValue (float _f)	
String	printf (String _fmt)	конвертирует значение с плавающей точкой в форматированную строку printf-стиля и возвращает её

Примеры:

double.tks (см. double.tks на стр. 135.)

Envelope (диапазон)

* интерполированный массив значений с плавающей точкой (см. Массив чисел с плавающей точкой (FloatArray) на стр. 74.)
* поддержка sample&hold линейной, косинусоидальной, квадратичной, кубической интерполяции, а также интерполяции пятой степени.

Наследование

```
Object -> FloatArray -> Envelope
```

(см. Массив чисел с плавающей точкой (FloatArray) на стр. 74.)

Свойства

```
deltaTime     -
interpolation -
speed         -
time          -
```

Константы

```
ENV_SH,   ENV_LINEAR,   ENV_COSINE,   ENV_QUADRATIC,   ENV_CUBIC,
ENV_QUINTIC, ENV_SHRESET, NUM_ENVELOPE_TYPES.
```

Методы

float	get()	
float	getDeltaTime()	
int	getInterpolation()	
float	getSpeed()	
float	getTime()	
	reset()	
	setInterpolation (int _type)	один из ENV_SH, ENV_LINEAR, ENV_COSINE..
	setSpeed (float _speed)	
	setTime (float _t)	
	tickPrecise (float _prec)	

Событие

* используется для универсального интерфейса события onEvent()
* может быть доступно во внешних C++ классах

Наследование

```
Object -> Event
```

Свойства

```
timestamp -
string    -
```

Методы

int	getTimestamp()
	setTimestamp(int)
String	getString()
	setString(String _s)

File

* используется для обращения к реальной локальной файловой системе
* прозрачная конверсия порядка следования байт
* использование файловых функций ANSI-C <stdio.h>
* также смотри класс PakFile (см. PakFile на стр. 88.), который может прозрачно отобразить логические имена файлов в локальные файлы.

Наследование

```
Object -> Stream -> File
```
(см. Поток на стр. 101.)

Свойства

`byteOrder` — чтение/запись, BIG_ENDIAN или LITTLE_ENDIAN
`object` — чтение/запись,
`offset` — чтение/запись
`size` — чтение
`i8` — чтение/запись байт из/в поток
`i16` — чтение/запись слова (*`word`*) из/в поток
`i32` — чтение/запись двойного (*`dword`*) слова из/в поток
`f32` — чтение запись числа с плавающей точкой (*`floating point number`*) из/в поток
`f64` — конвертирование *`float<->double`*

Константы

`SEEK_BEG, SEEK_CUR, SEEK_END` — параметры для *`seek()`*
`ERRINVALIDSEEK, ERRIO, ERRCREATEFILE, ERROPENFILE` — код ошибки
`IOS_OUT, IOS_IN, IOS_INOUT` — параметры для *`openLocal()`*

Методы

	close ()	закрывает файл, открытый ранее с помощью методов open() или openTemp(). Временные файлы не будут удалены автоматически.
	deserialize (Object _o, int _rtti)	преобразование объекта из потока (дополнительная информация о типе)
int	eof ()	проверка достижения конца файла
	flush ()	сброс файловых буферов (т.е. конец записи)
int	getByteOrder ()	возвращает текущий порядок байт файла потока (BIG_ENDIAN/LITTLE_ENDIAN)
int	getErrorCode ()	вернет последнюю ошибку, если файл открыт
String	getErrorStringByCode (int _code)	используется для конвертирования кода ошибки в строку, если файл открыт.
int	getI8 ()	считывание байта

int	getI16 ()	считывание 16-битного целого числа, установка порядка следования байт, расширение знака
int	getI32 ()	считывание 32-битного целого числа, установка порядка следования байт
float	getF32 ()	считывание 32-битного числа с плавающей точкой
float	getF64 ()	считывание 64-битного числа с плавающей точкой ("double"), приведение к 32-битному
	getObject (Object _o)	преобразование объекта из потока (без информации о типе)
int	getOffset ()	возвращает текущее смещение чтения/записи
int	getSize ()	возвращает размер файла.
int	isOpen ()	проверка, открыт файл или нет
	setI8 (int _i)	запись байта.
	setI16 (int _i)	запись 16 битного числа. Конвертирование к потоковому порядку байт.
	setI32 (int _i)	запись 32 битного числа. Конвертирование к потоковому порядку байт.
	setF32 (float _f)	запись 32 битного числа с плавающей точкой
	setF64 (float _f)	запись 32 битного числа с плавающей точкой ("double"), расширение до 64 битного
int	open (String _name, int _iosmode)	открытие локального файла, _iosmode должен быть один из IOS_IN, IOS_OUT, IOS_INOUT
String	openTemp (String _directory, String _prefix)	создает временный файл и возвращает его название
int	readBuffer (Buffer _b, int _off, int _num, int _resize)	считывание _num байт из потока и сохранение их в буфере по смещению _off. изменение размеров буфера, если _resize==true.
int	readString (String _s, int _num)	считывание максимальных _num символов из потока (соотв. до ASCIIZ)
	removeTemp ()	удаление временных файлов, ранее открытых при помощи метода openTemp()
	seek (int _mode, int _off)	корректировка смещения чтения/записи
	serialize (Object _o, int _rtti)	преобразование объекта в поток (доп. информация о типе)
	setByteOrder (int _order)	BIG_ENDIAN или LITTLE_ENDIAN
	setObject (Object _o)	преобразование объекта в поток (без информации о типе)
	setOffset (int _off)	подобно seek (SEEK_BEG, _off)

int	writeBuffer (Buffer _b, int _off, int _num)	запись _num байт в поток
int	writeString (String _s, int _off, int _num)	запись _num символов в поток

Примеры

testfileio.tks (см. testfileio.tks на стр. 126.)
3ds.tks (см. 3ds.tks на стр. 137.)
md2.tks (см. md2.tks на стр. 154.)

Float

* класс-оболочка для одного IEEE 32-битного числа с плавающей точкой
* поддерживает форматированную печать
* также смотри классы Integer (см. Integer на стр. 82.) и Double (см. Double на стр. 67.)

Наследование

```
Object -> Float
```

Свойства

value — чтение/запись, значение текущего числа с плавающей точкой

Методы

float	getValue()	
String	setValue(float _f)	
String	printf(String _fmt)	конвертирует число с плавающей точкой в строку формата printf-style и возвращает его.

Массив чисел с плавающей точкой (FloatArray)

* буферизированный массив чисел с плавающей точкой
* может использоваться, как список массивов
* может быть доступно во внешних C++ классах

Наследование

```
Object -> FloatArray
```

Свойства

numElements — количество используемых элементов
maxElements — количество доступных элементов

Методы

	add (float _f)	добавление элемента в конец массива (elements[numElements++])
	addEmpty (int _num)	увеличение numElements на _num
int	alloc (int _num)	размещение _num элементов, удаляя предыдущие элементы
	blend (FloatArray _o, float _f)	смешивание элементов из этого массива и массива _b
	blendAB (FloatArray _a, FloatArray _b, float _bamount)	смешивание элементов _a и _b и хранение в этом массиве
	copyFrom (FloatArray _src, int _off, int _len, int _doff)	копирование элементов из массива _src
	delete (int _idx)	удаление элемента с индексом _idx
	empty ()	numElements=0;
	fill (float _f)	установка всех элементов в _f
	free ()	удаление элементов
int	getNumElements ()	возвращает число используемых элементов
int	getMaxElements ()	возвращает число доступных элементов
	insert (int _idx, float _f)	вставка элемента _f по индексу _idx
int	read32 (Stream _ifs, int _num, int _doff)	считывание _num 32 битного числа с плавающей точкой из потока
int	read64 (Stream _ifs, int _num, int _doff)	считывание _num 64 битного числа (doubles) из потока и конвертирование его в формат с плавающей точкой

	realloc (int _num)	изменение размеров массива. Сохранение элементов, если _num>maxElements
	reverse ()	изменения порядка следования элементов
	scale (FloatArray _src, float _s)	масштабирование всех элементов _src _s и сохранение их в этом массиве
	setNumElements (int _num)	установка числа используемых элементов
int	visit (FloatArray _o, int _off, int _len)	создает отображение в другой массив (элементы не копируются)
int	write32 (Stream _ofs, int _num, int _soff)	запись _num 32 битного числа с плавающей точкой в поток
int	write64 (Stream _ofs, int _num, int _soff)	конвертирование 32 битного числа с плавающей точкой в 64 битное (doubles) и запись в поток

Функция

* ссылка на функцию скрипта

Наследование

```
Object -> Function
```

Методы

String	getName()
<var>	call()
<var>	callWithArgs()
Variable	findVariable(String _name)
	voidCall()
	voidCallWithArgs(List _args)

Пример

tempscript.tks (см. раздел tempscript.tks на стр. 162.)

Хэш-таблица

* ассоциативный массив, индексированный String
* может быть доступно во внешних C++ классах

Наследование

```
Object -> HashTable
```

Свойства

numElements — количество используемых элементов
maxElements — количество доступных элементов

Методы

int	addInt(String _name, int _i)	создание нового целочисленного элемента (integer)
int	addFloat(String _name, float _f)	создание нового элемента в формате float
int	addObject(String _name, Object _o)	создание нового элемента-объекта (копирование)
int	addObjectRef(String _name, Object _o)	создание нового элемента-объекта (ссылка)
int	addString(String _name, String _s)	создание нового строкового элемента (копирование)
int	alloc(int _num)	выделение памяти для _num элементов (только простые числа)
	delete(String _name)	удаление элемента (если существует)
int	exists(String _name)	проверка существования элемента _name
	free()	удаление всех элементов
	get(String _name)	возвращает элемент хэш-таблицы _name (ссылка)
int	getNumElements()	возвращает количество используемых элементов (numElements)
int	getMaxElements()	возвращает количество доступных элементов (maxElements)

Примеры

```
int n=150;
HashTable hash1; hash1.alloc(10000);
HashTable hash2; hash2.alloc(10000);
int i=0;
for(i=0; i<10000; i++) {
    hash1["foo_" + i] = i;
}
loop(n) {
    String k;
    foreach k in hash1
hash2[k] += hash1[k];
```

tgclso_hash.tks (см. tgclso_hash.tks на стр. 164.)
tgclso_hash2.tks (см. tgclso_hash2.tks на стр. 165.)

Целочисленный массив (IntArray)

* буферизированный массив целых чисел
* может быть использован как список массивов
* может быть доступно во внешних C++ классах

Наследование

```
Object -> IntArray
```

Свойства

numElements — количество используемых элементов
maxElements — количество доступных элементов

Методы

	add(int _i)	добавление элемента, т.е. elements[numElements++]
	addEmpty(int _num)	увеличение numElements на _num
int	alloc (int _num)	удаление элементов и размещение _num новых
	delete (int _idx)	удаление элемента с индексом _idx
	empty ()	установка numElements в 0
	fill (int _i)	установка всех элементов в _i
	free ()	удаление всех элементов
int	getNumElements ()	возвращает число используемых элементов
int	getMaxElements ()	возвращает число доступных элементов
int	insert (int _idx, int _i)	вставка элемента с индексом _idx
int	read8 (Stream _ifs, int _num, int _doff)	считывание _num байт из потока, конвертирование порядка следования байт
int	read16 (Stream _ifs, int _num, int _doff)	считывание _num 16 битного числа из потока, конвертирование порядка следования
int	read32 (Stream _ifs, int _num, int _doff)	считывание _num 32 битного числа из потока, конвертирование порядка следования байт
int	realloc (int _num)	изменение размера массива, сохранение элементов, если _num>maxElements

	reverse ()	изменение порядка следования элементов на противоположный
	setNumElements (int _num)	установка числа используемых элементов
	swapByteOrder ()	изменение порядка следования байт элемента (RGBA <=> ABGR)
int	visit (IntArray _src, int _off, int _num)	создание отображения в другой массив (элементы не копируются)
int	write8 (Stream _ofs, int _num, int _soff)	запись _num байт в поток, конвертирование порядка следования байт
int	write16 (Stream _ofs, int _num, int _soff)	запись _num 16 битного числа в поток, конвертирование порядка следования байт
int	write32 (Stream _ofs, int _num, int _soff)	запись _num 32 битного числа в поток, конвертирование порядка следования байт

Примеры:

```
int ia[10];
trace "ia.numElements="+ia.numElements;
trace "ia.maxElements="+ia.maxElements;
loop(10)
  ia.add(rnd(100));
trace "ia.numElements="+ia.numElements;
trace "ia.maxElements="+ia.maxElements;


// ---- 256x256x32 bit rotzoomer ----
int ibuffer[256*256*2];
float deltat=0;

function render() compile {
    int dx;
    int dy;
    int fx;
    int fy;
    int ffx;
    int ffy;
    int i;
   // ---- Вычисление изменения масштаба изображения и поворота----
    dx=sin(deltat)*(550.0+(sin(deltat/3.0)*500.0));
    dy=cos(deltat)*(550.0+(sin(deltat/3.0)*500.0));
    fx=((128+(sin(deltat/4.0)*900.0))<<8)-128*dx-128*dy;
    fy=((128+(cos(deltat/4.0)*300.0))<<8)-128*dy+128*dx;
    i=65536;  // буфер "tpix"  часть "buffer"
    loop(256)
```

```
{
    ffx = fx; ffy = fy;
    loop(256)
{
    fy+=dy;
    fx+=dx;
    ibuffer[i++]=ibuffer[(fy&$ff00)|((fx>>8)&$FF)];
}
    fx = ffx + dy; fy = ffy - dx;
}
}

loop(1000)
{
    render();
    deltat+=0.03;
}
```

Integer

* 32-битное знаковое целое число в объектной оболочке
* поддержка форматированной печати
* также смотри класс Float (см. Float на стр. 73.)

Наследование

```
Object -> Integer
```

Свойства

value - -

Методы

int	getValue()	
	setValue(int _i)	
String	printf(String _fmt)	конвертирует целое число (integer) в форматированную строку printf_style и возвращает ее

Список (list)

* контейнер для узлов (элементов) списка ListNodes (см. Узел (элемент) списка на стр. 85.)

Наследование

```
Object -> List
```

Свойства

* head — возвращает головной элемент списка (не имеющий предшественников)
* tail — возвращает хвостовой элемент списка (не имеющий последователей)
* copy — возвращает копию списка, начинающуюся с этого узла (ссылочные объекты)
* string — возвращает строковое представление списка (“формата {1,2,3}”)

Методы

<var>	operator [] (int _index)	получение значения узла списка #_index
	operator + (List _l)	соединение этого списка и _l (ссылочные объекты)
	operator + (ListNode _l)	соединение этого списка и _l (ссылочные объекты)
	operator ^ (List _l)	соединение этого списка и _l (захват объектов)
	operator ^ (ListNode _l)	соединение этого списка и _l (захват объектов)
ListNode	addFirst (Value _v)	вставка нового узла перед первым узлом. Копирование/захват значения _v.
ListNode	addLast (Value _v)	добавление нового узла после последнего узла. Копирование/захват объектов _v.
List	getCopy ()	создание копии этого списка. ссылочные объекты.
ListNode	getHead ()	возвращает первый узел списка
int	getSize ()	возвращает количество узлов в списке
String	getString ()	тоже, что и head.string но с проверкой на head==null
ListNode	getTail ()	возвращает последний элемент списка
int	isEmpty ()	возвращает истину (true) если список пустой.
ListNode	removeAll ()	отвязывает узлы списка и возвращает первый узел (изменяемый)
ListNode	removeFirst ()	отвязывает первый узел списка и возвращает его (изменяемый)

ListNode	removeLast ()	отвязывает последний узел списка и возвращает его (изменяемый)
	reverse ()	изменение порядка следования узлов на противоположный

<методы ListNode> (см. Методы на стр. 83.)

Примеры

смотри также tgclso_lists.tks

Узел (элемент) списка

* получен из класса Value (см. Value на стр. 110.)
* узел или заголовок двунаправленного списка
* может быть доступно во внешних C++ классах
* выражение списка {} (см. Выражение списка {} на стр. 34.) используется для создания новых двунаправленных списков значений объектов во время выполнения
* класс List (см. Список (list) на стр. 83.) также может быть использован для обработки узлов (элементов) списка

Наследование

```
Object -> Value -> ListNode
```
(см. Value на стр. 110.)

Свойства

head — возвращает головной элемент списка (который не имеет предшественников)
tail — возвращает хвостовой элемент списка (который не имеет последователей)
prev — возвращает предыдущий узел списка
next — возвращает следующий узел списка
copy — возвращает копию списка, начиная с этого узла (ссылочные объекты)

Методы

<var>	operator [] (int _index)	получить значение узла списка #_index
ListNode	operator + (ListNode _l)	соединить этот список и _l (ссылочные объекты)
ListNode	operator + (Value _v)	соединить этот список и _v (ссылочные объекты)
ListNode	operator ^ (ListNode _l)	соединить этот список и _l (захват объектов)
ListNode	operator ^ (Value _v)	соединить этот список и _v (захват объектов)
ListNode	append ()	добавить/вставить узел списка сразу после этого узла (предыдущий next становится next нового узла)
ListNode	appendValue (Value _v)	добавить в конец новый узел и установить значение, например l.appendValue(#(PI));
ListNode	copy ()	возвращает копию этого узла. Члены объекта будут ссылочными, они не будут копироваться.
String	getDebugString ()	возвращает строку в целях отладки (только этот узел)
String	getDebugStrings ()	возвращает строку в целях отладки (полный список)

ListNode	getHead ()	возвращает первый узел списка
ListNode	getNext ()	возвращает следующий узел списка (или null)
ListNode	getPrev ()	возвращает предыдущий узел списка или nulll)
String	getString ()	возвращает строку, например "{1,2,3}"
ListNode	getTail ()	возвращает последний узел списка
	setValue (Value _v)	присваивает значение этому значению (приведение типов, в случае необходимости). захват объектов.

<методы Value> (см. Методы на стр. 110.)

Примеры

```
Value v;
foreach v in {1,2.3,"hello, world",{1,2,3}}
   trace "v.type="+v.type+" v.value="+v.value;
```

list.tks (см. list.tks на стр. 166.)
tgclso_lists.tks (см. tgclso_lists.tks на стр. 167.)

Массив объектов (ObjectArray)

* буферизированный массив однородных объектов (т.е. все объекты имеют один и тот же тип)
* может использоваться как список массивов
* может быть доступно во внешних C++ классах
* также смотри PointerArray (см. Массив указателей на стр. 90.), StringArray (см. Строковый массив (StringArray) на стр. 105.) и Pool (см. Пул (Pool) на стр. 91.)

Наследование

```
Object -> ObjectArray
```

Свойства

numElements - количество используемых элементов
maxElements - количество доступных элементов
nextFree - возвращает `elements[num_elements++]`
template - устанавливает шаблон массива, например *`oa.template=Time;`*

Методы

`int`	`alloc(int _num)`
	empty ()
	free ()
`Object`	getNextFree ()
`int`	getNumElements ()
`int`	getMaxElements ()
`Object`	getTemplate ()
`int`	realloc (int _num)
	reverse ()
	setNumElements (int _num)
	setTemplate (Object _t)

PakFile

* связан с поименованными бинарными файлами (буфер данных), хранящимися в gzipd упакованном TKX файле (см. Проекты на стр. 11.) (запланирована поддержка для zip/rar файлов)
* прозрачная конверсия порядка следования байт
* прозрачное отображение имен файлов как в локальные файлы, так и в упакованные
* также смотри классы File (см. PakFile на стр. 88.), StdInStream (см. StdInStream на стр. 97.), StdOutStream (см. StdOutStream на стр. 99.) и Stream (см. Поток на стр. 101.)

Наследование

```
Object -> Stream -> PakFile
```
(см. Поток на стр. 101.)

Свойства

byteOrder - чтение/запись, BIG_ENDIAN или LITTLE_ENDIAN
object - чтение,
offset - чтение/запись
size - чтение
i8 - считывание байта (*byte*) из потока
i16 - считывание слова (*word*) из потока
i32 - считывание двойного слова (*dword*) из потока
f32 - считывание числа с плавающей точкой (*floating point number*) из потока

Константы

SEEK_BEG, SEEK_CUR, SEEK_END — аргументы для seek ()
ERRINVALIDSEEK, ERRIO, ERRCREATEFILE, ERROPENFILE — кода ошибок

Методы

	close()	
	deserialize (Object _o, int _rtti)	
int	eof ()	
int	getByteOrder ()	
int	getErrorCode ()	
String	getErrorStringByCode(int _code)	
int	getI8 ()	
int	getI16 ()	расширение знака
int	getI32 ()	
float	getF32 ()	
	getObject (Object _o)	

int	getOffset ()	
int	getSize ()	
int	isOpen ()	
int	open (String _name)	
int	readBuffer (Buffer _b, int _off, int _num, int _resize)	
int	readString (String _s, int _num)	
	seek (int _mode, int _off)	
	serialize (Object _o, int _rtti)	
	setByteOrder (int _order)	BIG_ENDIAN или LITTLE_ENDIAN
	setOffset (int _off)	

Примеры

testfileio.tks (см. testfileio.tks на стр. 126.)

