ООО «ДинСофт»

Язык бортовых скриптов интеллектуальных мобильных роботов iScript

Евстигнеев Д.В.

16.09.2019 г.

Оглавление:

1. Введение в язык

Язык iScript основан на синтаксисе языка JavaScript. Однако, функции, объекты и методы, используемые в iScript, отличаются от функций объектов и методов, используемых в JavaScript.

Программа на языке iScript для робота должна быть помещена в папку «C:\DynRobot\scripts» . Обычно файл имеет расширение «.i». Например «main.i». Имя главного файла программы определяется в конфигурационном файле робота «config.txt» в параметре SCRIPT_FILE.

Выполнение файла начинается с первой строки файла. Файл должен быть в кодировке Windows-1251 или в кодировке, указанной в параметре CODE_PAGE файла config.txt.

Пример простейшей программы:

```
print("Hello, world!\n");
```

Данная программа выводит в консоль сообщение «Hello, world», после чего программа завершается.

Чтобы программа не завершалась после выполнения скрипта, необходимо организовать главный вечный цикл программы. Например:

```
while(true) sync();
```

В циклах, длительных по времени, следует вызывать функцию sync() или sleep(*time*) или GoToPlace(*place*), производящих синхронизацию состояния переменных скрипта, выдачу команд управления, а также дать возможность выполнится программным событиям.

Самой простой скрипт, управляющим роботом, приведен ниже (скрипт приведен для примера и не проверяет ошибок):

```
while(true)    // повторять бесконечно долго
{
     GoToPlace(1); // идти в точку 1 на карте
     sleep(60000); // ждать 60 секунд
     GoToPlace(2); // идти в точку 2 на карте
     sleep(60000); // ждать 60 секунд
     GoToPlace(3); // идти в точку 3 на карте
     sleep(60000); // ждать 60 секунд
}
```

1.1.Глобальные и локальные переменные. Функции

Глобальные переменные вводятся непосредственно присвоением из значения. Локальные переменные, используемые в функциях, следует объявлять с префиксом var.

```
a = 5;       // ввести глобальную переменную a
```

Пример объявления функций и локальных переменных:

```
function MyFunction(a,b)
{
   var c; // объявить локальную переменную
   var d = a - b;     // объявить лок.переменную и присвоить
   c = a + b;         // расчет
   return a * c / d;  // вернуть значение
}
```

```
// вызвать функцию, поместить в глоб. переменную res результат
res = MyFunction(10,4);
print("return value is ", res, "\n");
```

1.2.Операторы языка

В языке используются стандартные операторы C/C++, за исключением оператора switch/case:

- if (*условие*) *действие*; – оператор условия if.
- if (*условие*) *действие*$_1$; else *действие*$_2$. – оператор условия if/else.
- while(условие) *действие*; – цикл while.
- do *действие*; while(*условие*); – цикл do/while.
- for(*инициализация*; *условие*; *инкременты*) *действие*; – цикл for.
- { *действие*$_1$; *действие*$_2$; … *действие*$_n$ } – оперативные скобки.
- break; – выйти из цикла.
- continue; – пропустить текущую итерацию цикла.
- return; – выйти из функции.
- return *результат*; – выйти из функции и вернуть результат.

Пример использования операторов условий (if/else):

```
a = 10;
if (a > 5) print("a > 5\n"); else print("a < 5\n");

if (a > 6)
{
   print("a > 6\n");
   print("Next string\n");
}

if (a < 0)
{
   print("a < 0\n");
   print("Next string\n");
}
else
{
   print("a>=0\n");
}
```

Пример использования циклов for:

```
for(i=0; i < 10; i++)
{
     print("i=",I,"\n");
}
```

Пример использования циклов while:

```
PlaySpeech("Привет, мир!"); // сказать фразу
while(PlaySpeech())
{
    print("Say in progress\n"); // вывести в цикле
    sync();
}
print("Say over\n"); // вывести после цикла
```

Пример использования оператора do/while:

```
i=0;
do
{
    sync();
    print("i=",i);
    i++;
} while(i < 10);
```

2. Типы данных

2.1. Используемые типы данных

В скриптовом языке используются следующие типы данных:

Тип данных	Описание
Null	Отсутствие данных. Может принимать единственное значение null.
Bool	Логический тип данных. Может принимать значение true или false.
Binary	Блок бинарных данных.
Int	Целое 32-битное со знаком
Float	Число с плавающей запятой (double)
String	Строка
object	Указатель на объект
array	Указатель на массив
function	Указатель на функцию, объявленную в скрипте
buildinfunc	Указатель на встроенную функцию

Типы данных указываются неявно. Они формируются автоматически.

3. Стандартные функции языка

Стандартные функции языка в отличие от специальных функций платформы, поддерживаются всеми реализациями данного скриптового процессора.

3.1.Функции преобразования типов данных

Функция toBool

```
function toBool(a)
```

Преобразует входные данные *a* в тип данных bool.

Функция toInt

```
function toInt(a)
```

Преобразует входные данные *a* в тип данных int.

Функция toFloat

```
function toFloat(a)
```

Преобразует входные данные в тип данных float.

Функция toString

```
function toString(a)
```

Преобразует входные данные *a* в строку (тип данных string)

Функции typeof

```
function typeof(a)
```

Возвращает тип входных данных *a* в виде строки. Возможные значения:

Возвращаемая строка	Описание
Null	Отсутствие данных (тип данных null)
Bool	Логический тип данных
Binary	Блок бинарных данных
Int	Целое 32-битное со знаком
Float	Число с плавающей запятой (double)
String	Строка
Object	Указатель на объект
Array	Указатель на массив
function	Указатель на функцию, объявленную в скрипте
buildinfunc	Указатель на встроенную функцию.