3ds.tks (см. 3ds.tks на стр. 137.)

md2.tks (см. md2.tks на стр. 154.)

Массив указателей

* буферизированный массив смешанных объектов
* может использоваться как список массивов
* может быть доступно во внешних C++ классах
* также смотри ObjectArray (см. Массив объектов (ObjectArray) на стр. 87.), StringArray (см. Строковый массив (StringArray) на стр. 105.) и Pool (см. Пул (Pool) на стр. 91.)

Наследование

```
Object -> PointerArray
```

Свойства

numElements — количество использующихся элементов
maxElements — количество доступных элементов

Методы

int	alloc(int _num)
	empty()
int	findPointer(Object _o)
	free()
int	getNumElements()
int	getMaxElements()
int	realloc(int _num)
	reverse()
	setNumElements(int _i)

Пул (Pool)

* буферизированный нелинейный массив однородных объектов
* присвоение имен-идентификаторов (целых) для размещенных объектов
* очень быстрые методы alloc/free, не требуется линейный поиск
* смотри также PointerArray (см. Массив указателей на стр. 90.), ObjectArray (см. Массив объектов (ObjectArray) на стр. 87.) и StringArray (см. Строковый массив (StringArray) на стр. 105.)

Наследование

```
Object -> Pool
```

Свойства

numElements — количество используемых элементов
maxElements — количество доступных элементов
template — шаблон объекта, например *p.template=String;*

Методы

int	alloc(int _num)
	empty()
	free()
int	getIDByObject(Object _o)
int	getNumElements()
int	getMaxElements()
Object	getTemplate()
	setTemplate(Object _t)
int	qAlloc()
	qFree(int _id)
	qFreeByObject(Object _o)

Пример

```
//

module Main;

class MyPoolEntry {
    String name;
}
```

```
function main() {
    Pool pool;
    int i;
    int id;
    MyPoolEntry ce;
    String s;

    pool.template=MyPoolEntry;
    pool.alloc(32);
    loop(10)
{
    id=pool.qAlloc();
    ce<=pool[id];
    ce.name="poolentry_"+id;
}
    i=0;
    foreach id in pool
{
    trace("foreach qFree i=="+i);
    if(++i<5)
{
     pool.qFree(id);    // detach entry, does not affect iterated
list
}
    else
id=-1; // конец foreach
}
    foreach id in pool
{
    ce<=pool[id];
    s<=ce.name;
    trace("foreach entry \""+s+"\"\n");
}
    pool.free();
}
```

pool.tks (см. pool.tks на стр. 169.)

Скрипт

* используется для динамической загрузки, компиляции и выполнения скриптов

Наследование

```
Object -> Script
```

Методы

`int`	`load(String _script)`	загружает и компилирует скрипт из строки
	`unload()`	выгружает текущий откомпилированный скрипт
`Variable`	`findVariable(String _name)`	получить ссылку на переменную
`Function`	`findFunction(String _name)`	получить ссылку на функцию
	`eval()`	вычисление всех глобальных операторов

Примеры

<tempscript.tks> (см. tempscript.tks на стр. 162.)
<tempscript2.tks> (см. tempscript2.tks на стр. 163.)

Стек

Наследование

```
Object -> Stack
```

Свойства

index - указатель стека
size - размер стека
template - (определено пользователем) класс скриптового шаблона для стекового фрейма

Методы

int	getSize()
int	getIndex()
Object	getTemplate ()
	init(Object _t, int _size)
Object	pop()
Object	push()
	setIndex(int _idx)
int	setSize(int _size)
	setTemplate(Object _t)

StdErrStream

* статичный объект, связанный с <stderr> файловым потоком
* использует ANSI_C <stdio.h> файловые функции
* также смотри классы File (см. File на стр. 70.), PakFile (см. PakFile на стр. 88.), StdInStream (см. StdInStream на стр. 97.), StdOutStream (см. StdOutStream на стр. 99.) и Stream (см. Поток на стр. 101.)

Наследование

```
Object -> Stream -> StdErrStream
```
(см. Поток на стр. 101.)

Свойства

byteOrder - чтение/запись IG_ENDIAN или LITTLE_ENDIAN
object - запись,
offset - чтение
size - чтение
i8 - запись байта (*byte*) в поток
i16 - запись слова (*word*) в поток
i32 - запись двойного слова (*dword*) в поток
f32 - запись числа с плавающей точкой (*floating point number*) в поток
f64 - конвертирование *float<->double*

Константы

SEEK_BEG, SEEK_CUR, SEEK_END - аргументы для `seek()`

Методы

	close()
	deserialize(Object _o, int _rtti)
int	eof()
	flush()
int	getByteOrder()
int	getErrorCode()
String	getErrorStringByCode(int _code)
int	getOffset()
int	getSize()
int	isOpen()
	setI8(int _i)
	setI16(int _i)

	setI32(int _i)	
	setF32(float _f)	
	serialize(Object _o, int _rtti)	
	setByteOrder(int _order)	BIG_ENDIAN или LITTLE_ENDIAN
	setObject(Object _o)	
int	writeBuffer(Buffer _b, int _off, int _num)	
int	writeString(String _s, int _off, int _num)	

Пример

```
StdErrStream.writeString("hello",0,5);
```

StdInStream

- связан с файловым потоком <stdin>
- прозрачная конверсия порядка следования байт
- использует ANSI_C файловые функции
- также смотри классы File (см. File на стр. 70.), PakFile (см. PakFile на стр. 88.), StdOutStream (см. StdOutStream на стр. 99.) и Stream (см. Поток на стр. 101.)

Наследование

```
Object -> Stream -> StdInStream
```
(см. Поток на стр. 101.)

Свойства

byteOrder - чтение/запись, *BIG_ENDIAN* или *LITTLE_ENDIAN*
object - чтение,
offset - чтение/запись
size - чтение
i8 - считывание байта (*byte*) из потока
i16 - считывание слова (*word*) из потока
i32 - считывание двойного слова (*dword*) из потока
f32 - считывание числа с плавающей точкой (*floating point number*) из потока
f64 - конвертирование *float<->double*

Константы

SEEK_BEG, SEEK_CUR, SEEK_END – параметры для *seek()*

Методы

	close ()	
	deserialize (Object _o, int _rtti)	
int	eof ()	
	flush ()	
int	getByteOrder ()	
int	getErrorCode ()	
String	getErrorStringByCode (int _code)	
int	getI8 ()	
int	getI16 ()	расширение знака
int	getI32 ()	
float	getF32 ()	
	getObject (Object _o)	

int	getOffset ()	
int	getSize ()	
int	isOpen ()	
int	readBuffer (Buffer _b, int _off, int _num, int _resize)	
int	readString (String _s, int _num)	
	seek (int _mode, int _off)	
	serialize (Object _o, int _rtti)	
	setByteOrder (int _order)	BIG_ENDIAN или LITTLE_ENDIAN
	setOffset (int _off)	

Пример

```
// $ echo -n "hallo" | tks test.tks
String s;
trace StdInStream.readString(s, 2048);
trace "s.length="+s.length+" s=\""+s+"\"";
```

StdOutStream

* статичный объект, который сопоставляется с файлом потока <stdout>.
* использует ANSI-C файловые функции <stdio.h>
* также смотри классы File (см. File на стр. 70.), PakFile (см. PakFile на стр. 88.), StdInStream (см. StdInStream на стр. 97.) и Stream (см. Поток на стр. 101.)

Наследование

```
Object -> Stream -> StdOutStream
```
(см. Поток на стр. 101.)

Свойства

byteOrder - запись/чтение, BIG_ENDIAN или LITTLE_ENDIAN
object - запись,
offset - чтение
size - чтение
i8 - запись байта (*byte*) в поток
i16 - запись слова (*word*) в поток
i32 - запись двойного слова (*dword*) в поток
f32 - запись числа с плавающей точкой (*floating point number*) в поток
f64 - конвертирование *float<->double*

Константы

`SEEK_BEG, SEEK_CUR, SEEK_END` - параметры для *seek()*

Методы

	close()
	deserialize(Object _o, int _rtti)
int	eof()
	flush()
int	getByteOrder()
int	getErrorCode()
String	getErrorStringByCode (int _code)
int	getOffset ()
int	getSize ()
int	isOpen ()
	setI8 (int _i)
	setI16(int _i)
	setI32(int _i)

	setF32(float _f)	
	serialize(Object _o, int _rtti)	
	setByteOrder(int _order)	BIG_ENDIAN или LITTLE_ENDIAN
	setObject(Object _o)	
int	writeBuffer(Buffer _b, int _off, int _num)	
int	writeString(String _s, int _off, int _num)	

Пример

```
StdOutStream.writeString("hello",0,5);
```

Поток

* базовый класс для объектов Buffer и File
* поддерживает прозрачную конверсию порядка следования байт
* может быть доступен во внешних C++ классах
* смотри также классы File (см. File на стр. 70.), PakFile (см. PakFile на стр. 88.), StdInStream (см. StdInStream на стр. 97.) и StdOutStream (см. StdOutStream на стр. 99.)

Наследование

```
Object -> Stream
```

Свойства

byteOrder - чтение/запись, BIG_ENDIAN и LITTLE_ENDIAN
errorCode -
object - чтение/запись
string - чтение/запись
offset - чтение/запись
size - чтение
i8 - чтение/запись байта (*byte*) из/в поток
i16 - чтение/запись слова (*word*) из/в поток
i32 - чтение/запись двойного слова (*dword*) из/в поток
f32 - чтение/запись числа с плавающей точкой (*floating point number*) из/в поток
f64 - конвертирование *float<->double*

Методы

	deserialize (Object _o, int _btypeinfo)	обратное преобразование объекта из потока (дополнительно информация о типе)
int	eof ()	проверка достижения конца файла
int	getI8 ()	считывание следующего байта (byte)
int	getI16 ()	считывание следующего 16 битного числа, конвертирование порядка следования байтов
int	getI32 ()	считывание следующего 32 битного числа, конвертирование порядка следования байтов
float	getF32 ()	считывание следующего IEEE float
float	getF64 ()	считывание 64-битного значения с плавающей точкой ("double"), сжатие к 32битному
int	getByteOrder ()	возвращает текущий порядок следования байт (BIG_ENDIAN/LITTLE_ENDIAN)
String	getErrorStringByCode (int _errorcode)	конвертирование кодов ошибок в строки
	getObject (Object _o)	обратное преобразование объекта (без информации о типе)

int	getSize ()	возвращает текущий размер потока
	setOffset (int _i)	устанавливает текущее смещение чтения/ записи ("seek")
int	getOffset ()	возвращает текущее смещение чтения/ записи
int	readBuffer (Buffer _o, int _off, int _num , int _bresize)	считывание _num байт из потока
int	readLine (String _s, int _max)	считывание _max символов из потока или пока не достигнута "newline".
int	readString (String _s, int _len)	считывание _len символов из потока или пока не достигнуто ASCIIZ .
	seek (int _off, int _mode)	корректировка смещения чтения/записи. _mode=SEEK_CUR, SEEK_BEG или SEEK_END
	serialize (Object _o, int _btypeinfo)	преобразование объекта в поток (дополнительно информация о типе)
	setByteOrder (int)	установка порядка следования байт в потоке (BIG_ENDIAN/LITTLE_ENDIAN)
	setI8 (int _i)	запись байта (byte)
	setI16 (int _i)	запись 16 битного числа, конвертирование порядка следования байта
	setI32 (int _i)	запись 32 битного числа, конвертирование порядка следования байтов
	setF32 (float _f)	запись IEEE float
	setF64 (float _f)	запись 32 битного числа с плавающей точкой ("double"), расширение до 64 бит
	setObject (Object _o)	преобразование объекта в поток (нет информации о типе)
int	writeBuffer (Buffer _o, int _off, int _num)	запись _num байт в поток
	writeString (String _s, int _off, int _num)	запись _num символов в поток

Пример

testfileio.tks (см. testfileio.tks на стр. 126.)
3ds.tks (см. 3ds.tks на стр. 137.)

смотри также: File (см. File на стр. 70.), PakFile (см. PakFile на стр. 88.), StdInStream (см. StdInStream на стр. 97.), StdOutStream (см. StdOutStream на стр. 99.).

String

* буферизированная строка (String)
* может быть доступно во внешних C++ классах

Наследование

```
Object -> String
```

Свойства

lenght - количество символов (включая *ASCIIZ*)

Методы

int	alloc (int _num)	размещение строкового буфера
	append (String _s)	добавление строки _s
	copy (String _src)	копирование строки _s
	empty ()	сбрасывание количества используемых символов строкового буфера
int	endsWith (String _s)	проверка, оканчивается ли строка подстрокой _s
	free ()	очистка последовательности символов
	freeStack ()	очистка строкового стека (successive to words(), split())
int	getc (int _off)	получение одного символа, вы также можете использовать оператор [], например s[1]
int	getLength ()	количество символов, включая ASCIIZ
String	getWord (int _nr)	возвращает элемент из строкового стека
int	indexOf (String _s, int _startoff)	возвращает смещение первого вхождения _s
	insert (int _off, String _s)	вставка строки по указанному смещению
int	isBlank ()	возвращает истину (true) если строка содержит только символы пробелов (пробел, символ новой строки, табуляцию)
int	lastIndexOf (String _s)	возвращает смещение последнего вхождения _s
int	load (String _name, int _b_ascii)	загружает файл из VFS, удаляя возврат каретки MS-DOS если _b_ascii==1
int	loadLocal (String _name, int _b_ascii)	загружает локальный файл, удаляя возврат каретки MS-DOS если _b_ascii==1
int	numIndicesOf (String _s)	вычисление количества вхождений _s

TreeNode	parseXML ()	синтаксический анализ XML-файла и создание дерева хэш-таблицы
int	patternMatch (String _p)	простое '*?' сопоставление с образцом
	putc (int _off, int _c)	альтернативный способ установки символа, вы также можете использовать s[1]='a';
int	replace (String _s, String _t)	замена всех вхождений _s на _t
int	saveLocal (String _name)	сохранение строки в локальный файл
int	split (int _char)	разбиение строки при каждом вхождении _char и создание строкового стека
StringArray	splitSpace (int _includeembeddedstrings)	разбиение строки на слова и возврат строкового массива
StringArray	splitChar (int _char)	разбиение строки на _char и возврат строкового массива
int	startsWith (String _s)	проверка, начинается ли строка с подстроки _s
	substring (String _d, int _off, int _len)	извлечение подстроки
	toLower ()	конвертирование символов в нижний регистр
	toUpper ()	конвертирование символов с верхний регистр
	trim ()	удаление начальных и конечных пробелов строки
int	words (int _includeembeddedstrings)	разбиение строки на слова и создание строкового стека

Пример

```
int n=40000;
String str="";
loop(n)
   str.append("hello\n");
int len = str.length;
```

Пример

```
trace "tkscript "+(("word1 word2 \"rules !\"").splitSpace(true))[2]);

prints "tkscript rules !" to the console. =]
```

Пример

Заголовочный файл "YInG" C++ preprocessor (см. preprocessor на стр. 170.)

Строковый массив (StringArray)

* может быть использован, как список массивов
* может быть доступен во внешних C++ классах
* также смотри PointerArray (см. Массив указателей на стр. 90.), ObjectArray (см. Массив объектов (ObjectArray) на стр. 87.) и Pool (см. Пул (Pool) на стр. 91.)

Наслеование

```
Object -> StringArray
```

Свойства

numElements — количество используемых элементов
maxElements — количество доступных элементов
nextElements — возвращает elements[num_elements++]

Методы

	add (String _s)
	addEmpty (int _num)
int	alloc (int _num)
int	delete (int _idx)
	empty ()
	free ()
String	getNextFree ()
int	getNumElements ()
int	getMaxElements ()
int	insert (int _idx, String _s)
int	realloc (int _num)
	reverse ()
	setNumElements (int _num)
	sortByLength ()
	sortByValue (IntArray _ia, int _casesensitive)
	unset ()

Time

Наследование

```
Object -> Time
```

Свойства

houer -
min -
month -
sec -
year -
monthday -
weekday -
yearday -

Методы

	calc()
int	getHour()
int	getMin()
int	getMonth()
int	getSec()
int	getYear()
int	getMonthday()
int	getWeekday()
int	getYearday()
	now()
	setHour(int _hour)
	setMin(int _min)
	setMonth(int _month)
	setSec(int _sec)
	setYear(int _year)

Пример

`testtime.tks` (см. testtime.tks на стр. 200.)

TKS

Наследование

```
Object -> TKS
```

Методы

String	constantToString (int _val, String _prefix)
int	evalMethodByName (Object _o, String _methodname, ListNode _argumentlist, Value _return)
int	getClassID (Object _o)
String	getClassName (Object _o)
int	getVersion ()
String	getVersionString ()
	setIntPropertyByName (Object _o, String _property, int _val)
	setFloatPropertyByName (Object _o, String _property, float _val)
	setObjectPropertyByName (Object _o, String _property, Object _v)
int	getNumProperties (Object _o)
int	getPropertyAccessKeyByName (Object _o, String _name)
int	getPropertyAccessKeyByIndex (Object _o, int _index)
<var>	getPropertyByName (Object _o, String _property)
<var>	getPropertyByAccessKey (Object _o, int _ak)
String	getPropertyClassName (Object _o)
String	getPropertyNameByAccessKey (Object _o, int _ak)
<var>	newObjectByName (String _name)
<var>	newObjectByID (int class_ID)
<var>	stringToConstant (String _const)

Пример

```
testlang.tks (см. testlang.tks на стр. 197.)
```

Узел дерева (TreeNode)

* одиночный узел или заголовок прав/лев дерева
* может быть доступно во внешних C++ классах

Наследование

```
Object -> Value -> TreeNode
```

Свойства

left -
right -
parent -
root -
name -
id -

Методы

TreeNode	insertLeft (Object _t)
TreeNode	insertRight (Object _t)
TreeNode	findByName (String _s, int _depth)
	free ()
String	getId ()
TreeNode	getLeft ()
String	getName ()
int	getNumNodes ()
TreeNode	getParent ()
TreeNode	getRight ()
TreeNode	getRoot ()
TreeNode	seekByPathName (String _s)
	setId (String _s)
	setLeft(TreeNode _n)
	setName (String _name)
	setRight (TreeNode _n)
	writeToHashTable (HashTable _d)

<методы Value> (см. Методы на стр. 110.)

Пример

`testxml.tks` (см. testxml.tks на стр. 199.)

Value

* одиночное динамически вводимое значение
* может быть доступно во внешних C++ классах
* #(<expr>) value expression (см. Выражение Value #() на стр. 33.) (значение выражения) используется для создания нового объекта Value в ходе выполнения
* {} list expression (см. Выражение списка {} на стр. 34.) используется для создания новых двунаправленных списков объектов Value в процессе выполнения.

Наследование

```
Object -> Value
```

Свойства

intValue - возвращает значение, приведенное к типу *int*
floatValue - возвращает значение, приведенное к типу *float*
objectValue - возвращает значение, приведенное к типу *object*
stringValue - возвращает значение, приведенное к типу *String*
value - возвращает фактическое значение
type - возвращает тип значения *(0=void, 1=int, 2=float, 3=object, 4=string)*
newObject - создание нового объекта по шаблону

Методы

int	getIntValue ()	возвращает значение, приведенное к типу int
float	getFloatValue ()	возвращает значение, приведенное к типу float
Object	getObjectValue ()	возвращает ссылку (только для чтения) на объект
Object	derefObjectValue ()	отвязывает объектный указатель и возвращает изменяемый объект
String	getString ()	возвращает debug-style String
String	getStringValue ()	возвращает значение, приведенное к типу String
<var>	getValue ()	возвращает фактическое значение
int	getType ()	возвращает тип значения (0=void, 1=int, 2=float, 3=object, 4=string)
	setIntValue (int)	устанавливает тип и значение объекта
	setFloatValue (float)	устанавливает тип и значение объекта
	setObjectValue Object)	устанавливает тип и значение объекта, ссылку (только для чтения) на данный объект
	setStringValue (String)	устанавливает тип и значение объекта, ссылку (только для чтения) на данный объект
	setValue (Value _v)	(deref) копирование из другого значения

typecast (int _type)	изменение типа и конвертирование значения (0=void, 1=int, 2=float, 3=object, 4=string)
unset ()	удаление объектного указателя, если необходимо
setNewObject (Object _template)	создание нового образца _template

Пример

```
Value v=#(new Time);
    trace v.string;
prints
$ tks ../../test.tks
<Object "Time" RW (0x8b0008, 9109512, 1.27651e-038 "")>
to the console.
```

Пример

```
list.tks          (см. list.tks на стр. 166.)
tgclso_lists.tks  (см. tgclso_lists.tks на стр. 167.)
```

Переменная

* поименованное значение
* ссылается на скриптовую переменную в модуле/функции

Наследование

```
Object -> Value -> Variable
```
(см. Value на стр. 110.)

Свойства

name - возвращает название переменной

Методы

int	query ()	повторное считывание переменной скрипта (если она еще существует)
int	store ()	запись в скриптовую переменную (если она еще существует)
String	getName ()	возвращает название скриптовой переменное (если она еще существует)
int	storeDeref ()	отвязывает объектный указатель от значений и помещает его в скриптовую переменную (если она еще существует)

`<методы Value>` (см. Методы на стр. 110.)

Пример

`tempscript.tks` (см. tempscript.tks на стр. 162.)

tks_engine_wrapper

* встроенный класс

Методы:

void	add_typedef (const char*from, const char*to)	Объявление синонима для идентификатора (классы, стат. функции и глобальные переменные). Т.е. для идентификатора, использующегося в фабрике классов, объявляется синоним, код которым этот идентификатор будет использоваться в скрипте. (Аналогично конструкции typedef в C++).
void	add_using (const char*name)	Позволяет использовать идентификаторы из заданного пространства имен в TKScript. Смотри Пример1.
bool	load_library (const char*file_name)	Загрузка библиотеки. В качестве параметра передается имя файла, который необходимо загрузить. Позволяет загружать плагины. Возвращает логическое значение (false-библиотека не загружена; true-библиотека загружена).
bool	run_file (const char*file_name)	Запуск необходимого файла. В качестве параметра передается путь к файлу, который необходимо запустить. Функция возвращает логическое значение (false — файл не запущен; true — файл запущен).
unsigned	size ()const	Возвращает количество типов, зарегистрированных в Фабрике классов.
unsigned	staticfunc_size ()const	Возвращает количество статических функций, зарегистрированных в Фабрике классов.
unsigned	staticvar_size ()const	Возвращает количество статических переменных, зарегистрированных в Фабрике классов.
void	update ()	Позволяет обновить установленные свойства (необходимо использовать каждый раз после загрузки новых библиотек). При этом будет обновляться вся информация о типах, переменных, функциях и т.п.
	get (unsigned index)const	Позволяет получить тип, соответствующий указанному индексу (от 0 до size-1)

get_static_func (unsigned index)const	Позволяет получить статическую функцию, соответствующую указанному индексу (от 0 до size-1)
get_static_var (unsigned index)const	Позволяет получить статическую переменную, соответствующую указанному индексу (от о до size-1)

Пример1:

```
cppf.add_using("Forms::");

// аналогично конструкции C++: "using namespace Forms;"
```

type_t

* встроенный класс
* служит для описания всех классов Фабрики классов.

Методы:

size_t	constructor_size ()const	Если в качестве параметра передается имя класса — функция возвращает количество конструкторов. Иначе — возвращает 0.
size_t	func_size ()const	Если в качестве параметра передается имя класса — возвращает количество функций. Иначе — возвращает 0.
	get_constructor (unsigned index)const	Получить конструктор, соответствующий указанному индексу.
	get_func (unsigned index)const	Получить функцию, соответствующую указанному индексу.
const char*	get_module_name ()const	Получить имя модуля, в котором описывается функция.
const char*	get_name ()const	Получить имя типа.
	get_prop (unsigned index)const	Получить свойство, соответствующее указанному индексу.
	get_type_kind ()const	Получить классификатор (вид) класса. Возможны варианты: встроенный (tk_builtin), класс (tk_class), перечисление (tk_enum), событие (tk_closure).
size_t	prop_size ()const	Получить количество свойств.

member_func_t

* встроенный класс
* служит для описания функций - членов класса

Методы:

const char*	full_name ()const	Возвращает полное имя функции.
size_t	get_min_param_count ()const	Возвращает минимально-допустимое количество параметров.
const char*	get_modifier_str ()const	Возвращает модификатор вызова.
const char*	get_name ()const	Возвращает название функции.
	get_param (size_t index)const	Возвращает структуру, описывающую параметры функции.
size_t	get_param_count ()const	Возвращает максимально-допустимое количество параметров.
const char*	get_param_str ()const	Возвращает строку параметров в формате, указанном при объявлении функции.
const char*	get_result_type ()const	Возвращает тип результата.
const char*	get_retval_str ()const	Возвращает тип результата в формате, использующемся при объявлении.
bool	is_constant ()const	Возвращает логическое значение true или false если функция соответственно константная или не константная.
bool	is_static ()const	Возвращает логическое значение true в случае, если функция имеет модификатор static. Иначе возвращает false.
bool	is_virtual ()const	Возвращает логическое значение true если функция объявлена, как виртуальная. Иначе возвращает false.
const char*	own_class ()const	Возвращает класс-владелец функции, т.е. класс в котором эта функция объявлена (смотри пример 1).

Пример 1:

```
class a_t{
   void f();
}

class b_t:public a_t{
   ...
};
//  Функция own_class ()  вернет значение a_t
```

member_var_t

* встроенный класс
* служит для описания переменной - члена класса.

Методы:

const char*	get_name ()const	Возвращает название переменной
const char*	get_type ()const	Возвращает тип переменной
bool	is_static ()const	Возвращает логическое значение true если переменная имеет модификатор static. Иначе — возвращает false.
const char*	own_class ()const	Возвращает класс-владелец переменной, т.е. класс в котором эта переменная объявлена (аналогично own_class ()const для member_func_t).

static_var_t

* встраиваемый класс
* служит для описания статической переменной.

Методы:

const char*	get_module_name ()const	Возвращает имя модуля, в котором описывается переменная.
const char*	get_name ()const	Возвращает имя переменной.
const char*	get_type ()const	Возвращает тип переменной.

static_func_t

* встроенный класс
* служит для описания статических функций.

Методы:

const char*	full_name ()const	Возвращает полное имя функции.
size_t	get_min_param_count ()const	Возвращает минимально-возможное количество параметров.
const char*	get_modifier_str ()const	Возвращает модификатор вызова.
const char*	get_module_name ()const	Возвращает название модуля, в котором эта функция была объявлена.
const char*	get_name ()const	Возвращает название функции.
	get_param (size_t index)const	Возвращает структуру, описывающую параметры функции.
size_t	get_param_count ()const	Возвращает максимально-возможное количество параметров.
const char*	get_param_str ()const	Возвращает строку параметров в формате, указанном при объявлении функции.
const char*	get_result_type ()const	Возвращает тип результата.
const char*	get_retval_str ()const	Возвращает тип результата в формате, использующемся при объявлении.

func_param_t

* встроенный класс
* служит для описания параметра функции.

Методы:

const char*	get_name ()const	Возвращает название параметра функции
const char*	get_type ()const	Возвращает тип параметра функции

type_agent_t

* встроенный класс

Методы:

const char*	get_base_name ()const	Возвращает базовое имя
const char*	get_name ()const	Возвращает полное имя
bool	is_ref ()const	Возвращает логическое значение true если значение является ссылкой. Иначе возвращает false.
unsigned	pointer_level ()const	Возвращает уровень указателя.

Рассмотрим использование этих функций на примере.

Пример:

```
any some = f();
typeid(some)

// Теперь рассмотрим варианты значений, которые вернут рассмот-
// ренные функции, если some принимает значение bool** или bool&
```

	bool**	**bool&**
get_base_name	bool	bool
get_name ()	`bool**`	`bool&`
is_ref ()	false	true
pointer_level ()	2	0

Различия между TK-скриптом и его модифицированной версией

В отличии от TKS-скрипта, в модифицированной версии не используются:

* TKX-архивы не поддерживаются.
* JIT-компилятор не поддерживается.
* метод **use**
* Такие типы данных (классы), как Event, Script, Function, Variable считаются устаревшими. Использование этих классов не рекомендуется.

testlocal.tks

```
// ---- эти две функци выполняют в основном одну и ту же вещь:
// приведение целового к строковому и возврат
// изменяемого (т.е. удаляемого) объекта.
//

function ftest2(int k) {
     String s=k;
     return tcstring(s); // создание копии "s"
}

function ftest(int k){
     local String s=k;
     return deref s;
}

print ftest(2);
print ftest2(3);
```

fibonacci.tks

```
// file   : testrecursion.tks
// date   : 05Mar2002, updated 21Apr2004
// author : Bastian Spiegel <bs@tkscript.de>
// rem    : рекурсивное выполнение алгоритма Фибоначи.

module Main;

function Fib(local int _i) {
   if(_i==0)
       return 0;
   else
       if(_i==1)
           return 1;
       else
           return Fib(_i-1)+Fib(_i-2);
}

print "Fib(9)="+Fib(9);

/* эквивалентно выполнению в "C":
*
*
* #include <stdio.h>
*
* int Fib(int _i) {
*     if(_i==0)
*         return 0;
*     else
*         if(_i==1)
*             return 1;
*         else
*               return Fib(_i-1)+Fib(_i-2);
* }
*
* int main(int,char**) {
*     printf("Fib(9)=%i\n", Fib(9));
* }
*
*
*/
```

testfileio.tks

```
// file  : testfileio.tks
// author: Bastian Spiegel <flink@tkscript.de>
// date  : 21Nov02, 01Jul03
// rem   : тестирование программы для следующих классов:
//         File, Stream и Buffer

//использование tksdl;

Integer hexio;

class CTest {
    tag int myint;
    int myint_notserialized;
    tag float myfloat;
    float myfloat_notserialized;
    //tag Vector myvector;
    //Vector myvector_notserialized;
    tag String mystring;
    String mystring_notserialized;

    init() {
myint=42;
myfloat=PI;
//myvector.init(1,2,3);
mystring="hello, world.";

myint_notserialized=64;
myfloat_notserialized=2PI;
//myvector_notserialized.init(11,22,33);
mystring_notserialized="hello, unserialized world.";
    }
}

class CDTest : CTest {
    tag int myint2;
    int myint2_notserialized;
    tag float myfloat2;
    float myfloat2_notserialized;
    //tag Vector myvector2;
    //Vector myvector2_notserialized;
    tag String mystring2;
    String mystring2_notserialized;

    init() {
CTest::init();
myint2=32;
myfloat2=E;
//myvector2.init(16,32,64);
mystring2="this is yet another string.";
    }
```

```
}

class CDDTest : CDTest {
    int myint3;
    float myfloat3;
    //Vector myvector3;
    String mystring3;
}

// ---- сохранение некоторых форматированных данных в поток ----
function serialize(Pointer _stream) {
    Stream ofs;
        ofs<=_stream;  //  (внутренний)  поиск  таблицы  проверяет
(отмечает) является ли Stream базовым классом _stream.
    String s;
    Buffer b;
    int i;
    int l;
    float f;
    //IVector iv;
    //Vector v;
    //Vector4 v4;
    Integer io;
    Float fo;
    //Matrix m;

    ofs.writeString("test",0,0);
    trace("ofs.offset="+ofs.offset);

    // ---- сохранение 8 битного значения ----
    ofs.i8=255;

    // ---- сохранение 16 битного значения ----
    ofs.i16=0x7fff;

    // ---- сохранение 32 битного значения ----
    ofs.i32=(-0xffffffff /** 1 **/);

    // ---- сохранение 32 битного значенияс плавающей точкой ----
    ofs.f32=PI;

    // ---- преобразование объекта типа Integer
    //(и сохранение 32 битного целого значения) ----
    io.value=42;
    ofs << io;

    // ---- преобразование объекта типа Float
    //(и сохранение 32 битного значенияс плавающей точкой) ----
    fo.value=19.23;
    ofs << fo;

    // сохранение объекта, представляющего значение 2Пи с
    // плавающей точкой , создание и преобразование данных объекта
    // типа Float (только 32 значение с плавающей точкой)
```

```
    // без информации о типе
    ofs.object=tcobject(2PI); // сохранение (32 битного) значения
    // с плавающей точкой в поток

    //// ---- преобразование Vector Object ----
    //v.init(1,2,3); ofs << v;

    //// ---- преобразование IVector Object ----
    //iv.init(9, 23); ofs << iv;

    //// ---- преобразование Vector4 Object ----
    //v4.init(1.1, 2.2, 3.3, 4.4); ofs << v4;

    //// ---- преобразование Matrix Object ----
    //m.loadIdentity();
    //m.scale(16.0);
    //ofs << m;

    // добавление строки стиля "pascal" в файловый поток.
    // Сохранение  стиля String в поток файла. сохранение
    // размеров,а затем символов.
    s="a <256 char test string";
    ofs.i8=s.length-1;
    ofs.writeString(s,0,s.length-1); // 0 ASCII не записан

    // ---- преобразование String в поток File----
    ofs << "12345678";

    // ---- запись некотрых "необработанных" данных, полученных
    //из строки ----
    s="*DATA";
    l=s.length-1;
    for(i=0; i<16; i++)
{
    ofs.writeString(s,0,l);
}

    // ---- преобразование хэш-таблицы ----
    HashTable ht;
    ht.alloc(256);
    ht["myint"]=42;
    ht["myfloat"]=PI;
    ht["mystring"]="hello, hashtable.";
    //v<=new Vector;
    //v.init(32.0, 64.0, 128.0);
    //ht["vector"]=deref v;
    HashTable hti<=new HashTable;
    hti["myinnerint"]=23;
    hti["myinnerfloat"]=2PI;
    hti["myinnerstring"]="hello, inner hashtable.";
    ht["myhashtable"]=deref hti;
    trace "io.offset="+ofs.offset;
    ofs << ht;
    trace "+io.offset="+ofs.offset;
```

```
    // ---- преобразование Time ----
    //Time t;
    //t.now();
    //trace("serialize t.monthday="+t.monthday);
    //trace("serialize t.month="+t.month);
    //trace("serialize t.year="+t.year);
    //ofs << t;


    // ---- преобразование PointerArray ----
    PointerArray pa<=["a1","b2","c3"];
    ofs << pa;

    // ---- преобразование Color ----
    //Color clr;
    //clr.colori=#DEADBEEF;
    //ofs << clr;

    // ---- преобразование VectorArray ----
    //Vector cva,va[3];
    //cva<=va.nextFree; cva.init(1,2,3);
    //cva<=va.nextFree; cva.init(4,5,6);
    //cva<=va.nextFree; cva.init(7,8,9);
    //trace("serialize va[0]="+va[0].string);
    //trace("serialize va[1]="+va[1].string);
    //trace("serialize va[2]="+va[2].string);
    //ofs << va;

    // ---- преобразование Pool ----
    Pool pool;
    int id;
    Integer tio;
    pool.template=Integer;
    pool.alloc(8);
    //i=1;
    loop(8)
{
    id=pool.qAlloc();
    tio<=pool[id];
    tio.value=0x80+id;
    //v<=pool[id]
    //init(i, i*i, i*i*i);
    //i++;
}
    ofs << pool;

    // ---- преобразование класса -----
    CTest ctest;
    ctest.init();
    ofs << ctest;

    // ---- преобразование полученного класса ----
    CDDTest cdtest;
```

```
    cdtest.init();
    ofs << cdtest;

    // ---- преобразование ClassArray ----
    CDDTest cdda[4];
    for(i=0; i<4; i++)
_=cdda[i].init();
    cdda.numElements=4;
    ofs << cdda;

    // ---- преобразование ForcePoint ----
    //ForcePoint fp;
    //fp.position.init(1,2,3);
    //fp.speed.init(4,5,6);
    //fp.acceleration.init(7,8,9);
    //fp.damping.init(10,11,12);
    //ofs << fp;

    // ---- преобразование некоторых других данных для проверки
    // правильности потоковой позиции при обратном преобразовании--
    ofs.i32=42;
    ofs << "hello, world.";

    trace("serialized "+ofs.offset+" bytes.");
}

// ---- считывание ранее сохраненных данных и вывод их в консоле---
function deserialize(Pointer _stream) {
    Stream ifs;
    ifs<=_stream; // таблица поиска проверяет, был ли _stream
    // получен из Stream.
    String s;
    Buffer b;
    int i;
    int l;
    float f;
    //IVector iv;
    //Vector v;
    //Vector4 v4;
    Integer io;
    Float fo;
    //Matrix m;
    HashTable ht;
    HashTable hti;

    ifs.readString(s, 255);
    trace("s=\""+s+"\"");
    trace("ifs.offset="+ifs.offset);

    // ---- считывание 8 битного значения ----
    trace("i8="+ifs.i8);

    // ---- считывание 16 битного значения ----
    trace("i16="+ifs.i16);
```

```
    // ---- считывание 32 битного значения ----
    trace("i32="+ifs.i32);

    // ---- считывание 32 битного значения с плавающей точкой ----
    trace("f32="+ifs.f32);

    // ---- обратное преобразоватние объекта типа Integer ----
    io << ifs;
    trace("io.value="+io.value);

    // ---- обратное преобразование объекта типа Float----
    fo << ifs;
    trace("fo.value="+fo.value);

  // --обратное преобразование необработанного объекта типа Float--
    ifs.getObject(fo);
    trace("fo.value="+fo.value);

    //// ---- обратное преобразование объекта Vector ----
    //v << ifs;
    //trace("v="+v.string);

    //// ---- обратное преобразование объекта IVector ----
    //iv << ifs;
    //trace("iv="+iv.string);

    //// ---- обратное преобразование объекта Vector4 object ----
    //v4 << ifs;
    //trace("v4="+v4.string);

    //// ---- обратное преобразование объекта Matrix----
    ////m << ifs;
    //ifs.deserialize(m, 1);
    //trace("m="+m.string);

    // ---- считывание строки стиля "pascal" ----
    ifs.readString(s, ifs.i8);
    trace("pascal string=\""+s+"\"");

    // ---- обратное преобразование String Object ----
    s << ifs;
    //ifs.deserialize(s, 1);
    trace("\"1234567\"=\""+s+"\"");

   // ---- считывание некоторых данных "*DATA*DATA*DATA*DATA" ----
    int rlen=ifs.readString(s, 80);
    trace(rlen+" bytes read");
    trace("data=\""+s+"\"");

    // ---- обратное преобразование хэш-таблицы ----
    ht << ifs;
    trace("ht.maxElements="+ht.maxElements);
    trace("ht.numElements="+ht.numElements);
```

```
    trace("ht[\"myint\"]="+ht["myint"]);
    trace("ht[\"myfloat\"]="+ht["myfloat"]);
    trace("ht[\"mystring\"]=\""+ht["mystring"]+"\"");

//      //v<=ht["vector"];
//      //trace "testptr v="+v;
//      //trace("ht[\"vector\"]="+v.string);

    hti<=ht["myhashtable"];
    trace("hti.maxElements="+hti.maxElements);
    trace("hti.numElements="+hti.numElements);
    trace("hti[\"myinnerint\"]="+hti["myinnerint"]);
    trace("hti[\"myinnerfloat\"]="+hti["myinnerfloat"]);
    trace("hti[\"myinnerstring\"]=\""+hti["myinnerstring"]+"\"");


    // ---- обратное преобразование Time ----
    //Time t;
    //t << ifs;
    //trace("deserialize t.monthday="+t.monthday);
    //trace("deserialize t.month="+t.month);
    //trace("deserialize t.year="+t.year);

    // ---- обратное преобразование PointerArray ----
    PointerArray pa;
    trace "deserialize ptr";
    pa << ifs;
    trace("pa[0]=\""+pa[0]+"\"");
    trace("pa[1]=\""+pa[1]+"\"");
    trace("pa[2]=\""+pa[2]+"\"");

    // ---- обратное преобразование Color ----
    //Color clr;
    //clr << ifs;
    //hexio.value=clr.colori;
    //trace("clr.rgbapi=#"+hexio.printf("%08x"));

    // ---- обратное преобразование VectorArray ----
    //Vector va[];
    //va << ifs;
    //trace("deserialize va[0]="+va[0].string);
    //trace("deserialize va[1]="+va[1].string);
    //trace("deserialize va[2]="+va[2].string);

    // ---- обратное преобразование Pool ----
    trace "io.offset="+ifs.offset;
    Pool pool;
    pool << ifs;
    int id;
    Integer tio;
    foreach id in pool
{
    tio<=pool[id];
    //v<=pool[id];
```

```
    //trace("pool["+id+"]="+v.string);
    trace("pool["+id+"]="+tio.value);
}
    trace "io.offset="+ifs.offset;

    // ---- обратное преобразование класса ----
    CTest ctest;
    trace "ctest="+ctest;
    ctest << ifs;
    trace("ctest.myint="+ctest.myint);
    trace("ctest.myfloat="+ctest.myfloat);
    trace("ctest.mystring="+ctest.mystring);
    //trace("ctest.myvector="+ctest.myvector.string);

    // ---- обратное преобразование полученного класса ----
    CDDTest cdtest;
    cdtest << ifs;
    trace("cdtest.myint="+cdtest.myint);
    trace("cdtest.myfloat="+cdtest.myfloat);
    trace("cdtest.mystring="+cdtest.mystring);
    //trace("cdtest.myvector="+cdtest.myvector.string);
    trace("cdtest.myint2="+cdtest.myint2);
    trace("cdtest.myfloat2="+cdtest.myfloat2);
    trace("cdtest.mystring2="+cdtest.mystring2);
    //trace("cdtest.myvector2="+cdtest.myvector2.string);


    // ---- преобразование ClassArray ----
    CDDTest cdda[];
    cdda << ifs;
    trace("cdda.numElements="+cdda.numElements);
    for(i=0; i<4; i++)
{
    trace("cdda["+i+"].myint="+cdda[i].myint);
    //trace("cdda["+i+"].myvector2="+cdda[i].myvector.string);
}

    // ---- обратное преобразование ForcePoint ----
    //ForcePoint fp;
    //fp << ifs;
    //trace fp.position.string;
    //trace("fp.position="+ fp.position.string );
    //trace("fp.speed="+fp.speed.string);
    //trace("fp.acceleration="+fp.acceleration.string);
    //trace("fp.damping="+fp.damping.string);

 //обратное преобразование оставшегося для проверки позиции потока
    trace("42="+ifs.i32);
    ifs.deserialize(s, 1);
    trace("\"hello, world.\"=\""+s+"\"");

    trace("deserialized "+ifs.offset+" bytes.");
}
```

```
function TestTemp() {
    trace "TestTemp()";
    File fw;
    String tmpName=fw.openTemp("","");  // директория по умолчанию,
    // без префикса
    trace "[...] temp file \""+tmpName+"\" opened.\n\n";
    fw<<tcobject(42);
    fw.close();
    File fr;
    Integer io;
    if(fr.open(tmpName, IOS_IN))
{
    trace "fr.size="+fr.size;
    io<<fr;
    trace "fr.offset="+fr.offset;
    fr.close();
    trace "io='"+io.value+"'.";
    fw.removeTemp();
}
    else
die "failed to read tmp file \""+tmpName+"\"";
}

function main {
    File fs;
    Buffer b;
    // ---- преобразование stuff в File ----
    fs.open("outfile2", IOS_OUT);
    serialize(fs);
    trace("wrote "+fs.offset+" bytes to \"outfile2\"");
    fs.close();

    // ---- обратное преобразование stuff из File ----
    fs.open("outfile2", IOS_IN);
    trace("read \"outfile2\" file size="+fs.size);
    deserialize(fs);
    fs.close();

//      // ---- считывание бинарного файла ----
//      fs.openLocal("outfile2", IOS_IN);
//      trace("read \"outfile2\" file size="+fs.size);
//     fs.readBuffer(b, 0 /**offset**/, 1 /**size**/, 1 /**autore-
size**/);
//      deserialize(b);
//      fs.close();

    // ---- запись/чтение временных файлов ----
    TestTemp();
}
```