3.2.Функции управления объектами

Функция object

Функция создает новый объект и возвращает на него указатель.

```
function object();
```

Возвращает указатель на созданный объект.

По умолчанию новый объект не имеет никаких полей. Новые поля объекта определяется простым присвоением. Методы объекта – это указатели на функции.

Объекты автоматически уничтожаются, когда на них уничтожается последняя ссылка.

Пример:

```
// объявить функцию, которая станет методом объекта
function MyMethod()
{
   return x + y;
}

//---------- создать объект -----------
myObj = object();     // создать объект
myObj.x = 1;          // создать свойство x и присвоить ему 1
myObj.y = 5;          // создать свойство y и присвоить ему 5
myObj.Method = MyMethod; // поместить в объект метод

a = myObj.Method();   // вызвать метод объекта

myObj = null;        // уничтожить объект (ссылка потеряна)
```

3.3.Функции управления массивами

Функция array

Функция создает массив и возвращает на его указатель

```
function array();
```

или

```
function array(a1,a2, a3, …, an);
```

Функция создает пустой массив (в случае отсутствия параметров) или массив, состоящий из перечисленных элементов.

Массивы автоматически уничтожаются, когда на них пропадает последняя ссылка.

Пример:

```
myArray1 = array();   // создать массив 1
myArray1[0] = 10;
myArray1[1] = 20;

myArray2 = array(10,20);   // создать массив 2

ref = myArray1;       // создать ссылку на массив 1

myArray2 = null; // уничтожить массив 2 (потерян указатель)
myArray1 = null; // массив 1 все еще остался
```

Функция count

```
function count(var a);
```

Функция возвращает количество элементов в массиве. Для бинарных данных возвращает размер блока памяти. Для строк возвращает длину.

Функция array_push

```
function array_push(a, d1, d2, … , dn);
```

Добавляет элементы *d1*, *d2*… *dn* в конец массива *a*.
Возвращает новое количество элементов в массиве.
Пример:

```
myArray = array();
array_push(myArray, 5);
array_push(myArray, 2);
array_push(myArray, 1);

// в массиве [5,2,1]
```

Функция array_push_if_not_exists

```
function array_push_if_not_exists(a, d1, d2, … , dn);
```

Добавляет элементы *d1*, *d2*… *dn* в конец массива *a*, если этих значений еще нет в массиве.
Возвращает новое количество элементов в массиве.
Пример:

```
myArray = array(1,2,3);
array_push_if_not_exists(myArray, 5, 1, 6);

// в массиве [1,2,3,5,6]
```

Функция array_pop

```
function array_pop(a);
```

Удаляет последний элемент массива и возвращает его значение. Если массив пуст, возвращает null.
Пример:

```
myArray = array(5, 10, 20);
print( array_pop(), "\n" );     // напечатать 20
print( array_pop(), "\n" );     // напечатать 10
print( array_pop(), "\n" );     // напечатать 5
```

Функция array_shift

```
function array_shift(a);
```

Удаляет первый элемент массива и возвращает его значение. Если массив пуст, возвращает null.

Пример:

```
myArray = array(5, 10, 20);

print( array_shift(), "\n" );     // напечатать 5
print( array_shift(), "\n" );     // напечатать 10
print( array_shift(), "\n" );     // напечатать 20
```

Функция array_unshift

```
function array_unshift(a, n);
```

или

```
function array_unshift(a, n, d1, d2,…, dn);
```

Функция удаляет из конца массива *n* элементов и вставляет в конец новые элементы *d1*, *d2*,…, *dn*, если они заданы.

Возвращает количество элементов в массиве после совершения операции.

Пример:

```
myArray = array(10, 20, 30, 40);
array_unshift(myArray, 2, 5);

// теперь в массиве 10, 20, 5
```

Функция array_splice

```
function array_splice(a, ofs)
```

или

```
function array_splice(a, ofs, len);
```

или

```
function array_splice(a, ofs, len, newData);
```

Функция удаляет *len* элементов массива, начиная с индекса *ofs*. Вместо них вставляет элемент *newData* (если *newData* не является массивом), или элементы массива *newData* (если *newData* является массивом).

Функция возвращает массив из удаленных элементов.

Если *len* не задан, то функция удаляет все до конца массива.

Если *len* отрицательный, то отсчет удаляемых элементов производится в отрицательную сторону от *ofs*.

Если *ofs* отрицательный, то начальный индекс рассчитывается от конца массива.

Примеры:

```
myArray = array(10, 20, 30, 40, 50);

m = array_splice(myArray, 1, 2, 60);

// myArray=[10, 60, 40, 50 ];  m = [20, 30];

array_splice(myArray, -1); // myArray = [10,20,30,40]

array_splice(myArray, 2, -3, array(70, 80, 90));

// myArray = [70, 80, 90, 40]
```

Функция array_slice

```
function array_slice(a, ofs);
```

или

```
function array_slice(a, ofs, len);
```

Возвращает новый массив, состоящий из *len* элементов исходного массива *a*, начиная с индекса *ofs*. Исходный массив при этом не изменяется.

Если *len* не задан, то функция использует все элементы до конца массива.

Если *len* отрицательный, то отсчет элементов производится в отрицательную сторону от *ofs*.

Если *ofs* отрицательный, то начальный индекс рассчитывается от конца массива.

Пример:

```
myArray = array(10, 20, 30, 40, 50);

m = array_slice(myArray, -2); // m = [40, 50]
```

Функция array_compare

```
function array_compare(a1, a2);
```

Функция возвращает true, если массивы имеют одинаковое содержимое. В противном случае возвращает false.