double.tks

```
// вывод программы на amd2400 + 512 МБ Win2k/pro
// откомпилированной с vc6:
/*
$ tks double.tks
7.7570187939210662e-008
7.7570184942032938e-008
difference=2.9971777235720094e-015
dc=6.6000001430511475
loop1
loop2
loop3
t1=1708 t2=1708 t3=10
*/

//использование tksdl;

int t1,t2,t3;

Double da,db,dc;
Float fa,fb,fc;
float rfa,rfb,rfc;

da=2PI;   // присвоение 32-битного значения с плавающей точкой
db=0.0000001;
dc=da*db*0.12345678; // проверка mixed Double-Double, Double-float
arithmetics
print dc.printf("%4.24g");

fa=2PI;   // рисвоение 32-битного числа с плавающей точкой
fb=0.0000001;
fc=fa*fb*0.12345678;  // проверка mixed Float-Float, Float-float
arithmetics
print fc.printf("%4.24g");
print "difference="+((dc-fc).printf("%4.24g"));

da=2.2;
dc=3*da; // проверка int-Double
print "dc="+dc.printf("%4.24g");


// ---- эталонная проверка некоторых простых цыклов для выяснения
скорости (медлительности) "объектных" арифметических операций ----
db=1.1;
fb=1.1;
rfb=1.1;
dc=3.3;
fc=3.3;
rfc=3.3;

print "loop1"; // проверка mixed float-Double, Double-float, Dou-
ble-Double arithmetics
```

```
                // t1=SDL.ticks;
loop(1000000)
   da=2.2*db*dc*4.4;
                // t1=SDL.ticks-t1;

print "loop2"; // проверка mixed float-Float, Float-float, Float-
Float arithmetics
                // t2=SDL.ticks;
loop(1000000)
  fa=2.2*fb*fc*4.4;
                // t2=SDL.ticks-t2;

print "loop3"; // проверка "real" 32bit float arithmetics
                // t3=SDL.ticks;
compile loop(1000000)
   rfa=2.2*rfb*rfc*4.4;
                // t3=SDL.ticks-t3; // ~170 раз быстрее чем loop1 и
loop2

                //print "t1="+t1+" t2="+t2+" t3="+t3;
```

3ds.tks

```
/*
 *   3DS объектный загрузчик
 *   a TKScript module written by Bastian Spiegel <bs@tkscript.de>
 *
 *   created: 8.Dec.2003
 *   changed: 12.Mar.2004
 *
 *
 */

module Simple3DS;
use tksdl;

enum {
    3DS_MAIN                        = 0x4D4D,
    3DS_EDITOR                      = 0x3D3D,

    3DS_COLOR_RGBF                  =0x010,
    3DS_COLOR_RGBI                  =0x011,
    3DS_COLOR_UNK                   =0x013,

    3DS_OBJ                         = 0x4000,
       3DS_OBJ_MESH                 = 0x4100,
       3DS_OBJ_MESH_VERTICELIST     = 0x4110,
       3DS_OBJ_MESH_FACELIST        = 0x4120,
           3DS_OBJ_MESH_FACEMAT     = 0x4130,
       3DS_OBJ_MESH_MAPCOORDLIST    = 0x4140,
           3DS_OBJ_MESH_SMOOTHLIST  = 0x4150,
       3DS_OBJ_MESH_LOCALCOORDS     = 0x4160,

       3DS_OBJ_LIGHT                = 0x4600,
           3DS_OBJ_LIGHT_SPOTLIGHT  = 0x4610,

       3DS_OBJ_CAMERA               = 0x4700,

    3DS_MAT                         = 0xAFFF,
    3DS_MAT_NAME                    = 0xA000,
    3DS_MAT_AMBIENT                 = 0xA010,
    3DS_MAT_DIFFUSE                 = 0xa020,
    3DS_MAT_SPECULAR                = 0xA030,
    3DS_MAT_MAP_TEXTURE1            = 0xA200,
    3DS_MAT_MAP_BUMP                = 0xa230,
    3DS_MAT_MAP_REFLECTION          = 0xa220,
    3DS_MAT_MAP_FILENAME            = 0xa300,
    3DS_MAT_MAP_PARAMETERS          = 0xa351,

    3DS_PAD__

};

enum {
```

```
    3DS_COLORTYPE_PAD
};

#define 3DS_AVGNUMMATERIALS 16
#define 3DS_AVGNUMOBJECTS 8

function int2string(int _i) {
    Integer io; io.value=_i; return io.printf("%04x");
}

class CGLMesh {
    FloatArray vertices;
    FloatArray normals;
    FloatArray uvcoords;
    IntArray   colors;
    Texture    tex;

    CGLMesh() {
tex<=null;
    }

    draw();
}

CGLMesh::draw() {
    glEnableClientState(GL_VERTEX_ARRAY);
    zglVertexPointer(vertices);
    glEnableClientState(GL_NORMAL_ARRAY);
    zglNormalPointer(normals);
    if(tex)
{
    tex.bind();
    glEnable(GL_TEXTURE_2D);
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);
    zglTexCoordPointer2f(uvcoords);
}
    glDrawArrays(GL_TRIANGLES, 0, vertices.numElements/3);
    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_NORMAL_ARRAY);
    if(tex)
{
    glDisableClientState(GL_TEXTURE_COORD_ARRAY);
    glDisable(GL_TEXTURE_2D);
}
}

class C3DSMaterial {
    String  name;
    Color   ambient;
    Color   diffuse;
    Color   specular;
    Texture tex_ambient;
    String  tex_ambient_name;
```

```
    /*void*/ loadTexture() {
trace "C3DSMaterial::loadTexture(material=\""+name+"\")";
if(!tex_ambient_name.isBlank())
    {
trace      "Material["+name+"]      load      ambient      texture
\""+tex_ambient_name+"\"";
tex_ambient.loadImage(tex_ambient_name, 256, 256, 4);
//tex_ambient.loadLocalImage("tex.png", 512, 512, 4);
//  tex_ambient.saveImage("test.png");
trace "tex_ambient.width="+tex_ambient.sx;
trace "tex_ambient.height="+tex_ambient.sy;
trace "tex_ambient.z="+tex_ambient.z;

tex_ambient.flags=TEX_MINFILTERLINEAR|TEX_MAGFILTERLINEAR|TEX_REPE
AT_S|TEX_REPEAT_T|TEX_MODULATE;
tex_ambient.upload();
    }
    }
}

class C3DSMesh {
    FloatArray   vertices;
    IntArray     faces;
    FloatArray   facenormals;
    FloatArray   normals;
    FloatArray   uv;
    C3DSMaterial materials; // ссылка (только для чтения) на массив
в  C3DS объект

    /*void*/ calcNormals() {
FloatArray fa<=facenormals;
if(fa.realloc((faces.numElements/4)*3))
    {
fa.numElements=faces.numElements*3;
// pass1: generate surface normals
int i=0;
int fai=0;
Vector v;
float ax,ay,az,bx,by,bz;
compile
    loop(faces.numElements/4)
    {
ax= vertices[faces[i  ]*3  ] -
    vertices[faces[i+1]*3  ];
ay= vertices[faces[i  ]*3+1] -
    vertices[faces[i+1]*3+1];
az= vertices[faces[i  ]*3+2] -
    vertices[faces[i+1]*3+2];

bx= vertices[faces[i+1]*3  ] -
    vertices[faces[i+2]*3  ];
by= vertices[faces[i+1]*3+1] -
    vertices[faces[i+2]*3+1];
```

```
bz= vertices[faces[i+1]*3+2] -
    vertices[faces[i+2]*3+2];

v.x=ay*bz - az*by;
v.y=az*bx - ax*bz;
v.z=ax*by - ay*bx;

v.unit();
fa[fai++]=v.x;
fa[fai++]=v.y;
fa[fai++]=v.z;
i=i+4;
    }
// pass2: generate point normals
FloatArray fp<=normals;
if(fp.realloc(vertices.numElements))
    {
fp.numElements=vertices.numElements;
fp.fill(0);
IntArray ifc<=faces;
fai=0;
i=0;
compile
    loop(ifc.numElements/4)
    {
fp[ifc[fai]*3  ]+=fa[i  ];
fp[ifc[fai]*3+1]+=fa[i+1];
fp[ifc[fai]*3+2]+=fa[i+2];

fp[ifc[fai+1]*3  ]+=fa[i  ];
fp[ifc[fai+1]*3+1]+=fa[i+1];
fp[ifc[fai+1]*3+2]+=fa[i+2];

fp[ifc[fai+2]*3  ]+=fa[i  ];
fp[ifc[fai+2]*3+1]+=fa[i+1];
fp[ifc[fai+2]*3+2]+=fa[i+2];
i+=3;
fai+=4;
    }
    }
i=0;
compile
    loop(normals.numElements/3)
    {
v.x=fp[i];
v.y=fp[i+1];
v.z=fp[i+2];
v.unit();
fp[i++]=v.x;
fp[i++]=v.y;
fp[i++]=v.z;
    }
    }
    }
```

```
    /*CGLMesh*/ getGLMesh() { // получение сети,
    // приготовленной к glDrawArrays()
CGLMesh r<=new CGLMesh;
IntArray fi<=faces;
int numTris=fi.numElements/4;
if(numTris)
    {
trace "getGLMesh numtris="+numTris;
FloatArray s_v<=vertices;
FloatArray s_n<=normals;
FloatArray s_uv<=uv;

r.tex<=materials[fi[3]].tex_ambient;

FloatArray d_v<=r.vertices;
d_v.alloc(numTris*3*3); d_v.numElements=d_v.maxElements;
FloatArray d_n<=r.normals;
d_n.alloc(numTris*3*3); d_n.numElements=d_n.maxElements;
IntArray d_c<=r.colors;
d_c.alloc(numTris*3); d_c.numElements=d_c.maxElements;
FloatArray d_uv<=r.uvcoords;
d_uv.alloc(numTris*3*2); d_uv.numElements=d_uv.maxElements;

int i_vn=0;
int i_uv=0;
int i_c=0;
int i_fi=0;

int ii;
compile loop(numTris)
    {
loop(3)
    {
ii=fi[i_fi]*3;
d_v[i_vn  ]=s_v[ii  ];
d_v[i_vn+1]=s_v[ii+1];
d_v[i_vn+2]=s_v[ii+2];
d_n[i_vn  ]=s_n[ii  ];
d_n[i_vn+1]=s_n[ii+1];
d_n[i_vn+2]=s_n[ii+2];
ii=fi[i_fi]*2;
d_uv[i_uv  ]=s_uv[ii ];
d_uv[i_uv+1]=s_uv[ii+1];
d_c[i_c++]=#ffffffff;

i_uv+=2;
i_vn+=3;
i_fi++;
    }
i_fi++; // skip material index
    }
    }
return deref r;
```

```
    }

}


class C3DSObject {
    String name;
    C3DSMesh meshes[];
}

class C3DS {
    C3DSMaterial materials[];
    C3DSMaterial cmaterial;

    C3DSObject objects[];
    C3DSObject cobject;
    C3DSMesh   cmesh;

    int total_numvertices;
    int total_numfaces;

    C3DS() {
    }

    Stream ifs;
    int i_filesize;
    int i_chunk_id;
    int i_chunk_size;
    int i_subchunk_id;
    int i_subchunk_size;
    String s_tmpname;

    float maxx;
    float maxy;
    float maxz;
    float minx;
    float miny;
    float minz;

    readChunkHeader() {
i_chunk_id = ifs.i16;
i_chunk_size=ifs.i32-6;
return i_chunk_id;
    }
    readSubChunkHeader() {
i_subchunk_id = ifs.i16 &0xFFFF;
i_subchunk_size=ifs.i32 -6;
//trace  "readSubChunkHeader:  i_subchunk_id="+getSubChunkName()+"
size="+i_subchunk_size;
return i_subchunk_id;
    }
    getChunkName() {
return int2string(i_chunk_id);
    }
```

```
    getSubChunkName() {
return int2string(i_subchunk_id);
    }
    readTempName() {
int i=0;
s_tmpname.alloc(128);
int c;
do {
    c=ifs.i8;
    s_tmpname[i++]=c;
} while c!=0;
s_tmpname.fixLength();
i_subchunk_size-=i;
return s_tmpname;
    }

    readColorChunk(Color _c) {
int cf=ifs.i16&0xFFFF;
int cfsz=ifs.i32-6;
switch(cf)
    {
    case 3DS_COLOR_RGBF:
trace "        colortype RGBF.";
_c.rf=ifs.f32;
_c.gf=ifs.f32;
_c.bf=ifs.f32;
_c.af=1;
break;
    case 3DS_COLOR_RGBI:
trace "        colortype RGBI.";
_c.ri=ifs.i8;
_c.gi=ifs.i8;
_c.bi=ifs.i8;
_c.af=1;
break;
    case 3DS_COLOR_UNK:
    default:
trace "      unknown color format "+cf+"..skipping.";
_c.colori=0;
ifs.seek(cfsz, SEEK_CUR);
break;
    }
trace "read color "+_c.string;
    }

    skipSubChunk() {
ifs.seek(i_subchunk_size, SEEK_CUR);
//trace            "skipSubChunk            "+getSubChunkName()+"
size="+i_subchunk_size;
    }

    seekNextSubChunkByID(int _id) {
while ( readSubChunkHeader() != _id )
    skipSubChunk();
```

```
    }

    getMaterialIndexByName(String _name) {
C3DSMaterial m;
int i=0;
foreach m in materials
    {
if(m.name==_name)
    return i;
else i++;
    }
    }

    validateIndices() {
C3DSObject o;
C3DSMesh m;
int maxmat=materials.numElements;
trace "---maxmat="+maxmat;
foreach o in objects
    foreach m in o.meshes
    {
int vtxnum=m.vertices.numElements/3;
IntArray fl<=m.faces;
int flnum=fl.numElements;
for(int i=0; i<flnum; i+=4)
    {
if(fl[i]>=vtxnum) fl[i]=0;
if(fl[i+1]>=vtxnum) fl[i+1]=0;
if(fl[i+2]>=vtxnum) fl[i+2]=0;
if(fl[i+3]>=maxmat) fl[i+3]=0;
    }
    }
    }


    /** bool **/ loadLocal3DS(String _name) {
File fs;
ifs<=fs;
if(fs.open(_name, IOS_IN))
    {
return load3DSFromStream(fs);
    }
else
    {
stderr "Load3DS: cannot open local file \""+_name+"\"";
dtrace true;
return false;
    }
    }

    /** bool **/ load3DS(String _name) {
PakFile fs;
ifs<=fs;
if(fs.open(_name))
```

```
    {
return load3DSFromStream(fs);
    }
else
    {
stderr "Load3DS: cannot open logic file \""+_name+"\"";
dtrace true;
return false;
    }
    }

    /** bool **/ load3DSFromStream(Stream _ifs) {
materials.alloc(3DS_AVGNUMMATERIALS);
objects.alloc(3DS_AVGNUMOBJECTS);
cobject<=null;
cmaterial<=null;
cmesh<=null;
total_numvertices=0;
total_numfaces=0;
maxx=-10000;
maxy=-10000;
maxz=-10000;
minx=10000;
miny=10000;
minz=10000;

FloatArray verts;
int numverts;

//  dtrace false;

if(1)//fs.openLocal(_name, 0))
    {
readChunkHeader();
if (i_chunk_id==3DS_MAIN)
{
    trace "--3DS_MAIN-- ("+i_chunk_size+" bytes)";
      seekNextSubChunkByID(3DS_EDITOR);
//      dtrace false;
    while(!ifs.eof())
{
    //trace "ifs.offset="+ifs.offset+" ifs.size="+ifs.size;
    readSubChunkHeader();
    switch(i_subchunk_id)
{
default:
//      trace "----неизвестный кусок (size="+i_subchunk_size+").";
//      trace "         i_subchunk_id="+getSubChunkName();
//      trace "         i_subchunk_size="+i_subchunk_size;
//      trace "         x readSubChunkHeader: ifs.offset="+ifs.off-
set;
    skipSubChunk();
//          dtrace false;
    break;
```

```
case 3DS_OBJ:
    trace "-------- found OBJECT chunk.";
    readTempName();
    trace "         name:" +s_tmpname;
    objects.realloc(objects.numElements+1);
    cobject<=objects.nextFree;
//        dtrace true;
//        dtrace false;
    break;

case 3DS_OBJ_MESH:
    trace "-------- found OBJECT::MESH CHUNK.";
    if(cobject)
{
    _=cobject.meshes.realloc(cobject.meshes.numElements+1);
    cmesh<=cobject.meshes.nextFree;
    cmesh.materials<=materials;
}
    else
{
    cmesh<=null;
    skipSubChunk();
}
    break;

case 3DS_OBJ_MESH_VERTICELIST:
    trace "-------- found OBJECT::MESH::VERTICELIST CHUNK.";
    if(cmesh)
{
    numverts=ifs.i16&0xFFFF;
    float cfx,cfy,cfz;
    verts<=cmesh.vertices;
    verts.alloc(numverts*3); verts.numElements=0;
    int vi=0;
    compile loop(numverts)
{
    cfx=-ifs.f32;
    if(cfx>maxx) maxx=cfx;
    if(cfx<minx) minx=cfx;
    verts.add(cfx);

    cfz=ifs.f32;
    cfy=ifs.f32;

    if(cfy>maxy) maxy=cfy;
    if(cfy<miny) miny=cfy;

    if(cfz>maxz) maxz=cfz;
    if(cfz<minz) minz=cfz;

    verts.add(cfy);
    verts.add(cfz);
```

```
        trace "v["+vi+++"]=("+cfx+";"+cfy+";"+cfz+")";
    }
        trace "read "+numverts+" vertices.";
        total_numvertices+=numverts;
    }
        else
    {
        trace "vertice list outside of mesh";
        skipSubChunk();
    }
        break;

    case 3DS_OBJ_MESH_FACELIST:
        trace "-------- found OBJECT::MESH::FACELIST CHUNK.";
        if(cmesh)
    {
        int numfaces=ifs.i16&0xFFFF;
        IntArray fia<=cmesh.faces;
        fia.alloc(numfaces*4); fia.numElements=0;
        compile loop(numfaces)
    {
        fia.add(ifs.i16&0xFFFF);
        fia.add(ifs.i16&0xFFFF);
        fia.add(ifs.i16&0xFFFF);
        fia.add(ifs.i16&0); // later material index
    }
        trace "read "+numfaces+" faces.";
        total_numfaces+=numfaces;
    }
        else
    {
        trace "face list outside of mesh";
        skipSubChunk();
    }
        break;

    case 3DS_OBJ_MESH_FACEMAT:
        trace "-------- found OBJECT::MESH::FACEMAT CHUNK.";
        if(cmesh)
    {
        int mi=getMaterialIndexByName(readTempName());
        int numentries=ifs.i16&0xFFFF;
        IntArray faces<=cmesh.faces;
        loop(numentries)
    {
        int fi=ifs.i16&0xFFFF;
        faces[fi*4+3]=mi;
    }
        trace "read material assignments for "+numentries+" faces.";
    }
        else
    skipSubChunk();
        break;
```

```
case 3DS_OBJ_MESH_MAPCOORDLIST:
    trace "-------- found OBJECT::MESH::MAPCOORDLIST CHUNK.";
    verts<=cmesh.uv;
    numverts=ifs.i16&0xFFFF;
    verts.alloc(numverts*2); verts.numElements=0;
    compile loop(numverts)
{
    float xtf=ifs.f32;
//        trace "u="+xtf;
    verts.add(xtf);
    xtf=ifs.f32;
//        trace "v="+xtf;
    verts.add(1-xtf);
//      verts.add(ifs.f32);
//      verts.add(ifs.f32);
}
    trace "read "+numverts+" u/v coordinates.";
    //skipSubChunk();
    break;

case 3DS_OBJ_MESH_SMOOTHLIST:
    trace "-------- found OBJECT::MESH::SMOOTHLIST CHUNK.";
    skipSubChunk();
    break;

case 3DS_OBJ_MESH_LOCALCOORDS:
    trace "-------- found OBJECT::MESH::LOCALCOORDS CHUNK.";
    skipSubChunk();
    break;

case 3DS_OBJ_LIGHT:
    trace "-------- found OBJECT::LIGHT CHUNK.";
    skipSubChunk();
    break;

case 3DS_OBJ_LIGHT_SPOTLIGHT:
    trace "-------- found OBJECT::LIGHT::SPOTLIGHT CHUNK.";
    skipSubChunk();
    break;

case 3DS_OBJ_CAMERA:
    trace "-------- found OBJECT::CAMERA CHUNK.";
    skipSubChunk();
    break;

case 3DS_MAT:
    trace "-------- found MATERIAL CHUNK.";
    materials.realloc(materials.numElements+1);
    cmaterial<=materials.nextFree;
    //skipSubChunk();
    break;

case 3DS_MAT_NAME:
    trace "-------- found MATERIAL::NAME CHUNK.";
```

```
    if(cmaterial)
{
    cmaterial.name=readTempName();
    trace "---- name=\""+cmaterial.name+"\"";
}
    else
{
    skipSubChunk();
}
    break;

case 3DS_MAT_AMBIENT:
    trace "-------- found MATERIAL::AMBIENTCOLOR CHUNK.";
    if(cmaterial)
    readColorChunk(cmaterial.ambient);
    else
skipSubChunk();
    break;

case 3DS_MAT_DIFFUSE:
    trace "-------- found MATERIAL::DIFFUSECOLOR CHUNK.";
    if(cmaterial)
    readColorChunk(cmaterial.diffuse);
    else
skipSubChunk();
    break;

case 3DS_MAT_SPECULAR:
    trace "-------- found MATERIAL::SPECULAR CHUNK.";
    if(cmaterial)
readColorChunk(cmaterial.specular);
    else
skipSubChunk();
    break;

case 3DS_MAT_MAP_TEXTURE1:
    trace "-------- found MATERIAL::MAP_TEXTURE1 CHUNK.";
    break;

case 3DS_MAT_MAP_BUMP:
    trace "-------- found MATERIAL::MAP_BUMP CHUNK.";
    skipSubChunk();
    break;

case 3DS_MAT_MAP_REFLECTION:
    trace "-------- found MATERIAL::MAP_REFLECTION CHUNK.";
    skipSubChunk();
    break;

case 3DS_MAT_MAP_FILENAME:
    trace "-------- found MATERIAL::MAP::FILENAME CHUNK.";
    if(cmaterial)
{
    cmaterial.tex_ambient_name=readTempName();
```

```
            trace   "----   ambient   texture   filename=\""+cmate-
rial.tex_ambient_name+"\"";
}
    else
skipSubChunk();
    break;

case 3DS_MAT_MAP_PARAMETERS:
    trace "-------- found MATERIAL::MAP::PARAMETERS CHUNK.";
    skipSubChunk();
    break;
}
}

    // ---- конец синтаксического анализа
    dtrace true;
    validateIndices();
     trace "Load3DS: found "+materials.numElements+" materials and
"+
objects.numElements+" objects.";
     trace "                  #triangles="+total_numfaces+" #verti-
ces="+total_numvertices;
    return true;
}
else {
    stderr "Load3DS: missing 3DS_MAIN.";
    dtrace true;
    return false;
}
    }
    }


    normalize(float _scl) {
trace "normalize(): max=("+maxx+","+maxy+","+maxz+")";
trace "             min=("+minx+","+miny+","+minz+")";
//  float scx=1.0/(maxx-minx);
//  float scy=1.0/(maxy-miny);
//  float scz=1.0/(maxz-minz);
float scx=1.0/mathAbsMaxf(maxx,minx);
float scy=1.0/mathAbsMaxf(maxy,miny);
float scz=1.0/mathAbsMaxf(maxz,minz);
float scl;
compile
    if(scx<scy)
if(scx<scz)
    scl=scx;
else
    scl=scz;
    else
if(scy<scz)
    scl=scy;
else
    scl=scz;
```

```
scl=abs(scl)*_scl;
//  trace "normalize: scl="+scl;

float ax=(minx+maxx)*-0.5;
float ay=(miny+maxy)*-0.5;
float az=(minz+maxz)*-0.5;
//  trace "ax="+ax+" ay="+ay+" az="+az;

C3DSMesh m;
C3DSObject o;
foreach o in objects {
    foreach m in o.meshes {
FloatArray va<=m.vertices;
int i=0;
compile loop(va.numElements/3) {
    va[i]=(va[i]+ax)*scl;
    i++;
    va[i]=(va[i]+ay)*scl;
    i++;
    va[i]=(va[i]+az)*scl;
    i++;
    //trace "v=("+va[i-3]+","+va[i-2]+","+va[i-1]+")";
}
    }
}
    }

    calcNormals() {
C3DSObject o;
C3DSMesh m;
foreach o in objects
    foreach m in o.meshes
          m.calcNormals();
    }

    loadTextures() {
C3DSMaterial m;
foreach m in materials
    {
m.loadTexture();
    }
    }


}









/*
```

http://www.spacesimulator.net/tut4_3dsloader.html

In this table we can see the offset (in byte) and the length (in byte) of each field in a typical chunk:
Offset Length
0 2 Chunk identifier
2 4 Chunk length: chunk data + sub-chunks(6+n+m)
6 n Data
6+n m Sub-chunks

```
MAIN CHUNK 0x4D4D
   3D EDITOR CHUNK 0x3D3D
      OBJECT BLOCK 0x4000
         TRIANGULAR MESH 0x4100
            VERTICES LIST 0x4110
            FACES DESCRIPTION 0x4120
               FACES MATERIAL 0x4130
            MAPPING COORDINATES LIST 0x4140
               SMOOTHING GROUP LIST 0x4150
            LOCAL COORDINATES SYSTEM 0x4160
         LIGHT 0x4600
            SPOTLIGHT 0x4610
         CAMERA 0x4700
      MATERIAL BLOCK 0xAFFF
         MATERIAL NAME 0xA000
         AMBIENT COLOR 0xA010
         DIFFUSE COLOR 0xA020
         SPECULAR COLOR 0xA030
         TEXTURE MAP 1 0xA200
         BUMP MAP 0xA230
         REFLECTION MAP 0xA220
         [SUB CHUNKS FOR EACH MAP]
            MAPPING FILENAME 0xA300
            MAPPING PARAMETERS 0xA351
      KEYFRAMER CHUNK 0xB000
         MESH INFORMATION BLOCK 0xB002
         SPOT LIGHT INFORMATION BLOCK 0xB007
         FRAMES (START AND END) 0xB008
            OBJECT NAME 0xB010
            OBJECT PIVOT POINT 0xB013
            POSITION TRACK 0xB020
            ROTATION TRACK 0xB021
            SCALE TRACK 0xB022
            HIERARCHY POSITION 0xB030



MAIN CHUNK 0x4D4D
   3D EDITOR CHUNK 0x3D3D
      OBJECT BLOCK 0x4000
         TRIANGULAR MESH 0x4100
            VERTICES LIST 0x4110
```

```
            FACES DESCRIPTION 0x4120
            MAPPING COORDINATES LIST 0x4140

Опишем в деталях этот кусок:
MAIN CHUNK
Identifier 0x4d4d
Length 0 + sub-chunks length
Chunk father None
Sub chunks 3D EDITOR CHUNK
Data None
3D EDITOR CHUNK
Identifier 0x3D3D
Length 0 + sub-chunks length
Chunk father MAIN CHUNK
Sub chunks OBJECT BLOCK, MATERIAL BLOCK, KEYFRAMER CHUNK
Data None
OBJECT BLOCK
Identifier 0x4000
Length Object name length + sub-chunks length
Chunk father 3D EDITOR CHUNK
Sub chunks TRIANGULAR MESH, LIGHT, CAMERA
Data Object name
TRIANGULAR MESH
Identifier 0x4100
Length 0 + sub-chunks length
Chunk father OBJECT BLOCK
Sub chunks VERTICES LIST, FACES DESCRIPTION, MAPPING COORDINATES
LIST
Data None
VERTICES LIST
Identifier 0x4110
Length varying + sub-chunks length
Chunk father TRIANGULAR MESH
Sub chunks None
Data Vertices number (unsigned short)
Vertices list: x1,y1,z1,x2,y2,z2 etc. (for each vertex: 3*float)
FACES DESCRIPTION
Identifier 0x4120
Length varying + sub-chunks length
Chunk father TRIANGULAR MESH
Sub chunks FACES MATERIAL
Data Polygons number (unsigned short)
Polygons list: a1,b1,c1,a2,b2,c2 etc. (for each point: 3*unsigned
short)
Face flag: face options, sides visibility etc. (unsigned short)
MAPPING COORDINATES LIST
Identifier 0x4140
Length varying + sub-chunks length
Chunk father TRIANGULAR MESH
Sub chunks SMOOTHING GROUP LIST
Data Vertices number (unsigned short)
Mapping coordinates list: u1,v1,u2,v2 etc. (for each vertex:
2*float)
*/
```

md2.tks

```
module MD2;


//
.....................................................................
..........
function StreamReadString(Stream _ifs, int _l) {
    int i=0;
    String s_tmpname<=new String;
    s_tmpname.alloc(_l);
    int c;
    do {
c=_ifs.i8;
s_tmpname[i]=c;
    } while (++i<_l);
    s_tmpname.fixLength();
    return deref s_tmpname;
}

// ---------------------------------------------------------------------
--------------
class CMD2_Triangle {
    int vertexIndices [3];
    int textureIndices[3];
}


// ---------------------------------------------------------------------
--------------
class CMD2_Frame {
    CMD2  md2;

    float    scaling[3];
    float    translation[3];
    String   name;
    int      indexed_vertices[];
    float    vertices[];
    float    frame_scl;

    float    face_normals[];  // md2.numTriangles
    float    point_normals[]; // numTriangles*3

    /* void    */ convertIndexedVertices();
    /*         */ scaleFrame(float _scl);
    /*         */ calcNormals();

    /* CGLMesh */ getGLMesh(Texture _tex);
    /*         */ createGLMesh(CGLMesh _mesh, Texture _tex);
}
```

```
//
.....................................................................
............
CMD2_Frame::convertIndexedVertices() {
    int ni=(indexed_vertices.numElements/4);
      vertices.alloc(ni*3);  vertices.numElements=vertices.maxEle-
ments;
    int i=0;
    int j=0;
    float maxx=0,maxy=0,maxz=0,t;
    compile loop(ni)
{
    float x,y,z;
    t=(indexed_vertices[i++]*scaling[0])+translation[0];
    if(abs(t)>maxx)
maxx=abs(t);
    vertices[j++]=t;
    t=(indexed_vertices[i++]*scaling[1])+translation[1];
    if(abs(t)>maxy)
maxy=abs(t);
    vertices[j++]=t;
    t=(indexed_vertices[i++]*scaling[2])+translation[2];
    if(abs(t)>maxz)
maxz=abs(t);
    vertices[j++]=t;

    i++; // пропуск lightNormalIndex
}
    // ---- нормализация ----
    maxx=(maxx==0)?1:(1/maxx);
    maxy=(maxy==0)?1:(1/maxy);
    maxz=(maxz==0)?1:(1/maxz);

frame_scl=(maxx<maxy)?(maxx<maxz)?maxx:maxz:(maxy<maxz)?maxy:maxz;
     //трассировка "CMD2_Frame::convertIndexedVertices: converted
"+indexed_vertices.numElements+" vertices.";
}

CMD2_Frame::scaleFrame {
    int j=0;
    float cx,cy,cz;
    int ni=indexed_vertices.numElements/4;
    compile loop(ni)
{
    cx=vertices[j]; cy=vertices[j+1]; cz=vertices[j+2];
    vertices[j  ]=-cy*_scl;
    vertices[j+1] =cz*_scl;
    vertices[j+2]=-cx*_scl;
    j=j+3;
}
}

CMD2_Frame::calcNormals() {
}
```

```
//
.....................................................................
............
CMD2_Frame::getGLMesh {
    CGLMesh m<=new CGLMesh;
    createGLMesh(m, _tex);
    return deref m;
}

CMD2_Frame::createGLMesh {
    CGLMesh m<=_mesh;
    m.tex<=_tex;
    FloatArray m_vtx<=m.vertices;
    FloatArray m_nrm<=m.normals;
    FloatArray m_uv<=m.uvcoords;
    IntArray m_col<=m.colors;
    int i_totalNumVertices=md2.numTriangles*3;
                                m_vtx.alloc(i_totalNumVertices*3);
m_vtx.numElements=m_vtx.maxElements;
                                m_nrm.alloc(i_totalNumVertices*3);
m_nrm.numElements=m_nrm.maxElements;
    m_uv.alloc(i_totalNumVertices*2);  m_uv .numElements=m_uv .max-
Elements;
                                  m_col.alloc(i_totalNumVertices);
m_col.numElements=m_col.maxElements;
    int i=0,j=0,k=0,l=0;
    int ti;
    CMD2_Triangle ctri;
    compile loop(md2.numTriangles)
{
    ctri<=md2.triangles[j++];

    ti=ctri.vertexIndices[0]*3;
    m_vtx[i++]=vertices[ti++];
    m_vtx[i++]=vertices[ti++];
    m_vtx[i++]=vertices[ti];
        //trace  "vtx=("+m_vtx[i-3]+";  "+m_vtx[i-2]+";  "+m_vtx[i-
1]+")";
    ti=ctri.textureIndices[0]*2;
    m_uv[k++]=md2.texcoords[ti++];
    m_uv[k++]=md2.texcoords[ti];
//      trace "uv=("+m_uv[k-2]+"; "+m_uv[k-1]+")";
    m_col[l++]=#ffffffff;

    ti=ctri.vertexIndices[1]*3;
    m_vtx[i++]=vertices[ti++];
    m_vtx[i++]=vertices[ti++];
    m_vtx[i++]=vertices[ti];
    ti=ctri.textureIndices[1]*2;
    m_uv[k++]=md2.texcoords[ti++];
    m_uv[k++]=md2.texcoords[ti];
    m_col[l++]=#ffffffff;
```

```
    ti=ctri.vertexIndices[2]*3;
    m_vtx[i++]=vertices[ti++];
    m_vtx[i++]=vertices[ti++];
    m_vtx[i++]=vertices[ti];
    ti=ctri.textureIndices[2]*2;
    m_uv[k++]=md2.texcoords[ti++];
    m_uv[k++]=md2.texcoords[ti];
    m_col[l++]=#ffffffff;
}
}

// ---------------------------------------------------------------
--------------
class CMD2 {
    int version;
    int skinWidth;
    int skinHeight;
    int frameSize;
    int numSkins;
    int numVertices;
    int numTexCoords;
    int numTriangles;
    int numGLCommands;
    int numFrames;
    int offsetSkins;
    int offsetTexCoords;
    int offsetTriangles;
    int offsetFrames;
    int offsetGLCommands;
    int offsetEnd;

    CMD2_Triangle triangles[];
    CMD2_Frame    frames[];
    float         texcoords[]; /* u/v pairs, normalized */

    /*bool*/ loadMD2(String _name);
    /*bool*/ loadMD2FromStream(Stream _ifs);
    /*     */ parseTriangles(Stream _ifs);
    /*     */ parseFrames(Stream _ifs);
    /*     */ parseTexCoords(Stream _ifs);
    /*CGLShapeAnim*/ getGLShapeAnim(Texture _tex);
}


//
.....................................................................
............
CMD2::loadMD2 {
    PakFile f;
    if(f.open(_name))
{
    f.byteOrder=LITTLE_ENDIAN;
    int r=loadMD2FromStream(f);
    f.close();
```

```
    return r;
}
    else
{
    trace "[---] error opening MD2 filestream \""+_name+"\".";
    return 0;
}
}

//
.....................................................................
............
CMD2::loadMD2FromStream() {
    trace "loadMD2FromStream()";
    if(_ifs.i8=='I')
if(_ifs.i8=='D')
    if(_ifs.i8=='P')
if(_ifs.i8/*=='2'*/)
    {
trace "[...] found header";
version          = _ifs.i32;
skinWidth        = _ifs.i32;
skinHeight       = _ifs.i32;
frameSize        = _ifs.i32;
numSkins         = _ifs.i32;
numVertices      = _ifs.i32;
numTexCoords     = _ifs.i32;
numTriangles     = _ifs.i32;
numGLCommands    = _ifs.i32;
numFrames        = _ifs.i32;
offsetSkins      = _ifs.i32;
offsetTexCoords  = _ifs.i32;
offsetTriangles  = _ifs.i32;
offsetFrames     = _ifs.i32;
offsetGLCommands = _ifs.i32;
offsetEnd        = _ifs.i32;

trace "\tversion         ="+version;
trace "\tskinWidth       ="+skinWidth;
trace "\tskinHeight      ="+skinHeight;
trace "\tframeSize       ="+frameSize;
trace "\tnumSkins        ="+numSkins;
trace "\tnumVertices     ="+numVertices;
trace "\tnumTriangles    ="+numTriangles;
trace "\tnumGLCommands   ="+numGLCommands;
trace "\tnumFrames       ="+numFrames;
trace "\toffsetSkins     ="+offsetSkins;
trace "\toffsetTexCoords ="+offsetTexCoords;
trace "\toffsetTriangles ="+offsetTriangles;
trace "\toffsetFrames    ="+offsetFrames;
trace "\toffsetGLCommands="+offsetGLCommands;
trace "\toffsetEnd       ="+offsetEnd;

  parseTexCoords(_ifs);
```

```
  parseTriangles(_ifs);
  parseFrames(_ifs);

return 1;
    }
    return 0;
}

//
.....................................................................
............
CMD2::parseTriangles {
    _ifs.offset=offsetTriangles;
    triangles.alloc(numTriangles);
    CMD2_Triangle ctri;
    loop(numTriangles)
{
ctri<=triangles.nextFree;
IntArray tex<=ctri.textureIndices;
IntArray vtx<=ctri.vertexIndices;
tex.numElements=3;
vtx.numElements=3;
vtx[0]=_ifs.i16;
vtx[1]=_ifs.i16;
vtx[2]=_ifs.i16;
tex[0]=_ifs.i16;
tex[1]=_ifs.i16;
tex[2]=_ifs.i16;
}
    trace "[...] read "+numTriangles+" triangles.";
}

//
.....................................................................
............
CMD2::parseFrames {
    _ifs.offset=offsetFrames;
    frames.alloc(numFrames);
    CMD2_Frame cf;
    int framenr=0;
    float scl=1;
    loop(numFrames)
{
    cf<=frames.nextFree;
    cf.md2<=this;
    IntArray vtx<=cf.indexed_vertices;
    vtx.alloc(numVertices*4); vtx.numElements=vtx.maxElements;
    FloatArray fa<=cf.scaling;
    fa[0]=_ifs.f32;
    fa[1]=_ifs.f32;
    fa[2]=_ifs.f32;
    fa<=cf.translation;
    fa[0]=_ifs.f32;
    fa[1]=_ifs.f32;
```

```
    fa[2]=_ifs.f32;
    cf.name=StreamReadString(_ifs, 16);
    trace "parseFrame["+framenr+++"] name=\""+cf.name+"\".";
    int i=0;
    compile loop(numVertices)
{
    vtx[i++]=_ifs.i8;
    vtx[i++]=_ifs.i8;
    vtx[i++]=_ifs.i8;
    vtx[i++]=_ifs.i8;
}
    cf.convertIndexedVertices();
    if(cf.frame_scl<scl)
scl=cf.frame_scl;
}
    frames.numElements=0;
    loop(numFrames)
{
    cf<=frames.nextFree;
    cf.scaleFrame(scl);
}
}

//
.....................................................................
............
CMD2::parseTexCoords {
    trace "parseTExCoords:";
    if(numTexCoords&&skinWidth&&skinHeight)
{
    texcoords.alloc(2*numTexCoords);
    _ifs.offset=offsetTexCoords;
    int i=0;
    compile loop(numTexCoords)
{
    texcoords[i++]=tcfloat(_ifs.i16)/skinWidth;
    texcoords[i++]=tcfloat(_ifs.i16)/skinHeight;
      trace "parse texcoord ("+texcoords[i-2]+"; "+texcoords[i-
1]+")";
}
}
   trace "[...] CMD2::parseTexCoords: parsed "+numTexCoords+" tex-
ture coordinates.";
}

//
.....................................................................
............
CMD2::getGLShapeAnim {
    CGLShapeAnim a<=new CGLShapeAnim;
    a.meshes.alloc(numFrames);
    int i=0;
    compile loop(numFrames)
{
```

```
        CGLMesh m<=a.meshes.nextFree;
        CMD2_Frame f<=frames[i++];
        f.createGLMesh(m, _tex);
    }
        a.initAnim(0, numFrames, 0, numFrames);
        a.initVertices();
        return deref a;
    }
```