Функция array_remove

```
function array_remove (a, d1, d2 … dn)
```

Функция удаляет элементы из массива *a*, значение которых равно *d1*, *d2*… *dn*. Функция возвращает новый размер массива.

Примеры:

```
myArray = array(10, 20, 30, 40, 50);

array_remove(myArray, 30, 40);

// myArray = [10, 20, 50]
```

3.4.Функции управления бинарными данными

Функция binary

```
function binary();
```

или

```
function binary(size);
```

Создает блок бинарный данных и возвращает ссылку на него. Если указан параметр *size*, то блок создается из *size* байт данных.

Блок бинарных данных автоматически уничтожается, если на него уничтожается последняя ссылка.

Функция set_bin_size

```
function set_bin_size(data, size);
```

Устанавливает размер бинарных данных. Старые данные в блоке сохраняются.

Функция get_bin_size

Функция является синонимом функции count().

```
function get_bin_size(data);
```

Функция возвращает размер бинарных данных (если data является типом данных binary), а также длину массива (если data является массивом) или длину строки (если data является строкой).

Функция get_bin_byte

```
function get_bin_byte(data, ofs);
```

Возвращает значение байта из бинарного блока *data* по смещению *ofs*. Возвращаемое значение представляет собой тип данных int и может лежать в диапазоне от 0 до 255.

Если *ofs* отрицательный, то отсчет смещения ведется от конца бинарных данных.

Функция set_bin_byte

```
function get_bin_byte(data, ofs, value);
```

Устанавливает значение байта в бинарном блоке *data* по смещению *ofs* в значение *value*. Величина *value* должна быть в диапазоне от 0 до 255, в противном случае используются младшие 8 бит числа *value*, приведенного к int.

Если *ofs* отрицательный, то отсчет смещения ведется от конца бинарных данных.

Функция bin_subrange

```
function bin_subrange(data, ofs)
```

или

```
function bin_subrange(data, ofs, len)
```

Функция возвращает новый блок бинарных данных, полученный путем копирования *len* байт из исходного бинарного блока *data*, начиная со смещения *ofs*.

Если *len* не указано, то для копирования используется весь остаток исходных данных, начиная от смещения *ofs*.

Если *len* отрицательное, то отсчет ведется в обратную сторону, начиная от смещения *ofs*.

Если *ofs* отрицательное, то начальное смещение задается относительно конца блока данных.

Функция bin_to_array

```
function bin_to_array(data);
```

Функция возвращает массив, полученный путем преобразования байтов бинарного блока в элементы массива.

3.5.Функции работы со строками

Функция strlen

```
function strlen(s)
```

Возвращает длину строки *s*. Входные данные автоматически приводятся к типу данных string.

Функция strpos

```
function strpos(s, substring);
```

или

```
function strpos(s, substring, startInx)
```

Функция возвращает позицию первого вхождения подстроки *substring* в строку *s*, начиная с позиции *startInx*. Если *startInx* не указан, то поиск производится с начала строки.

Если подстрока не найдена, возвращает null.

Пример:

```
s = "Моя первая программа";

n = strpos(s, "первая");
if (n===null)
  print("Подстрока не найдена!\n");
else
  print("Смещение ", n, "\n");
```

Функция WordCase

Функция склоняет по падежам слово, в зависимости от количества *n*, переданного в качестве параметра, для фраз типа «Мы собрали 4 *гриба*».

```
function WordCase(n, one, two, five);
```

или

```
function WordCase(n, one, poly);
```

Здесь:

n – (int) количество.

one – (string) слово в единственном числе (для случая 1 гриб).

two – (string) слово в ед. числе в родительном падеже (для случая 2 гриба).

five – (string) слово во множественном числе в родительном падеже (для случая 5 грибов).

poly – (string) слово во множественном числе для фраз на английском языке.

Пример:

```
n = rand(1,1000);

PlaySpeech("Я нашла " +
             toString(n)+ " "+
                WordCase("гриб","гриба","грибов"));
while(1) sync();
```

Функция split

```
function split(splitter, s);
```

Функция разделяет строку s на подстроки, разделенные строкой *splitter*. Функция возвращает массив полученных подстрок.

Пример:

```
s = "Красный***Желтый***Зеленый";
a = split("***", s);

// a = ["Красный", "Желтый", "Зеленый"]
```

Функция join

```
function join(arr, splitter);
```

Функция объединяет элементы массива *arr* в единую строку, разделенную строкой *splitter*. Возвращает полученную строку.

Пример:

```
myArray = array("Красный", "Зеленый", "Синий");

s = join(myArray,"|"); // s = "Красный|Зеленый|Синий";
```

Функция getCharAt

```
function getCharAt(s, inx);
```

Функция возвращает один символ строки *s* с индексом *inx*. Возвращаемое значение имеет тип данных string, хотя и состоит из одного символа.

Если *inx* отрицательный, то индексация производится от конца строки.

Если *inx* указывает за пределы строки, то возвращается null.

Функция getCodeAt

```
function getCodeAt(s, inx);
```

Функция возвращает код символа строки *s* с индексом *inx*. Используется 8-битная кодировка ANSI. Возвращаемое значение имеет тип данных int.

Если *inx* отрицательный, то индексация производится от конца строки.

Если *inx* указывает за пределы строки, то возвращается null.

Функция substring

```
function substring(s, start)
```

или

```
function substring(s, start, len)
```

Возвращает подстроку из строки *s*, длиной *len* символов, начиная с символа *start*.

Если не указан параметр *len*,то копируется все, начиная с символа *start*, до конца строки.

Если *len* отрицательный, то отсчет копируемых символов ведется в противоположенную строну, начиная от смещения *start*.

Если *start* отрицательный, то начальное смещение отсчитывается от конца строки.