tempscript.tks

```
Script sc;
String s_script;//="trace \"hello, world.\";";
s_script.loadLocal("tempscript2.tks", true);

if(sc.load(s_script))
{
    trace "loaded.";
    sc.eval();

    Variable v;
    Function f;

    // чтение / запись некоторых переменных
    v<=sc.findVariable("i_test"); trace v.intValue;
    v.intValue=24; v.store();
    v<=sc.findVariable("f_test"); trace v.floatValue;
    v.floatValue="1.234567"; v.store();
    v<=sc.findVariable("s_test"); trace v.stringValue;
    v.stringValue="new s_test value"; v.store();
    v<=sc.findVariable("ia_test"); trace v.objectValue;

    _=sc.findFunction("F_Test").voidCall();

    f<=sc.findFunction("F_Test");
    if(f)
{
    trace "found function \""+f.name+"\".";
    f.voidCall();
}

    f<=sc.findFunction("F_Test_R");
    if(f)
{
    trace "found function \""+f.name+"\".";
    trace "f.call(): "+f.call();
}

    f<=sc.findFunction("F_Test_Args");
    if(f)
{
    trace "found function \""+f.name+"\".";
    Value args[4]; args.numElements=0;
    Value ca;
    ca<=args.nextFree; ca.intValue=42;
    ca<=args.nextFree; ca.floatValue=PI;
    ca<=args.nextFree; ca.stringValue="hello";
    ca<=args.nextFree; ca.objectValue=f;
    trace "f.call(): "+f.callWithArgs(args);
    f.voidCallWithArgs(args);

    f.voidCallWithArgs({23,2PI,"hi!",f});}}
```

tempscript2.tks

```
trace "hello, world.";

int i_test=23;
float f_test=64.64;
String s_test="hello";
IntArray ia_test; ia_test.alloc(10);

function F_Test() {
    trace "F_Test() called.";
    trace "i_test=\""+i_test+"\".";
    trace "f_test=\""+f_test+"\".";
    trace "s_test=\""+s_test+"\".";
    trace "ia_test=\""+ia_test+"\".";
    trace "F_Test() finished.";
}

function F_Test_R() {
    trace "F_Test_R() called.";
    return "hello";
}

function F_Test_Args(int _i, float _f, String _s, Pointer _p) {
    trace "i="+_i;
    trace "f="+_f;
    trace "s="+_s;
    trace "p="+_p;
    return _i+_f+_s;
}
```

tgclso_hash.tks

```
/*
 * http://www.bagley.org/~doug/shootout/bench/hash
 *
 * benchmark results (amd athlon 2400+, 512MB RAM)
 *  python: ~4.0s
 *  tkscript: ~4.3s
 *  perl: ~4.7s
 *  java: (crashes with "java.lang.OutOfMemoryError")
 *  I reduced the number of hash elements from 1,000,000 to 500,000
 *  then the program finished. Took ~3s so you can expect ~6s
 *  with 1,000,000
 *  elements. (btw: in interpreted mode it took ~9s
 *  for 500,000 elements)
 *
 *  perl0.5.8.0 (cygwin), python2.3 (win32),
 *  java 1.4.2_04 (win32), tkscript 0.8.7.2 (win32, MSVC compiled),
 *
 * benchmark results (g4 800, apple macosX 10.33, 768MB RAM)
 *       tkscript: ~15s
 */

int i=1, c = 0;
int n = 1000000; //(int)argv[-1]; if (n < 1) n = 1;
HashTable X; X.alloc(n);
Integer io; io.value=1;
String s;
loop(n) {
    io.value=i++;
    X[io.printf("%x")]=i;
}
loop(n)
  if (X[tcstring(--i)]) c++;

trace c;
```

tgclso_hash2.tks

```
/*
 *    http://www.bagley.org/~doug/shootout/bench/hash2
 *
 * 1.791s (amd athlon 2500+)
 * 1.806s (amd athlon 2500+, interpreted mode)
 */

use tksdl;
int t=SDL.ticks;

int n=150;
HashTable hash1; hash1.alloc(10000);
HashTable hash2; hash2.alloc(10000);
int i=0;
for(i=0; i<10000; i++) {
    hash1["foo_" + i] = i;
}
loop(n) {
    String k;
    foreach k in hash1
hash2[k] += hash1[k];

}

t=SDL.ticks-t;

trace "t="+t;
trace
    hash1["foo_1"]+" "+ hash1["foo_9999"]+" "+
    hash2["foo_1"]+" "+ hash2["foo_9999"];
```

list.tks

```
//
// ----    file: list.tks
// ----  author: bastian spiegel <bs@tkscript.de>
// ---- created: 14-Jan-2004, 28-Mar-2004
//

ListNode l;

l =  {1,2,new String};// копирование списка (присваивание объектов)
l <= {1,2}+{3,4,new String};//конкатенирование (ссылочные объекты)
l <= {1,2}^{3,4,new String};//конкатенирование списков
 //(присваивание объектов)

print "the 3rd listnode of {1,2,3,4} is "+({1,2,3,4}[2].string); /
/ печать списка, начиная с третьего элемента
print "the 3rd value of {1,2,3,4} is "+({1,2,3,4}[2].stringValue);
// печать значения третьео элемента

l<={1,4,3};
l[1]=2;
trace l.string;

l<={1,2}+#("3");
trace l.debugStrings;

l<={1,2.2,"hallo"};
ListNode c;
c<=l.tail; c.appendValue(#("welt"));
c<=l.tail; c.appendValue(#(tcobject(42)));
c<=l.tail; c.appendValue(#(new String()));

Value v;
foreach v in {1,2.2,"hi",tcobject(42),#(5),6}
    trace "v.type="+v.type+" v.string="+v.stringValue;
trace l;
trace l.string;

ListNode l2,l3;

function evalListExpr() {
    trace "--------------------------------";
    l2<={rnd(1),rnd(2),rnd(3),#(rnd(42))};
    l3<=l2.copy;
    c<=l3.tail; c.appendValue(#("EOL."));
    trace l3;
    trace l3.string;
}

loop(1)
   evalListExpr();
```

tgclso_lists.tks

```
// $Id: lists.java,v 1.1 2001/01/14 18:13:15 doug Exp $
// http://www.bagley.org/~doug/shootout/

// ported to tkscript on Wed Apr 7 19:37:10 2004 by Bastian Spiegel
<bs@tkscript.de>

// benchmark results: (amd 2400+ "barton", 512MB RAM)
//         gcc: ~0.3125 (source: http://www.bagley.org/~doug/shoot-
out/bench/lists/lists.gcc)
//        perl: ~0.3375s (10.8s for NUM=512)
//      python: ~0.3750s (12s for NUM=512)
//    tkscript: ~0.4375s (14s for NUM=512)
//        java: ~0.5531s (17,7s for NUM=512)
// java(nojit): ~1.2881s (39,3s for NUM=512)
//
// perl0.5.8.0 (cygwin), python2.3 (win32), java 1.4.2_04 (win32),
tkscript 0.8.7.2 (win32, MSVC compiled),
//
// benchmark results (g4 800, apple macosX 10.33, 768MB RAM)
//       tkscript: ~2s

#define SIZE 10000

function test_lists()  {
    int i,result = 0;
    // создание списка целых чисел (Li1) от 1 до SIZE
    List Li1<= new List();
    i=1;
    loop(SIZE)
Li1.addLast(#(i++)); // заполнение списка целыми числами

    // копирование списка в Li2 (не индивидуальными элементами)
    List Li2 <= new List(); Li2=Li1;
    List Li3 <= new List();

  // удаление каждого индивидуального элемента из левой части  Li2
  // и добавление довление в правую часть Li3 (сохраняя порядок)
    while (Li2.head)
  Li3.addLast(Li2.removeFirst());

// Li2 теперь должен быть пуст
// удаление каждого индивидуального значения из правой
// части  Li3 и добавление в правую часть Li2 (изменение списка)
    while (Li3.head)
  Li2.addLast(Li3.removeLast());

    // Li3 теперь должен быть пуст
    // reverse Li1
    Li1.reverse();

    // проверка, является ли первое значение SIZE
```

```
    if (Li1.head.intValue != SIZE)
die "first item of Li1 != SIZE";

    if(Li1.size!=Li2.size)
die "Li1 and Li2 differ.";

    // возвращение длины списка
    return Li1.size;
}


function main() {
    int n = Arguments.numElements?Arguments[0]:512;
    int result = 0;
    loop(n)
result = test_lists();

    print result;
}
```

pool.tks

```
//

module Main;

class MyPoolEntry {
    String name;
}

function main() {
    Pool pool;
    int i;
    int id;
    MyPoolEntry ce;
    String s;

    pool.template=MyPoolEntry;
    pool.alloc(32);
    loop(10)
{
    id=pool.qAlloc();
    ce<=pool[id];
    ce.name="poolentry_"+id;
}
    i=0;
    foreach id in pool
{
    trace("foreach qFree i=="+i);
    if(++i<5)
{
    pool.qFree(id);   // detach entry, не затрагивая итерированный
список
}
    else
id=-1; // конец foreach
}
    foreach id in pool
{
    ce<=pool[id];
    s<=ce.name;
    trace("foreach entry \""+s+"\"\n");
}
    pool.free();
}
```