Функция trim

```
function trim(s)
```

Функция убирает в начале и в конце строки *s* все символы пробелов, табуляции и перевода строки и возвращает полученную строку.

Функция trimleft

```
function trimleft(s)
```

Функция убирает в начале строки *s* все символы пробелов, табуляции и перевода строки и возвращает полученную строку.

Функция trimright

```
function trimright(s)
```

Функция убирает в конце строки *s* все символы пробелов, табуляции и перевода строки и возвращает полученную строку.

Функция match

```
function match(pattern, s)
```

Функция возвращает массив строк из строки *s*, совпавших по маске *pattern*. Если совпадений не найдено, возвращает null.

Строка *pattern* представляет собой маску поиска. Все ее символы должны совпадать с искомой строкой. Кроме того строка может содержать служебные символы «{» , «}», внутри которых содержится индекс массива, в который следует поместить найденный по маске элемент.

Пример:

```
s = "go to table 5";
```

```
if (g = match("go to {1} {0}", s))  // g = ["5", "table"]
{
     placeType = g[1];           // "table"
     placeNumber = toInt(g[0]);  // 5
}
```

Функция replace

```
function replace(s, from, to);
```

или

```
function replace(s, from);
```

Функция возвращает строку, в которой произведена замена всех подстрок *from* строки *s* на строку *to*.

Если параметр *from* представляет собой массив строк, а параметр *to* представляет собой строку, то каждая подстрока из массива *from* заменяется на значение *to*.

Если параметр *from* представляет собой массив строк, и параметр *to* представляет собой массив, то каждая подстрока из массива *from* заменяется на соответствующее значение элемента массива *to*. Длины массивов должны совпадать.

Если параметр to не указан, то подстроки просто удаляются.

Функция Declination

Функция склоняет слово или словосочетание по падежам и числам

```
function Declination(s, options);
```

или

```
function Declination(s);
```

Где:

s – (string) слово или словосочетание в именительном падеже ед. числа.
options – (string) опции склонения (см. ниже)

Функция возвращает слово или словосочетание в указанном падеже и числе.

Если слову или словосочетанию соответствует несколько вариантов, то они объединяются через знак «|» (как во фреймах).

Параметр options определяет опции склонения по падежам и числам. Для разных языков (задаваемых с помощью SetDeclinatorLang) используется разная строка опции:

для русского языка:

Формат options:

<падеж>

или

<падеж><число>

Где:

<падеж> – символ:

- И – именительный;
- Р – родительный;
- Д – дательный;
- В – винительный;
- Т – творительный;
- П – предложный;
- Ж – форма притяжательного причастия (например, Мамин).
- * – любой, кроме «Ж».

<число> – символ:

- E – единственное число
- M – множественное число.
- * – любой число.

Пример:

«PE» – родительный падеж, единственное число.

«*E» – любой падеж, единственное число.

По умолчанию «**». Если указан только падеж, то число подразумевается «*».

Для английского языка:

Формат:

<число> – символ:

- S – единственное число (singular).
- P – множественное число (pluaric).
- ' – притяжательный падеж для одушевленных.
- * – любое число.

Пример:

```
s = Declination("Река", "BE"); // результат «РЕКУ»
s = Declination("Река", "B*"); // результат «РЕКУ | РЕКИ»
s = Declination("Глаз", "PM"); // результат «ГЛАЗ|ГЛАЗОВ»
s = Declination("Желтая чашка","П*");
 // результат «желтой|желтых чашке|чашках»
```

Обратите внимание, что в силу того, что система работает без словаря, то склонение существительных II рода в родительный падеж множественного числа для системы весьма проблематично. Поэтому система выдает сразу два варианта слова.

Функция SetDeclinatorLang

Функция задает язык для системы склонения слов по падежам и числам.

```
function SetDeclinatorLang(lang)
```

Где:

lang – (string) идентификатор языка:

- «RU» – русский (по умолчанию).
- «EN» – английский.

Пример:

```
SetDeclinatorLang("EN");
s = Declination("book"); // BOOKS
s = Declination("man"); // MEN
```

Функция NumberToDesc

Функция переводит число в текст для произношения функцией PlaySpeech. При необходимости производит нужное склонение числительных. Адекватно работает с дробными числами.

```
function NumberToDesc(n, case);
```

или

```
function NumberToDesc(n);
```

Где:

n – число в формате (int) или (float) (по принципу, если не float значит int).

case – (int или string) склонение, по умолчанию 0:

Если case типа int, то:

0 – числ. муж. род (например, «один»).
1 – числ. жен. род (например, «одна»).
2 – числ. ср. род (например, «одно»).

3 – прил. муж.р, им. пад. (например: «первый»).
4– прил. жен.р, им. пад. (например: «первая»).
5– прил. ср.р, им. пад. (например: «первое»).

6 – прил. муж.р, род. пад. (например: «первого»).
7– прил. жен.р, род. пад. (например: «первой»).
8– прил. ср.р, род. пад. (например: «первого»).

9 – прил. муж.р, дат. пад. (например: «первому»).
10– прил. жен.р, дат. пад. (например: «первой»).
11– прил. ср.р, дат. пад. (например: «первому»).

12 – прил. муж.р, вин. пад. (например: «первый»).
13– прил. жен.р, вин. пад. (например: «первую»).
14– прил. ср.р, вин. пад. (например: **«первое»)**.

15 – прил. муж.р, твор. пад. (например: «первым»).
16– прил. жен.р, твов. пад. (например: «первой»).
17– прил. ср.р, твор. пад. (например: «первой»).

18 – прил. муж.р, твор. пад. (например: «первом»).
19– прил. жен.р, твов. пад. (например: «первой»).
20– прил. ср.р, твор. пад. (например: «первом»).

Если case типа string, то, строка из следующих комбинаций в любой последовательности, с соблюдением регистра:

Числ – числительное
Прил – прилагательное, образованное от числительного.

М – мужского рода.
Ж – женского рода
С – среднего рода.

Им – именительный падеж.
Род – родительный падеж.
Дат – дательный падеж
Вин – винительный падеж
Тв – творительный падеж
Пред – предложный падеж.

Следует отметить, что все перечисленные части строки может быть дополнены до полного слова, например «Тв» может быть дополнен до «Творительный».

Функция возвращает строку (string) с текстом.

Используется для русского и английского языка в зависимости от SetDeclinatorLang.

Для английского языка функция (где нет склонения числительных по родам), числа остаются числами, только разделяются словами типа «thousand», «million» и т.д.

Следует отметить, что для русского языка некоторые прилагательные, образованные от числительных, формируются не согласно правилам русского языка, а так, чтобы синтезатор речи их мог правильно произносить.

Пример:

```
s = NumberToDesc(3003.14);
print(s,"\n");  // три тысячи три целых, четырнадцать сотых

s = NumberToDesc(2, 1); // в женском роде
print(s,"\n");  // две

s = NumberToDesc(21, "Прил.Ж.Род");
print(s,"\n");  // двадцать первой
```

Функция DateToDesc

Функция переводит дату в текст для произношения функцией PlaySpeech. При необходимости производит нужное склонение числительных.

```
function DateToDesc(t, case, isYear);
```

или

```
function DateToDesc(t, case);
```

или

```
function DateToDesc(t);
```

Где:

t – (int) дата в формате TIMESTAMP (совместимо с функцией time()).

case – (int или string) падеж, по умолчанию 0:

Если case типа int, то:

0 –именительный.
1 –родительный.
2 – дательный.
3 – винительный.
4 – творительный.
5 – предложенный.

Если case типа string, то, строка с одним из следующих слов, с соблюдением регистра:

Им – именительный падеж.
Род – родительный падеж.
Дат – дательный падеж
Вин – винительный падеж
Тв – творительный падеж
Пред – предложный падеж.

Следует отметить, что все перечисленные части строки может быть дополнены до полного слова, например «Тв» может быть дополнен до «Творительный».

isYear – (bool) добавлять ли к дате год (по умолчанию true).

Функция возвращает строку (string) с текстом.

Используется для русского и английского языка в зависимости от SetDeclinatorLang.

Для русского языка даты строятся по принципу (при isYear=true) «Тринадцатое мая две тысячи девятнадцатого года».

Для английского языка функция возвращает строку, типа «May 13, 20 19» (при isYear=true);

Пример:

```
s = DateToDesc(time(), "Дательный", false);
print(s,"\n");  // тринадцатому мая

s = DateToDesc(time());
print(s,"\n"); // тринадцатое мая две тысячи девятнадцатого года
```

Функция GetWeekDay

Функция возвращает название дня недели в требуемом падеже.

```
function GetWeekDay(weekDay, case);
```

или

```
function GetWeekDay(weekDay);
```

Где:

weekDay – (int) день недели от 0 до 6 (0 – воскресенье).

case – (int или string) падеж, по умолчанию 0:

Если case типа int, то:

0 –именительный.

1 –родительный.

2 – дательный.

3 – винительный.

4 – творительный.

5 – предложенный.

Если case типа string, то, строка с одним из следующих слов, с соблюдением регистра:

Им – именительный падеж.

Род – родительный падеж.

Дат – дательный падеж

Вин – винительный падеж

Тв – творительный падеж

Пред – предложный падеж.

Следует отметить, что все перечисленные части строки может быть дополнены до полного слова, например «Тв» может быть дополнен до «Творительный».

Функция возвращает строку (string) с текстом.

Используется для русского и английского языка в зависимости от SetDeclinatorLang.

Для английского языка параметр case игнорируется

Пример:

```
s = GetWeekDay(1, "Пред");
print(s,"\n");  // Понедельнике
```

Функция GetMonthName

Функция возвращает название месяца в требуемом падеже.

```
function GetMonthName(month, case);
```

или

```
function GetMonthName(month);
```

Где:

month – (int) номер месяца от 1 до 12.

case – (int или string) падеж, по умолчанию 0:

Если case типа int, то:

0 –именительный.

1 –родительный.
2 – дательный.
3 – винительный.
4 – творительный.
5 – предложенный.

Если case типа string, то, строка с одним из следующих слов, с соблюдением регистра:

Им – именительный падеж.
Род – родительный падеж.
Дат – дательный падеж
Вин – винительный падеж
Тв – творительный падеж
Пред – предложный падеж.

Следует отметить, что все перечисленные части строки может быть дополнены до полного слова, например «Тв» может быть дополнен до «Творительный».

Функция возвращает строку (string) с текстом.

Используется для русского и английского языка в зависимости от SetDeclinatorLang.

Для английского языка параметр case игнорируется

Пример:

```
s = GetMonthName(1, "Род.");
print(s,"\n");  // Февраля
```

Функция ValuteToDesc

Функция переводит валюту в строку для правильного произношения функцией PlaySpeech.

```
function ValuteToDesc(n, volute, say00);
```

или

```
function ValuteToDesc(n, volute);
```

Где:

n – сумма в формате (float) (по принципу, если не float значит int).

volute – (string) валюта (определяется автоматически, как по полному названию, по трехбуквенному сокращению, по числовому коду валюты).

say00 – (bool) при отсутствии дробной части добавлять слово «ровно» («exactly»). По умолчанию false.

Функция возвращает строку (string) с текстом.

Используется для русского и английского языка в зависимости от SetDeclinatorLang.