preprocessor

```
// ---- файл          : ying.tks
// ---- автор         : (c) 2004 by Bastian Spiegel <bs@tkscript.de>
// ---- информация    : YInG — это Генератор Интерфейса YAC, т.е.
// ---- (YAC Interface Generator).
// ---- YAC добавляет пользовательский RTTI с поддержкой отражения
// ---- языка C++
// ----
// ---- лицензия      : распространяется на условиях
// ---- GNU Общедоступной лицензии
// ----
// ---- изменения     : 9-Feb-2004, 13-Feb-2004, 16-Feb-2004,
// ---- 17-Feb-2004, 19-Feb-2004, 21-Feb-2004, 23-Feb-2004,
// ---- 5-Mar-2004, 23-Mar-2004
// ----
// ----
// ----
module Main;

function UnMangle(String _s) {
    if(_s[0]=='_') {
String r;
_s.substring(r, 1, _s.length-1);
_s=r;
    }
}

function UnMangleR(String _s) {
    if(_s[0]=='_') {
String r;
_s.substring(r, 1, _s.length-1);
return r;
    }
}

String s_srcfile="test.h";
int    b_group=true;
int    b_clid=true;

if(Arguments.numElements<1)
    die("no filename.");

String ca;
foreach ca in Arguments
{
    switch(ca)
{
case "-ng":
case "--nogroup":
    b_group=false;
    break;
case "-nc":
```

```
case "--noclid":
    b_clid=false;
    break;
default:
    s_srcfile=ca;
    break;
}
}


enum {
    YCLASS_DYNAMIC   =0,
    YCLASS_STATIC    =1,
    YCLASS_RESTRICTED=2
};

String s_group="UNTITLED";

int b_method_defreturn=false;
HashTable scanned_classes;

YClass c_class;

function PadSpacesUntilColumn(String _s, int _col) {
    int l=_col-_s.length;
    loop(l) _s.append(" ");
}

function getCurrentDateString() {
    Time t; t.now();
    return
(("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"])[t.week-
day])+", "+t.monthday+"/"+
(("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",
"Oct", "Nov", "Dec"])[t.month])+"/"+t.year;
}

function getCurrentTimeString() {
    Time t;
    Integer io_h;
    Integer io_m;
    Integer io_s;
    t.now();
    io_h.value=t.hour;
    io_m.value=t.min;
    io_s.value=t.sec;
                                                    return
io_h.printf("%02i")+":"+io_m.printf("%02i")+":"+io_s.printf("%02i"
);
}

class YConst {
    int    type;
    String name;
```

```
    String expr;
}

int i_arginitcount;
int b_argvret;
class YArg {
    int    type;  // 0,1,2,3
        String  otype;    //  YAC_String/YAC_ListNode/YAC_TreeNode/
YAC_IntArray/YAC_FloatArray/_MyClass
    String stype;  // String/ListNode/TreeNode/IntArray/FloatArray/
MyClass
    int    isarray;

    getOrigArg() {
switch(type)
    {
    case 0:
    case 1:
    case 2:
return "";
    case 3:
switch(otype)
    {
    default:
      return "("+otype+"*)";
    case "":
return ""; // YAC_Object*
    case "YAC_String":
return "(YAC_String*)";
    case "YAC_Event":
return "(YAC_Event*)";
    case "YAC_ListNode":
return "(YAC_ListNode*)";
    case "YAC_TreeNode":
return "(YAC_TreeNode*)";
    case "YAC_Value":
return "(YAC_Value*)";
    case "YAC_IntArray":
return "(YAC_IntArray*)";
    case "YAC_FloatArray":
return "(YAC_FloatArray*)";
    }
return "";
    }
    }

  init(String _s) {
      String tcl;
      isarray=false;
      otype="";
      _s.trim();  // "mangled" название, например "_Pixmap"
      _s.replace("*","");
      tcl=_s; UnMangle(tcl); //tcl.replace("_","");
// не корректированное название ("Pixmap")
```

```
    switch(_s) {
    case 0:
    case "":
    case "void":
type=0;
return 1;
    case "int":
    case "sSI":
    case "sUI":
type=1;
i_arginitcount++;
return 1;
    case "float":
  case "double":
    case "sF32":
    case "sF64":
type=2;
i_arginitcount++;
return 1;
    case "YAC_Object":
type=3;
i_arginitcount++;
return 1;
    case "YAC_String":
type=3; otype=_s; i_arginitcount++; stype="String";
return 1;
    case "YAC_ValueObject":
type=3; otype=_s; i_arginitcount++; stype="Value";
return 1;
    case "YAC_ListNode":
type=3; otype=_s; i_arginitcount++; stype="ListNode";
return 1;
    case "YAC_TreeNode":
type=3; otype=_s; i_arginitcount++; stype="TreeNode";
return 1;
    case "YAC_IntArray":
type=3; otype=_s; i_arginitcount++; stype="IntArray";
return 1;
    case "YAC_FloatArray":
type=3; otype=_s; i_arginitcount++; stype="FloatArray";
return 1;
    case "YAC_Event":
type=3; otype=_s; i_arginitcount++; stype="Event";
return 1;
    case "YAC_Value":
b_argvret=true;
return 1;
    case "*":
return 1;
    default:
if(!scanned_classes.exists(tcl))
    die("[---] cannot handle non-interface class \""+tcl+"\".");
type=3; otype=_s; stype=tcl;
i_arginitcount++;
```

```
  return 1;
      }
  }

    implement(String _arg) {
String r;
switch(type)
    {
    default:
    case 0:
    case 1:
    case 2:
return "";
    case 3:
switch(otype)
    {
    default:
b_method_defreturn=true;
return "if(YAC_CHK("+_arg+", clid_"+UnMangleR(otype)+"))";
    case "":
return "";
    case "YAC_String":
b_method_defreturn=true;
return "if(YAC_CHK("+_arg+", YAC_CLID_STRING))";
    case "YAC_Value":
b_method_defreturn=true;
return "if(YAC_CHK("+_arg+", YAC_CLID_VALUE))";
    case "YAC_ListNode":
b_method_defreturn=true;
return "if(YAC_CHK("+_arg+", YAC_CLID_LISTNODE))\n";
    case "YAC_TreeNode":
b_method_defreturn=true;
return "if(YAC_CHK("+_arg+", YAC_CLID_TREENODE))";
    case "YAC_IntArray":
b_method_defreturn=true;
return "if(YAC_CHK("+_arg+", YAC_CLID_INTARRAY))";
    case "YAC_FloatArray":
b_method_defreturn=true;
return "if(YAC_CHK("+_arg+", YAC_CLID_FLOATARRAY))";
    }
    }
    }

    getParam() {
switch(type)
    {
    default:
    case 0: return "";
    case 1: return "sSI";
    case 2: return "sF32";
    case 3: return "YAC_Object *";
    }
    }
```

```
    getParam2() {
switch(type)
    {
    default:
    case 0: die("[---] getParam2: cannot handle type="+type+".");
    case 1: return "si";
    case 2: return "f4";
    case 3: return "io";
    }
    }

}

class YProperty : YArg {
    String name;
    //целочисленный тип;

    getTypeString() {
switch(type)
    {
    default:
    case 0:
die("YProperty::getTypeString: <void>");
break;
    case 1:
return "sSI";
    case 2:
return "sF32";
    case 3:
return "YAC_Object *";
    }
    }

    implementGet2();

    implementSet2() {
String r;
String s_prop=name;
s_prop[0]=ucchar(s_prop[0]);
r="void "+c_class.scriptname+"__set"+s_prop;
PadSpacesUntilColumn(r, 37);
r.append("(void *_o, yacmemptr _args");
PadSpacesUntilColumn(r, 78);
r.append(") {");
switch(type)
    {
    case 0:
die("[---] YProperty::implementSet: type==void.");
    case 1:
r.append(("("+c_class.name+"*)_o)->"+name+"=_args.si[0];");
break;
    case 2:
r.append(("("+c_class.name+"*)_o)->"+name+"=_args.f4[0];");
break;
```

```
    case 3:
r.append("if(((("+c_class.name+"*)_o)-
>"+name+")(("+c_class.name+"*)_o)->"+name+"->yacOperatorAs-
sign(_args.io[0]);");
break;
    }
r.append("}\n");
return r;
    }
}

class YFunction {
    String name;
    String mangled_name;
    String script_name;
    YArg   return_type;
    int    return_new;
    YArg   args[];

    YFunction() { return_type.type=0; args.alloc(16); }

    getNumArgs() {
int r=0;
YArg a; foreach a in args
    r+=(a.type!=0);
return r;
    }

    needWrapper() {
if(return_type.type==4)
    return true; // возвращает значения, находящиеся
                 // в дополнительном параметре (YAC_Value *_r)
else
    {
// ---- need wrapper method ? (return value) ----
//   ----   or   can   we   directly   call   the   method   (void
method(int|float..))
int r=0;
YArg a; foreach a in args {
    r=r||((a.type>2)&&(!a.otype.isBlank()));
}
return r;
    }
    }

    /*virtual*/ getWrapperDecl();     // ----
    /*virtual*/ implementWrapper();   // ---- глобальная функция

    appendRegister(String _s);
    appendRegisterArgType(String _s);
    appendRegisterArgOType(String _s);
}

class YMethod : YFunction {
```

```
    getWrapperDecl();
    implementWrapper();
}

class YClass {
    int       type; // 'D' 'R' 'S'
    String    name;
    String    scriptname;
    HashTable methods;
    HashTable properties;
    YConst    constants[];
    String    ordered_methods[];
    String    ordered_properties[];

    //Строка  mangled_name;

    YClass() {
ordered_methods.alloc(512);
ordered_properties.alloc(512);
constants.alloc(8192);
    }
}
String    s_class;
YMethod   c_method;

HashTable classes; classes.alloc(512);
String    ordered_classes[512];
YFunction functions[512];

String    yclass_typenames[]<=["","S","R"];

YFunction::appendRegisterArgType {
    String s="\tstatic const sUI   "+mangled_name+"_arg_types";
    PadSpacesUntilColumn(s, 64);
    s.append("[]={");
    if(args.numElements)
{
    YArg a;
    int b_addsep=false;
    foreach a in args {
if(b_addsep)
    s.append(", ");
else
    b_addsep=true;
s.append(tcstring(a.type));
    }
    s.append(",};\n");
}
    else
s.append("0,};\n");
    _s.append(s);
}

YFunction::appendRegisterArgOType {
```

```
    String s="\tstatic const char *"+mangled_name+"_arg_otypes";
    PadSpacesUntilColumn(s, 64);
    s.append("[]={");
    if(args.numElements)
{
    int b_addsep=false;
    YArg a;
    foreach a in args {
if(b_addsep)
    s.append(", ");
else
    b_addsep=true;
s.append("\""+a.otype+"\"");
    }
    s.append(",};\n");
}
    else
s.append("0,};\n");
    _s.append(s);
}

YFunction::appendRegister {
    int i_callstyle;
    if(needWrapper())
{
    if(return_type.type==4)
i_callstyle=(args.numElements==0)+4;
    else
i_callstyle=return_type.type;
}
    else
i_callstyle=8+(4*args.numElements)+return_type.type;
    _s.append("\t_host->yacRegisterFunction((void*)"+mangled_name
      +", \""+name+"\", "
      +return_type.type
      +", \""+return_type.otype+"\", "
      +args.numElements
      +", "+mangled_name+"_arg_types, "+mangled_name+"_arg_otypes,
"+i_callstyle+");\n");
}

YFunction::getWrapperDecl {
    if(needWrapper())
{
    String r; r.alloc(1024);
    switch(return_type.type)
{
case 0: r="YAC_APIC void "; break;
case 1: r="YAC_APIC sSI  "; break;
case 2: r="YAC_APIC sF32 "; break;
case 3: r="YAC_APIC void*"; break;
case 4: r="YAC_APIC void "; break;
}
    r.append(mangled_name);
```

```
    PadSpacesUntilColumn(r, 37);
    r.append("(");

    if(args.numElements)
{
    r.append("yacmemptr");
    if(return_type.type==4)
r.append(", YAC_Value *");
}
    else
if(return_type.type==4)
    r.append("YAC_Value *");
    PadSpacesUntilColumn(r, 68);
    return r+");\n";
}
    else
return "";
}

YMethod::getWrapperDecl {
    String r; r.alloc(1024);
    switch(return_type.type)
{
case 0: r="void "; break;
case 1: r="sSI  "; break;
case 2: r="sF32 "; break;
case 3: r="void*"; break;
case 4: r="void "; break;
}
    r.append(c_class.scriptname+"__"+name);
    PadSpacesUntilColumn(r, 37);
    r.append("(void *");

    if(args.numElements)
    r.append(", yacmemptr");
    if(return_type.type==4)
r.append(", YAC_Value *");
    PadSpacesUntilColumn(r, 68);
    return r+");\n";
}

YFunction::implementWrapper() {
    if(needWrapper())
{
    //трассировка "YFunction::implementWrapper";
    String r; r.alloc(1024);
    String arglist="";

    switch(return_type.type)
{
case 0: r="void "; break;
case 1: r="sSI  "; break;
case 2: r="sF32 "; break;
case 3: r="void*"; break;
```

```
case 4: r="void "; break;
}

    r.append(mangled_name);
    PadSpacesUntilColumn(r, 37);
    r.append("(");
    int i_arg=0;
    int b_addsep=false;
    YArg a;
    foreach a in args
{
    if(a.type)
{
    if(b_addsep)
arglist.append(", ");
    else
{
    r.append("yacmemptr _args");
    b_addsep=true;
}

                                                arglist.append(a.get-
OrigArg()+"_args."+a.getParam2()+"["+i_arg+"]");
    i_arg++;
}
}
    if(return_type.type==4)
{
    if(b_addsep)
{
    r.append(", ");
    arglist.append(", ");
}
    r.append("YAC_Value *_r");
    arglist.append("_r");
}
    PadSpacesUntilColumn(r, 78);
    r.append(") {");

    b_method_defreturn=false;
    // ---- декодирование параметров ----
    i_arg=0;
    foreach a in args
{
    if(a.type)
{
    r.append(a.implement("_args."+a.getParam2()+"["+i_arg+"]"));
    i_arg++;
}
}
    if(b_method_defreturn) r.append("{ ");

    // ---- вызов метода ----
    switch(return_type.type)
```

```
{
case 1: r.append("return (sSI)"); break;
case 2: r.append("return (sF32)"); break;
case 3: r.append("return (void*)"); break;
}
    r.append(name+"("+arglist+");");
    if(b_method_defreturn) {
r.append("}");
switch(return_type.type)
    {
    case 1:
r.append("return 0;");
break;
    case 2:
r.append("return 0.0f;");
break;
    case 3:
r.append("return (void*)0;");
break;
    }
    }
    return r+"}\n";
}
    else
{
    return "";
}
}

YMethod::implementWrapper() {
    String r; r.alloc(1024);

    String arglist="";

    switch(return_type.type)
      {
      case 0: r="void "; break;
      case 1: r="sSI  "; break;
      case 2: r="sF32 "; break;
      case 3: r="void*"; break;
      case 4: r="void "; break;
      }

    r.append(c_class.scriptname+"__"+name);
    PadSpacesUntilColumn(r, 37);
    r.append("(void *_o");
    int i_arg=0;
    int b_addsep=false;
    YArg a;
    foreach a in args
{
    if(a.type)
{
    if(b_addsep)
```

```
    arglist.append(", ");
    else
{
    r.append(", yacmemptr _args");
    b_addsep=true;
}
                                                arglist.append(a.get-
OrigArg()+"_args."+a.getParam2()+"["+i_arg+"]");
    i_arg++;
}
}
    if(return_type.type==4)
{
    r.append(", YAC_Value *_r");
    if(b_addsep)
arglist.append(", ");
    arglist.append("_r");
}
    PadSpacesUntilColumn(r, 78);
//      l=78-r.length;
//      if(l>0) loop(l) r.append(" ");
    r.append(") {");

    b_method_defreturn=false;
    // ---- декодирование параметров ----
    i_arg=0;
    foreach a in args
{
    if(a.type)
{
    r.append(a.implement("_args."+a.getParam2()+"["+i_arg+"]"));
    i_arg++;
}
}
    if(b_method_defreturn) r.append("{ ");

    // ---- вызов метода ----
    switch(return_type.type)
      {
      case 1: r.append("return (sSI)"); break;
      case 2: r.append("return (sF32)"); break;
      case 3: r.append("return (void*)"); break;
      }
    r.append(("("+c_class.name+"*)_o)->"+name+"("+arglist+");");
    if(b_method_defreturn) {
      r.append("}");
      switch(return_type.type)
{
case 1:
  r.append("return 0;");
  break;
case 2:
  r.append("return 0.0f;");
  break;
```

```
case 3:
  r.append("return (void*)0;");
  break;
}
    }
    return r+"}\n";
}

function AddDynamicClass() {
    //трассировка "AddClass(\""+s_class+"\")";
    c_class<=new YClass;
    classes[s_class]=deref c_class;
    c_class.type=YCLASS_DYNAMIC;
    //c_class.name="__YAC_"+s_class;
    c_class.name=s_class;
    c_class.scriptname=s_class;    UnMangle(c_class.scriptname);//
.replace("_","");
    ordered_classes.add(s_class);
}

function AddMethod(String _name, String _return, ListNode _args) {
  _name.replace("*","");
  c_method<=new YMethod;
  c_method.name=_name;
  _Debug();
  String s_scriptname=_name;
  UnMangle(s_scriptname); //.replace("_","");
  c_method.script_name=s_scriptname; // e.g. "_delete" => "delete"
  //трассировка  "c_method.script_name="+c_method.script_name;
  HashTable h<=c_class.methods;
  h[_name]=deref c_method;
  c_class.ordered_methods.add(_name);
  _return.replace("*","");
  YArg c_return<=c_method.return_type;
  if(!c_return.init(_return))
    die("[---] cannot handle return type \""+_return+"\"\n");
  i_arginitcount=0;
  b_argvret=false;
  Value v; foreach v in _args {
               //трассировка   "AddMethod   arg["+i_arginitcount+"]
type="+v.type+" string="+v.string;
      if(v.type)
  if(!c_method.args[i_arginitcount].init(v.stringValue))
     die("[---] YMethod: cannot handle argument "+i_arginitcount+"
type \""+v.stringValue+"\"\n");
  }
  if(b_argvret) c_return.type=4;
  c_method.args.setNumElements(i_arginitcount);
}

function AddFunction(String _name, String _return, ListNode _args)
{
                  //трассировка     "AddFunction(name=\""+_name+"\",
return=\""+_return+"\" args="+_args+")";
```

```
    _name.replace("*","");
    YFunction f<=functions.nextFree;
    f.name=_name;
    UnMangle(f.name);
    _return.replace("*","");
    YArg f_return<=f.return_type;
    if(!f_return.init(_return))
die("[---]    YFunction:    cannot    handle    return    type
\""+_return+"\"\n");
    i_arginitcount=0;
    b_argvret=false;
    Value v; foreach v in _args {
//trace   "AddFunction   arg["+i_arginitcount+"]   type="+v.type+"
string="+v.string;
if(v.type)
    if(!f.args[i_arginitcount].init(v.stringValue))
die("[---] YFunction: cannot handle argument "+i_arginitcount+"
type \""+v.stringValue+"\"\n");
    }
    f.args.setNumElements(i_arginitcount);
    if(b_argvret) f_return.type=4;
    if(f.needWrapper())
f.mangled_name="__APIC__"+_name;
    else
f.mangled_name=_name;
}

function AddProperty(String _name, String _type) {
  HashTable h<=c_class.properties;

  _name.replace("*", "");
  _name.trim();
  YProperty p<=new YProperty();
  h[_name]=deref p;
  c_class.ordered_properties.add(_name);
  p.name=_name;
  if(!p.init(_type))
    die("[---] cannot handle property type \""+_type+"\"");
}

function AddConstant(String _name, String _expr, int _type) {
//        trace "AddConstant(name=\""+_name+"\" expr=\""+_expr+"\"
type="+_type+")";
    YConst c<=c_class.constants.nextFree;
    //трассировка "c="+c;
    UnMangle(_name); //_name.replace("_","");
    c.name=_name;
    c.type=_type;
    c.expr=_expr;
}

String s_cpp;
//  String s_h;
String s_group_h;
```

```
function CPPOut() {
  String s_clid_def="";
  String tcl;
  String s_classfdecl="";
  s_cpp.alloc(16*1024); s_cpp.empty();

  s_cpp.append("// ---- автогенерация при помощи YInG
("+getCurrentDateString()+" "+getCurrentTimeString()+")\n");
          s_cpp.append("#ifndef     __YAC__"+s_group+"_h__\n#define
__YAC__"+s_group+"_h__\n\n");

  s_group_h.alloc(16*1024); s_group_h.empty();
   s_group_h.append("// ---- ying_"+s_group+".h: автогенерация при
помощи YInG ("+
       getCurrentDateString()+" "+getCurrentTimeString()+")\n");

  String scl;
  YClass c;
  foreach scl in ordered_classes {
    c<=classes[scl];
    c_class<=c;
    tcl=c.name; //"__YAC_"+scl;

    String s_cl;
//      String s_decl; s_decl.alloc(4*1024); s_decl.empty();
    String s_cdecl; s_cdecl.alloc(4*1024); s_cdecl.empty();
//      String s_def; s_def.alloc(4*1024); s_def.empty();
    String s_cdef; s_cdef.alloc(4*1024); s_cdef.empty();

    s_cl.alloc(16*1024); s_cl.empty();
    s_cdecl.append("// ---- ying_"+s_group+"_"+c.scriptname+".cpp:
автогенерация при помощи YInG ("+
       getCurrentDateString()+" "+getCurrentTimeString()+")\n\n");

    // ---- запись объявления /экспорта метода ----
    s_classfdecl.append("class "+tcl+";\n");
    int i_method=1;
    int i_member=0;
    String s_method_pfx;
    String s_method;
    int i_maxmethodlen=1;
    foreach s_method in c.ordered_methods {
      c_method<=c.methods[s_method];
      if(c_method.name.length>i_maxmethodlen)
i_maxmethodlen=c_method.name.length;
    }
    foreach s_method in c.ordered_methods {
      c_method<=c.methods[s_method];
      s_cdecl.append(c_method.getWrapperDecl());
    }
    String s_property;

    YProperty p;
```

```
                String    s_prop_impl;    s_prop_impl.alloc(4*1024);
s_prop_impl.empty();
    foreach s_property in c.ordered_properties {
      p<=c.properties[s_property];
      String s_prop=p.name;
      s_prop[0]=ucchar(s_prop[0]);
      int l;
      if(!c.methods.exists("get"+s_prop))
{
  i_method++;
  String s_get_decl;
  switch(p.type)
    {
    case 0: s_get_decl="void "; break;
    case 1: s_get_decl="sSI  "; break;
    case 2: s_get_decl="sF32 "; break;
    case 3: s_get_decl="void*"; break;
    }
  s_get_decl.append(tcl+"__get"+s_prop);
  PadSpacesUntilColumn(s_get_decl, 37);
  s_get_decl.append("(void *");
  PadSpacesUntilColumn(s_get_decl, 68);
  s_cdecl.append(s_get_decl+");\n");
  // ---- implement property get method ----
  s_prop_impl.append(p.implementGet2());
}
      if(!c.methods.exists("set"+s_prop))
{
  i_method++;
  String s_set_decl="void "+tcl+"__set"+s_prop;
  PadSpacesUntilColumn(s_set_decl, 37);
  s_set_decl.append("(void *, yacmemptr");
  PadSpacesUntilColumn(s_set_decl, 68);
  s_cdecl.append(s_set_decl+");\n");
  // ---- implement property set method ----
  s_prop_impl.append(p.implementSet2());
}
      i_member++;
    }
    if(b_clid)
s_cdef.append("YAC_C("+tcl+", \""+c.scriptname+"\");\n\n");
    s_group_h.append("extern sUI clid_"+c.scriptname+";\n");
                                                s_cpp.append("sUI
clid_"+c.scriptname+";\n");
       String  s_template="YAC_"+yclass_typenames[c.type]+"Template
<"+tcl+">";
    PadSpacesUntilColumn(s_template, 48);
    s_cpp.append(s_template+"*t_"+c.scriptname+";\n");

       s_cl.append("//  ------------------  YAC  class  \""+tcl+"\"
reflection map (implementation) ------------------\n");
    if(i_member)
{
    String s_cl_member_name="";
```

```
    String s_cl_member_type="";
    String s_cl_member_objecttype="";
    String s_cl_member_offset="";
    int b_addsep=false;
    foreach s_property in c.ordered_properties {
p<=c.properties[s_property];
if(b_addsep)
    {
s_cl_member_name.append(", ");
s_cl_member_type.append(", ");
s_cl_member_objecttype.append(", ");
s_cl_member_offset.append(", ");
    }
else b_addsep=true;
s_cl_member_name.append("\""+p.name+"\"");
s_cl_member_type.append(tcstring(p.type));
if(!p.stype.isBlank())
    s_cl_member_objecttype.append("\""+p.stype+"\"");
else
    s_cl_member_objecttype.append("(const char*)0");

s_cl_member_offset.append("(((sSI)&"+p.name+")-((sSI)this))");
    }
        s_cl.append("sUI                    "+tcl+"::yacMemberGet-
Num                 (void) {return "+i_member+";}\n");
            s_cl.append("const   char   **"+tcl+"::yacMemberGet-
Names                                   (void)  {static  const
char*r[]={"+s_cl_member_name+"};return r; }\n");
          s_cl.append("const   sUI        *"+tcl+"::yacMemberGet-
Types                                 (void)  {static  const  sUI
r[]={"+s_cl_member_type+"}; return r; }\n");
         s_cl.append("const   char   **"+tcl+"::yacMemberGetObject-
Types                                   (void)   {static   const
char*r[]={"+s_cl_member_objecttype+"};return r; }\n");
         s_cl.append("const   sUI       *"+tcl+"::yacMemberGetOff-
sets                                  (void)  {static  const  sUI
r[]={"+s_cl_member_offset+"};return r;}\n");
}
    else
{
        s_cl.append("sUI                    "+tcl+"::yacMemberGet-
Num                 (void) {return 0;}\n");
            s_cl.append("const   char   **"+tcl+"::yacMemberGet-
Names               (void) {return 0;}\n");
          s_cl.append("const   sUI        *"+tcl+"::yacMemberGet-
Types               (void) {return 0;}\n");
         s_cl.append("const   char   **"+tcl+"::yacMemberGetObject-
Types          (void) {return 0;}\n");
         s_cl.append("const   sUI       *"+tcl+"::yacMemberGetOff-
sets              (void) {return 0;}\n");
}


    if(i_method||c.ordered_methods.numElements)
```

```
{
    String s_cl_method_name="\"operator\"";
    String s_cl_method_numparam="2";
    String s_cl_method_param_types="";
    String s_cl_method_param_typesd="";
    String s_cl_method_param_otypes="";
    String s_cl_method_param_otypesd="";
    String s_cl_method_return_type="4";
    String s_cl_method_return_otype="\"\"";
    String s_cl_method_adr="(void*)Object__operator";
    String s_cl_method_adr2="(void*)0";
    int i_typesd=1;

            s_cl_method_param_typesd.append("static   const   sUI
rt0[]={1,3};");
    s_cl_method_param_types.append("rt0");
          s_cl_method_param_otypesd.append("static   const   char
*rs0[]={\"\",};");
    s_cl_method_param_otypes.append("rs0");

    foreach s_method in c.ordered_methods {
i_method++;
c_method<=c.methods[s_method];
s_cl_method_name.append(", ");
s_cl_method_numparam.append(", ");
s_cl_method_param_types.append(", ");
s_cl_method_param_otypes.append(", ");
s_cl_method_return_type.append(", ");
s_cl_method_return_otype.append(", ");
s_cl_method_param_typesd.append("static          const          sUI
rt"+i_typesd+"[]=");
s_cl_method_param_otypesd.append("static         const         char
*rs"+i_typesd+"[]=");
s_cl_method_numparam.append(tcstring(c_method.args.numElements));
s_cl_method_name.append("\""+c_method.script_name+"\"");

s_cl_method_return_type.append(tcstring(c_method.return_type.type)
);
if(!c_method.return_type.stype.isBlank())

s_cl_method_return_otype.append("\""+c_method.return_type.stype+"\
"");
else
    s_cl_method_return_otype.append("(const char*)0");
s_cl_method_adr.append(",                        (void*)"+c.script-
name+"__"+c_method.name);
s_cl_method_adr2.append(",               (void*)__JIT__"+c.script-
name+"__"+c_method.name);
int b_addsep3=false;
if(c_method.args.numElements)
    {
s_cl_method_param_typesd.append("{");
s_cl_method_param_otypesd.append("{");
YArg a; foreach a in c_method.args {
```

```
    if(b_addsep3)
{
    s_cl_method_param_typesd.append(", ");
    s_cl_method_param_otypesd.append(", ");
}
    else
b_addsep3=true;
    s_cl_method_param_typesd.append(tcstring(a.type));
    if(!a.stype.isBlank())
s_cl_method_param_otypesd.append("\""+a.stype+"\"");
    else
s_cl_method_param_otypesd.append("(const char*)0");
}
s_cl_method_param_typesd.append(",};");
s_cl_method_param_otypesd.append(",};");
    }
else
    {
s_cl_method_param_typesd.append("{0,};");//
s_cl_method_param_otypesd.append("{(const char*)0,};");
    }
s_cl_method_param_types.append("rt"+(i_typesd));
s_cl_method_param_otypes.append("rs"+(i_typesd++));
    }
    foreach s_property in c.ordered_properties {
p<=c.properties[s_property];
s_prop=p.name;
s_prop[0]=ucchar(s_prop[0]);
if(!c.methods.exists("get"+s_prop))
    {
s_cl_method_name.append(", \"get"+s_prop+"\"");
s_cl_method_numparam.append(", 0");
s_cl_method_param_typesd.append("static          const          sUI
rt"+i_typesd+"[]={0,};");
s_cl_method_param_otypesd.append("static        const        char
*rs"+i_typesd+"[]={\"\", };");
s_cl_method_param_types.append(", rt"+(i_typesd));
s_cl_method_param_otypes.append(", rs"+(i_typesd++));
s_cl_method_return_type.append(", "+p.type);
s_cl_method_return_otype.append(", \""+p.stype+"\"");
s_cl_method_adr.append(", (void*)"+tcl+"__get"+s_prop);
//i_method++;
    }
if(!c.methods.exists("set"+s_prop))
    {
s_cl_method_name.append(", \"set"+s_prop+"\"");
s_cl_method_numparam.append(", 1");
s_cl_method_param_typesd.append("static          const          sUI
rt"+i_typesd+"[]={"+p.type+",};");
s_cl_method_param_otypesd.append("static        const        char
*rs"+i_typesd+"[]={\""+p.stype+"\", };");
s_cl_method_param_types.append(", rt"+(i_typesd));
s_cl_method_param_otypes.append(", rs"+(i_typesd++));
s_cl_method_return_type.append(", 0");
```

```
s_cl_method_return_otype.append(", \"\"");
s_cl_method_adr.append(", (void*)"+tcl+"__set"+s_prop);
//i_method++;
    }
    }

        s_cl.append("sUI                    "+tcl+"::yacMethodGet-
Num                 (void) {return "+i_method+";}\n");
             s_cl.append("const   char   **"+tcl+"::yacMethodGet-
Names                                  (void)  {static  const  char
*r[]={"+s_cl_method_name+"}; return r;}\n");
       s_cl.append("const  sUI     *"+tcl+"::yacMethodGetNumParame-
ters                               (void)   {static   const   sUI
r[]={"+s_cl_method_numparam+"}; return r;}\n");
       s_cl.append("const  sUI    **"+tcl+"::yacMethodGetParameter-
Types        (void) {"+s_cl_method_param_typesd+"static const sUI
*r[]={"+s_cl_method_param_types+"}; return r;}\n");
       s_cl.append("const  char***"+tcl+"::yacMethodGetParameterOb-
jectTypes   (void)   {"+s_cl_method_param_otypesd+"static   const
char**r[]={"+s_cl_method_param_otypes+"}; return r;}\n");
        s_cl.append("const  sUI      *"+tcl+"::yacMethodGetReturn-
Types                                (void)   {static   const   sUI
r[]={"+s_cl_method_return_type+"}; return r;}\n");
      s_cl.append("const char **"+tcl+"::yacMethodGetReturnObject-
Types                        (void)     {static     const     char
*r[]={"+s_cl_method_return_otype+"}; return r;}\n");
                  s_cl.append("const    void    **"+tcl+"::yac-
MethodGetAdr                                (void) {static const void
*r[]={"+s_cl_method_adr+"}; return r;}\n");
}
    else
{
        s_cl.append("sUI                    "+tcl+"::yacMethodGet-
Num                 (void) {return 1;}\n");
             s_cl.append("const   char   **"+tcl+"::yacMethodGet-
Names              (void) {static const char *r[]={\"operator\",};
return r;}\n");
       s_cl.append("const  sUI     *"+tcl+"::yacMethodGetNumParame-
ters        (void) {return {2,};\n");
       s_cl.append("const  sUI    **"+tcl+"::yacMethodGetParameter-
Types             (void)  {static  const  sUI  r0[]={1,3,};  return
{r0,};}\n");
       s_cl.append("const  char***"+tcl+"::yacMethodGetParameterOb-
jectTypes   (void)   {static   const   char   *r0[]={\"\",};   return
{r0,};}\n");
        s_cl.append("const  sUI      *"+tcl+"::yacMethodGetReturn-
Types          (void) {return {4,};}\n");
      s_cl.append("const char **"+tcl+"::yacMethodGetReturnObject-
Types    (void) {return {\"\",};}\n");
                  s_cl.append("const    void    **"+tcl+"::yac-
MethodGetAdr                                  (void)  {static  const
void*r[]={(void*)Object__operator,}; return r;}\n");
}
```

```
    if(c.constants.numElements)
{
    String s_const;
    String s_const_name="";
    String s_const_type="";
    String s_const_value="";
    b_addsep3=false;
    int i_const=0;
        YConst co; foreach co in c.constants {
if(b_addsep3)
    {
s_const_name.append(", ");
s_const_type.append(", ");
    }
else
    b_addsep3=true;
s_const_name.append("\""+co.name+"\"");
s_const_type.append(tcstring(co.type));
if(co.type==1)
    s_const_value.append("m.si["+i_const+"]=(sSI)"+co.expr+";");
else
    s_const_value.append("m.f4["+i_const+"]=(sF32)"+co.expr+";");
i_const++;
    }
        s_cl.append("sUI                    "+tcl+"::yacConstantGet-
Num                 (void) {return "+i_const+";}\n");
            s_cl.append("const   char   **"+tcl+"::yacConstantGet-
Names                                   (void)   {static   const
char*r[]={"+s_const_name+"}; return r;}\n");
          s_cl.append("const   sUI        *"+tcl+"::yacConstantGet-
Types               (void) {static const sUI r[]={"+s_const_type+"};
return r;}\n");
         s_cl.append("yacmemptr         "+tcl+"::yacConstantGetVal-
ues                (void) {static sUI r["+i_const+"]; yacmemptr m;
m.ui=r; "+s_const_value+"; return m;}\n");
}
    else
{
        s_cl.append("sUI                    "+tcl+"::yacConstantGet-
Num                 (void) {return 0;}\n");
            s_cl.append("const   char   **"+tcl+"::yacConstantGet-
Names                   (void) {static const char*r[]={\"\"}; return
r;}\n");
          s_cl.append("const   sUI        *"+tcl+"::yacConstantGet-
Types               (void) {static const sUI r[]={0}; return r;}\n");
         s_cl.append("yacmemptr         "+tcl+"::yacConstantGetVal-
ues                     (void)  {static  sUI  r[]={0,};yacmemptr  m;
m.ui=(sUI*)r; return m;}\n");
}
//      s_h.append(s_decl+"\n");
//      s_h.append(s_def);
    s_cl=s_cdecl+"\n"+s_cl+"\n"+s_cdef;
```

```
    // ---- implement wrapper methods ----
    foreach s_method in c.ordered_methods {
c_method<=c.methods[s_method];
s_cl.append(c_method.implementWrapper());
    }

    s_cl.append(s_prop_impl);

    s_cl.saveLocal("ying_"+s_group+"_"+c.scriptname+".cpp");
  }

  s_group_h.append(s_classfdecl);
  s_group_h.append(s_clid_def);

     s_cpp.append("\nvoid  YAC_Init_"+s_group+"(YAC_Host  *_host)
{\n\t// ------------------------------- классы ----------------
----------------\n");
  foreach scl in ordered_classes {
    c<=classes[scl];
    tcl=c.name;
    //tcl="__YAC_"+scl;
    String s_newtemp="\t  t_"+c.scriptname;
    PadSpacesUntilColumn(s_newtemp, 32);
                                    s_cpp.append(s_newtemp+"=new
YAC_"+yclass_typenames[c.type]+"Template <"+tcl+">(_host);\n");
    s_newtemp="\tclid_"+c.scriptname;
    PadSpacesUntilColumn(s_newtemp, 32);
     s_cpp.append(s_newtemp+"=                  t_"+c.scriptname+"-
>ctemplate->class_ID;\n");
  }
  s_cpp.append("\t// ------------------------------- функции ----
--------------------------\n$(YAC_REGISTER_FUN)}\n");

     s_cpp.append("\nvoid  YAC_Exit_"+s_group+"(YAC_Host  *_host)
{\n");
  // ---- удаление в обратном порядке ---
  int i=ordered_classes.numElements;
  while(--i>=0) {
      scl=ordered_classes[i];
      c<=classes[scl];
      s_cpp.append("\tdelete t_"+c.scriptname+";\n");
  }
  s_cpp.append("}\n");

  //s_cpp.append("\n#include <yac_host.cpp>\n");

  // ---- процесс привязывния функций "C" ----
  String s_fun="";
  String s_fun_arg_types="";
  String s_fun_arg_otypes="";
  YFunction f;
  foreach f in functions {
      s_group_h.append(f.getWrapperDecl());
      s_cpp.append(f.implementWrapper());
```

```
      f.appendRegister(s_fun);
      f.appendRegisterArgType(s_fun_arg_types);
      f.appendRegisterArgOType(s_fun_arg_otypes);
  }
                                  s_cpp.replace("$(YAC_REGISTER_FUN)",
s_fun_arg_types+s_fun_arg_otypes+s_fun);

  s_cpp.append("\n\n#endif\n");
  if(b_group)
      {
  s_cpp.saveLocal("ying_"+s_group+".cpp");
  s_group_h.saveLocal("ying_"+s_group+".h");
      }

  trace "\n-------------------------------------\n";//+s_cpp;
}

YProperty::implementGet2() {
    String r;
    String s_prop=name;
    s_prop[0]=ucchar(s_prop[0]);
    switch(type)
      {
      case 0: r="void "; break;
      case 1: r="sSI  "; break;
      case 2: r="sF32 "; break;
      case 3: r="void*"; break;
      }
    r.append(c_class.scriptname+"__get"+s_prop);
    PadSpacesUntilColumn(r, 37);
    r.append("(void *_o");
    PadSpacesUntilColumn(r, 78);
    r.append(") {");
    switch(type)
      {
      case 1: r.append("return (sSI)"); break;
      case 2: r.append("return (sF32)"); break;
      case 3: r.append("return (void*)"); break;
      }
    r.append(("("+c_class.name+"*)_o)->"+name+";");
    r.append("}\n");
    return r;
}

// ------------------------------------------------------------------
----------------------------------------

String s_in;

if(!s_in.loadLocal(s_srcfile, true))
    die "[---] failed to load input file \""+s_srcfile+"\".";
s_in.split('\n');
String cl;
int i_cl=1;
```

```
int b_class=false;
int i,j;
String st,st2,st3,st4;

// ---- сканирование классов ----
foreach cl in s_in {
    i=cl.indexOf("YC class",0);
    if(i!=-1)
{
    // ---- распаковка названия класса ----
    cl.substring(st, i+8, cl.length-(i+8));
    String tcl=st.splitSpace(0)[0];
    tcl.replace(":",""); // "class MyClass: public Bla"
    UnMangle(tcl); //.replace("_","");
    scanned_classes[tcl]=true;
    trace "scanned class \""+tcl+"\"";
}
}

foreach cl in s_in {
    ListNode l,lc;
    String s_arg;
    //Integer io; io.value=i_cl;
    //trace "["+io.printf("%3i")+"]: "+cl;
    i_cl++;
    i=cl.indexOf("YG(\"",0);
    if(i!=-1)
{
    // ---- распаковка имени группы ----
    j=cl.indexOf("\")", i+4);
    if(j!=-1)
{
    cl.substring(s_group, i+4, j-(i+4));
    trace "[...] current group set to \""+s_group+"\".";
}
    else
die("[---] error parsing YG(\"\") group name definition.");
}
    else
{
    i=cl.indexOf("YC class",0);
    if(i!=-1)
{
    // ---- распаковка имени класса ----
    cl.substring(st, i+8, cl.length-(i+8));
    s_class=st.splitSpace(1)[0];
    AddDynamicClass();
    b_class=true;
}
    else
{
    st=cl; st.trim();
    if(st.startsWith("YM "))
{
```

```
    // ---- синтаксический анализ метода ----
    // YM void test(void);
    // YM int test(void);
    // YM void test(int _i, int _j, int _k)
    // YM YAC_ListNode * test(void)
    st.replace("//","");
    st.words(false);
    st2=st.getWord(2);
    int i_name;
    if(st2=="*")
{
    st2=st.getWord(3);
    i_name=4;
}
    else
i_name=3;
    i=st2.indexOf("("); // void name(void)
    if(i==-1)
{
    st2.append(st.getWord(i_name)); // void name (void)
    i=st2.indexOf("(");
}
    st2.substring(st3, 0, i); //get name ("test")
    i=st.indexOf("(");
    j=st.indexOf(")");
    st.substring(st4, i+1, j-(i+1)); // st4=args ("void","int")
    l<=new ListNode;
    lc<=l;
    foreach s_arg in st4.splitChar(',')
lc<=lc.appendValue(#(tcstring(s_arg.splitSpace(0)[0])));
    AddMethod(st3, st.getWord(1), l);
    l<=0;
    st.freeStack();
}
    else
{
    if(st.startsWith("YP "))
{
    // ---- синтаксический анализ свойств ----
    // YP sSI myint;
    // YP sF32 myfloat;
    //
    st.replace("//","");
    st.words(false);
    st2=st.getWord(2);
    i=st2.indexOf(";");
    st2.substring(st3, 0, i);
    AddProperty(st3, st.getWord(1));
    st.freeStack();
}
    else
{
    if(st.startsWith("#define ")||st.startsWith("//#define "))
{
```

```
    // ---- синтаксический анализ констант ----
    // #define YCI TEST_INT_CONSTANT 42
    // //#define YCI EXPORT_INT_CONSTANT 42
    // #define YCF TEST_INT_CONSTANT 42.42
    // //#define YCF EXPORT_INT_CONSTANT 42.42
    //
    st.words(false);
    if(st.getWord(2)=="YCI")
AddConstant(st.getWord(1), st.getWord(3),1);
    else if(st.getWord(2)=="YCF")
AddConstant(st.getWord(1), st.getWord(3),2);
    st.freeStack();
}
    else
if(st.startsWith("YF "))
    {
// ---- синтаксический анализ функций ----
// YF void test(void);
// YF int test(void);
// YF void test(int _i, int _j, int _k)
st.replace("//","");
st.words(false);
st2=st.getWord(2);
i=st2.indexOf("("); // void name(void)
if(i==-1)
    {
st2.append(st.getWord(3)); // void name (void)
i=st2.indexOf("(");
    }
st2.substring(st3, 0, i); //get name ("test")
i=st.indexOf("(");
j=st.indexOf(")");
st.substring(st4, i+1, j-(i+1)); // st4=args ("void","int")
l<=new ListNode;
lc<=l;
foreach s_arg in st4.splitChar(',')
    lc<=lc.appendValue(#(tcstring(s_arg.splitSpace(0)[0])));
AddFunction(st3, st.getWord(1), l);
l<=0;
st.freeStack();
    }

}
}
}
}
}
s_in.freeStack();

CPPOut();

trace "[...] found "+classes.numElements+" classes.";

trace ".";
```

testlang.tks

```
// требуется версия tkscript >=0.8.0.8

int ver=TKS.getVersion();
trace "using TKS version "
+(ver>>24)+"."+((ver>>16)&0xFF)+"."+((ver>>8)&0xFF)+"."+(ver&255);

#define MYCONST_A "test123"
#define MYCONST_B 42
#define MYCONST_C 3.14
enum { MYCONST_D, MYCONST_E, MYCONST_F };

String s=TKS.stringToConstant("MYCONST_A");
trace "s=\""+s+"\"";

s=TKS.constantToString(MYCONST_B, "MYCONST_");
trace "s=\""+s+"\"";

trace TKS.getClassName(s);

// ---- динамический поиск метода и его вычисление ----
s="hello, ";
Value v_ret;
if(TKS.evalMethodByName(s, "append", {"world."}, v_ret))
{
    trace "v_ret="+v_ret.string; // => void
    trace "s=\""+s+"\""; // => "hello, world."
}
else
{
    die "TKS.evalMethodByName() failed.";
}



class MyClass {
    int mc_int;
    float mc_float;
    String mc_string;
}

MyClass mco;

mco.mc_int=64;
mco.mc_float=1.23456789;
mco.mc_string="test123";

trace "mco.mc_int="+mco.mc_int;
trace "mco.mc_float="+mco.mc_float;
trace "mco.mc_string="+mco.mc_string;
```

```
/** вывод типа объекта, названия класса и количества свойств. **/
trace   TKS.getClassName(mco)+"   "+TKS.getPropertyClassName(mco)+"
has "+TKS.getNumProperties(mco)+" properties.";

/** считывание свойств метода, используя интерфейс свойства YAC **/
trace "mco.mc_int="+TKS.getPropertyByName(mco, "mc_int");
trace "mco.mc_float="+TKS.getPropertyByName(mco, "mc_float");
trace "mco.mc_string="+TKS.getPropertyByName(mco, "mc_string");

/** установка свойств класса, используя интерфейс свойства YAC **/
TKS.setIntPropertyByName(mco, "mc_int", 42);
TKS.setFloatPropertyByName(mco, "mc_float", 3.1415);
TKS.setObjectPropertyByName(mco, "mc_string", "hello, world");

/** еще одна проверка свойств класса **/
trace "2mco.mc_int="+TKS.getPropertyByName(mco, "mc_int");
trace "2mco.mc_float="+TKS.getPropertyByName(mco, "mc_float");
trace "2mco.mc_string="+TKS.getPropertyByName(mco, "mc_string");

int mco_class_ID=TKS.getClassID(mco);
trace "user defined object uses class_ID "+mco_class_ID;

/** создание нового экземпляра типа "MyClass" **/
Pointer mco2 <=TKS.newObjectByName("MyClass");
trace  TKS.getClassName(mco2)+"  "+TKS.getPropertyClassName(mco2)+"
has "+TKS.getNumProperties(mco2)+" properties.";
```

testxml.tks

```
String s;

s.loadLocal("input.xml", 1);
trace "s="+s;

TreeNode troot<=s.parseXML();
trace "troot="+troot;
if(troot)
{
    trace "tree has "+troot.numNodes+" nodes.";
    HashTable ht;
    troot.writeToHashTable(ht);
    String key; foreach key in ht
{
    trace "ht[\""+key+"\"]="+ht[key];
}
}
```

testtime.tks

```
// testtime.tks
//   - by Bastian Spiegel <flink@tkscript.de>
//
//     demonstrates usage of the Time class plus some tool function
//     that you can include in your applications.
//
//
module Main;

function getCurrentDateString() {
    Time t;
    t.now();
    PointerArray mnames;
    PointerArray dnames;
    prepare {
mnames<=[
 "Jan", "Feb", "Mar", "Apr", "May", "Jun",
 "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"];
    }
    prepare {
dnames<=["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"];
    }
    return t.monthday+"/"+mnames[t.month]+"/"+t.year;
}

function getCurrentTimeString() {
    Time t;
    Integer io_h;
    Integer io_m;
    Integer io_s;
    t.now();
    io_h.value=t.hour;
    io_m.value=t.min;
    io_s.value=t.sec;
                                                          return
io_h.printf("%02i")+":"+io_m.printf("%02i")+":"+io_s.printf("%02i"
);
}

function main() {
    trace("\nit's already "+getCurrentTimeString()+" and still the
"+getCurrentDateString()+".\n\n");

    Time t;
    t.now();
    PointerArray mnames<=[
  "January", "February", "March", "April", "May", "June",
  "July", "August", "Sep", "October", "November", "December"];
     PointerArray dnames<=["Sunday", "Monday", "Tuesday", "Wednes-
day", "Thursday", "Friday", "Saturday"];
```

```
     trace("today is "+dnames[t.weekday]+", the "+t.monthday+". of
"+mnames[t.month]+" in the year "+t.year+".");
     t.monthday=25;
     t.month=11;
     t.calc();
     trace("xmas is on a "+dnames[t.weekday]+", the "+t.yearday+".
day of this year.");

}
```