Адекватно работает только с валютами: рубль, доллар, евро, гривна и юань. Для остальных валют просто формирует дробное число и добавляет название валюты.

Для английского языка функция (где нет склонения числительных по родам), числа остаются числами, только разделяются словами типа «thousand», «million» и т.д.

Пример:

```
s = ValuteToDesc(24.05,"RUB", true);
print(s,"\n");  // двадцать четыре рубля пять копеек

s = ValuteToDesc(2, "Доллар", true);
print(s,"\n");  // два доллара ровно
```

3.6.Математические функции

Основные математические функции

```
function sin(a);       // синус
function cos(a);       // косинус
function tan(a);       // тангенс
function asin(a);      // арксинус
function acos(a);      // арккосинус
function atan(a);      // арктангенс
function atan2(y,x);   // частичный арктангенс
function log(a);       // натураный логорифм
function exp(a);       // экспонента
function sqrt(a);      // квадратный кронь
function sqr(a);       // возвещение в квадрат
function pow(a,b);     // возведение a в степень b
function lead(a);      // приводит угол a в диапазон от -π до +π
```

Функции аналогичны в использовании в других языках программирования.

Функция rand

```
function rand();
```

или

```
function rand(n);
```

или

```
function rand(a,b);
```

Функция возвращает псевдослучайное число.

Если функция вызвана без аргументов, то возвращает псевдослучайное число типа float от 0 до 1.

Если функция вызвана с одним аргументом *n*, то возвращает псевдослучайное число типа int 0 до *n* (*n* не включительно).

Если функция вызвана с двумя аргументами *a* и *b*, то возвращает псевдослучайное число типа int в диапазоне от *a* до *b* (обе границы включительно).

3.7.Функции работы с датой и временем

Функция time

```
function time();
```

или

```
function time(timeOfDay);
```

Возвращает текущую дату и время (timestamp). Если timeOfDay = true, то возвращает время текущих суток. Время возвращается в виде целого числа секунд от 0:00:00 1 января 1970 года по Гринвичу. Функция эквивалентна функции time в языке C.

Пример:

```
var tm = localtime(time());
```

```
print("Today is ", tm.day, ".", tm.mon, ".", tm.year, "\n");
```

Функция localtime

```
function localtime(timestamp);
```

Функция преобразует время в формате timestamp в структуру, содержащую дату и время в расшифрованном формате.

На входе:

timestamp – число секунд с 0:00:00 1 января 1970 года по Гринвичу (результат, возвращаемый функцией time).

На выходе формируется объект со следующими свойствами:

- year – год в 4-значном формате (например: 2016).
- month – месяц от 1 до 12.
- day – день месяца от 1 до 31.
- hour – час от 0 до 23.
- min – минута от 0 до 59.
- sec – секунда от 0 до 59.
- weekDay – день недели от 0 до 6. Воскресенье – 0, понедельник – 1 и т.д.

Пример:

```
var tm = localtime(time());
print("Today is ", tm.day, ".", tm.mon, ".", tm.year, "\n");
```

Функция mktime

```
function mktime(tm);
```

или

```
function mktime(year, mon, day, hour, min, sec);
```

или

```
function mktime(year, mon, day, hour, min);
```

или

```
function mktime(year, mon, day, hour);
```

или

```
function mktime(year, mon, day);
```

или

```
function mktime(year, mon);
```

или

```
function mktime(year);
```

Функция формирует время в формате timestamp (число секунд от 0:00:00 1 января 1970 года по Гринвичу). Функция обратная для функции localtime.

На входе:

Либо объект tm, содержащий следующие свойства:

- year – год в 4-значном формате (например: 2016).
- month – месяц от 1 до 12.
- day – день месяца от 1 до 31.
- hour – час от 0 до 23.
- min – минута от 0 до 59.
- sec – секунда от 0 до 59.

Либо:

year, mon, day, hour, min, sec – соответственно, год, месяц, день, час, минута, секунда.

Пример:

```
var t = time(); // текущее время

// разложить дату и время
var tm = localtime(t);

// установить 12 часов текущего дня
tm.hour = 12;
tm.min = 0;
tm.sec = 0;

// сформировать время, соответствующее 12 часам тек.дня
var t12 = mktime(tm);

// проверить, сколько времени прошло с 12 часов тек.дня
var deltaTime = t - t12;
```

Функция GetTickCount

```
function GetTickCount()
```

Функция возвращает время в миллисекундах от старта Windows (для Windows) или соответствующие текущим суткам (для Linux).

Функция sleep

```
function sleep(ms);
```

или

```
function Sleep(ms);
```

или

```
function Delay(ms);
```

Функция создает паузу выполнения скрипта на <*ms*> миллисекунд. В момент ожидания производится синхронизация работы системы скриптов, позволяя обработать поступившие входные сообщения и обслужить таймеры.

Пример:

```
PlaySpeech("Привет мир!");
Sleep(2000);
PlaySpeech("Пока мир!");
```

У функции есть алиасы Sleep (с заглавной буквы) и Delay.

3.8.Функции ввода/вывода

Функция include

```
function include(filename);
```

Подключает скрипт с именем файла filename.

Пример:

```
include("scripts/common.i");  // подключить файл скрипта
```

Функция print

```
function print(a1,a2,…,an);
```

Выводит в консоль строки *a1, a2, … an*, перечисленные в параметрах.

Пример:

```
print("Hello, world!\n");

a = 5;
print("a = ", a , "\n"); // несколько параметров
```

Функция show

```
function show(var);
```

или

```
function show(var, fileName);
```

Выводит либо в консоль, либо в указанный файл *fileName* значение переменной *var* в формате JSON. Это позволяет отобразить все поля объектов и массивов, включая все дочерние объекты и массивы, а также их значения.

Если указывается имя файла fileName, то вывод осуществляется исключительно в указанный файл. Путь к файлу задается либо абсолютный, либо относительно директории программы «ДинРобот».

Пример:

```
// создать объект
a = object();
a.x = 100;
a.y = 200;
a.lines = array();
a.lines[0] = 100;
a.lines[1] = 200;
a.lines[2] = 300;

// вывести в файл
show(a, "debug.txt");
```

В результате работы примера в файле debug.txt будет записано следующее:

```
{
  x : 100,
  y : 100,
  lines : [
    100,
    200,
    300
  ]
}
```

3.9.Объявленные константы

В языке заранее объявлены следующие константы:

null – число null;
true – правда (для типа данных bool);
false – ложь (для типа данных bool);
PI – значение числа π (3.14159265358979323846264338327 95)

4. Фреймы. Разбор запросов на естественном языке

4.1. Введение

В скриптовый язык iScript встроены операторы разбора текстовых запросов на естественном языке. С их помощью можно реализовать сложную систему фреймов (фреймообразную структуру) базы знаний робота[1]. Наличие знаний, а также механизмов логического вывода на их основе, позволяют создать интеллектуальную систему управления поведением робота (стратегический уровень системы управления).

На вход системы фреймов поступают текстовые запросы с системы распознавания речи, различные события, а также команды с центрального сервера управления. Текстовый запрос может быть сгенерирован и самой системой скриптов, что позволяет ей создавать сложные действия из набора простых.

Кроме того, встроенные функции языка позволяют добавлять новые знания в систему, что дает возможность в случае необходимости реализовать функционал самообучения.

4.2. Фрейм

Основным элементом языка запросов является *фрейм*.

Каждый фрейм состоит из *словесной предпосылки* и *программного следствия*.

Словесная предпосылка фрейма определяет ключевые слова, по которым он активизируется. Программное следствие является скриптом на языке iScript.

Пример объявления фрейма:

```
...
   frame("Как живешь")
   {
       PlaySpeech("Хорошо");
   }
...
```

В данном случае «Как живешь» является словесной предпосылкой, а скриптовый код «{ PlaySpeech(…); }» является программным следствием.

Фрейм «Как живешь» может быть активизирован текстовым запросом с фразой: «**Как** ты **живешь**?». Причем при запросе активизируется фрейм, в котором больше число активизированных слов.

4.3. Синонимы в словесной предпосылке

Словесная предпосылка может содержать слова-синонимы, перечисляемые через знак «|». В этом случае в указанном месте запроса может находиться любое из перечисленных слов. Например:

```
...
   frame("Дай шоколад|шоколадку|батончик")
   {
      ...
   }
...
```

[1] База знаний (согласно теории интеллектуальных систем) – это не база данных. База данных содержит данные (информацию о предметах, их количестве, описание и пр.) База знаний же содержит набор правил поведения робота, а также механизмы формирования новых знаний.

В данном случае фраза «**Дай** мне этот **батончик**» активизирует данный фрейм.

Если в качестве синонима используется словосочетание, то слова этого словосочетания следует объединить знаком «&». Например:

```
...
   frame("Принеси нож | ножик | чем & порезать")
   {
      ...
   }
...
```

Таким образом, данный фрейм может быть активизирован как фразой «**Принеси нож**», так и фразой «**Принеси чем порезать** мясо».

Синонимом может быть являться весь фрейм целиком. Для создания фреймов-синонимов нужно перечислить фреймы друг за другом, а затем указать их общее программное следствие:

```
...
   frame("Как тебя зовут")
   frame("Как твое имя")
   frame("Зовут тебя как")
   {
      PlaySpeech("Меня зовут Си-Си", "Мое имя Си-Си");
   }
...
```

В данном случае при активизации любого из перечисленных фреймов будет выполнено указанное программное следствие.

4.4. Необязательные слова

Во фрейме могут присутствовать необязательные слова. Такие слова заключают в квадратные скобки. Необязательные слова могут опционально присутствовать во фразе.

Например:

```
...
   frame("Расскажи [что-нибудь] про [ваше] мероприятие")
   {
      ...
   }
...
```

В данном случае фрейм будет активизироваться как фразой «Расскажи про мероприятие», так и фразами «Расскажи что-нибудь про мероприятие», «Расскажи про ваше мероприятие» или «Расскажи что-нибудь про ваше мероприятие».

Использование необязательных слов особенно актуально при использовании Microsoft Speech Recognizer, т.к. данный движок распознавания речи не приемлет присутствие слов, которых нет в заранее заложенной фразе.

Но даже, если используется движок распознавания речи с открытой лексикой, то приоритет активизации фрейма с наличием необязательных слов будет выше, чем аналогичного без обязательных слов.

4.5.Синонимы с необязательным словом

Во фрейме могут присутствовать синонимы (альтернативы) с необязательным словом. Для этого достаточно в середине слов-синонимов указать одну пустую альтернативу.

Например:

```
...
   frame("Расскажи про это||ваше мероприятие")
   {
      ...
   }
...
```

В данном случае фрейм будет активизироваться как фразой «Расскажи про мероприятие», так и фразами «Расскажи про это мероприятие», а также, «Расскажи про ваше мероприятие».

Использование необязательных слов особенно актуально при использовании Microsoft Speech Recognizer, т.к. данный движок распознавания речи не приемлет присутствие слов, которых нет в заранее заложенной фразе.

Но даже, если используется движок распознавания речи с открытой лексикой, то приоритет активизации фрейма с наличием необязательных слов будет выше, чем аналогичного без обязательных слов.

4.6.Параметры в словесной предпосылке

В словесной предпосылке фрейма могут содержаться параметры, являющимися ссылками на другие фреймы.

Например:

```
...
   frame("Принеси <что:Объекты>")
   {
       print(TextOf("что"), "\n");  // вывести текст параметра
   }
...
```

В данном случае фрейм «Принеси…» имеет параметр с названием «что», который является ссылкой на фреймсет «Объекты». Понятие «фреймсет» будет рассмотрено далее.

Если фреймсет «Объекты» содержит, например, объект «Кубик», то фрейм «Принеси…» может быть активизирован, например, фразой: «А **принеси**-ка мне тот **кубик**».

Допускается создание нетиповизированных параметров фрейма. В этом случае параметр вбирает в себя все слова запроса, стоящие от предыдущего до следующего слова словесной предпосылки. Если следующего слова словесной предпосылки нет, то параметр берет на себя все слова до конца фразы.

Например:

```
...
   frame("Принеси <n> коробок|коробку|коробки")
   {
       var n = TextOf("n");
```

```
        ...
    }
...
```

Таким образом, фрейм может быть активизирован фразой: «**Принеси 5 коробок**». Причем параметр «n» вбирает все слова между «Принеси» и «коробок». В данном случае параметр будет равен 5.

4.7. Числовые параметры в словесной предпосылке

В словесной предпосылке фрейма могут содержаться параметры, являющимися числами и набором цифр. Для этого эти параметры должны иметь тип «Number» и «Digits» соответственно (большие или малые буквы не важны).

Под числами (Number) подразумевается слово, состоящее из цифр, например «567».

Под набором цифр (Digits) подразумевается набор последовательных слов, состоящих из цифр. Однако текстовое значение данного параметра представляется путем соединения всех разрозненных цифр в одно число без пробелов. Например «5 6 44 2», в результате текстовое значение параметра будет «56442».

Набор цифр удобно применять при диктовке: люди произносят длинный номер (например, номер расчетного счета) по частям (в виде отдельных цифр), однако все эти цифры являются одним длинным номером.

Например:

```
...
   frame("<n:Digits>")
   {
     a = TextOf(n);
     print(a, "\n");  // вывести текст параметра
   }
...
```

или

```
...
   frame("Мы собрали <m:Number> гриб|гриба|грибов")
   {
     m = TextOf(m);
     print(m, "\n");  // вывести текст параметра
   }
...
```

Таким образом, в первом примере фрейм может быть активизирован текстом: «4 5 8 3 5», в то время как параметр «n» фрейма будет иметь значение «45835».

Во втором примере вместо параметра «m» может быть только одно число, например: «Мы собрали 43 гриба». Параметр «m» будет иметь значение 43. Однако фраза «Мы собрали 4 3 гриба» данный фрейм активирует только с параметром «m» равным 4.

4.8. Фреймсет

Каждый фрейм должен быть вложен во *фреймсет*. Каждый такой фреймсет определяет группу фреймов одного типа. Каждый фреймсет имеет название и приоритет выполнения.

Фреймсеты подразделяются на *командные* и *объектные*.

Командные фреймсеты содержат фреймы, определяющие действия робота. Например «Скажи …», «Принеси …», «Иди…». С них начинается обработка текстового запроса. Командные фреймсеты могут иметь различный приоритет, определяющий порядок выполнения действий, вложенных в него фреймов (выполнить немедленно или поставить в очередь).

Так, например, команды «Иди…», «Принеси…», следует ставить в очередь. А команда «Стой!», очевидно, должна быть выполнена немедленно.

Приоритет фреймсета может быть только положительным. Согласно внутреннему представлению данных, отрицательный приоритет определяет объектные фреймсеты.

Объектные фреймсеты определяют объекты предметной области, описывают их свойства, место, время и т.п. Например, «Стол», «Стул», «Конфета», «Здесь», «На столе», «сейчас», «через 5 минут» и т.п. Фреймы объектных фреймсетов могут быть только параметрами командных фреймов.

Объектные фрейметы не содержат приоритета (точнее их приоритет равен (-1)).

Пример командного фреймсета с приоритетом 2:

```
frameset("Команды", 2)
{
   frame("Иди") { ... }
   frame("стой") { ... }
   ...
}
```

Пример объектного фреймсета:

```
frameset("Объекты")
{
   frame("кубик|кубика|кубику") { return 1; }
   frame("коробка|коробки|коробке") { return 2; }
}
```

Далее рассматривается пример структуры фреймов, показывающий их взаимодействие:

```
// Командный фреймсет с приоритетом 1
frameset("Команды", 1)
{
    frame("Где <что:Объекты>")
    {
       ...
    }
    frame("Как тебя зовут")
    frame("как твое имя")
    {
       ...
    }
    frame("Иди|Подойди <куда:Места>")
    {
       ...
    }
}
```

```
// Объектный фреймсет, перечисляет известные роботу объекты.
// Все фреймы данного фреймсета возвращают идентификатор объекта
frameset("Объекты")
{
    frame("кубик|кубику|кубика") { return 1; }
    frame("шар|шару|шарика") { return 2; }
}


// Объектный фреймсет, перечисляющий понятные роботу места.
// Все фреймы данного фреймсета возвращают координаты места
frameset("Места")
{
    frame("на & столик | к & столику <n>") { ... }
    frame("на кухню") { ... }
    frame("к <obj:Объекты>") {...}
}

while(1) sync();   // главный бесконечный цикл программы
```

Как показано в примере, имеются три фреймсета. Только фреймсет «Команды» является командным. Его приоритет 1. В данный фреймсет вложено несколько фреймов.

Фрейм «Где <что:Объекты>» в словесной предпосылке имеет слот с параметром. Параметры обозначаются в треугольных скобках и имеют название параметра (в данном случае «что»), и, опционально, фреймсет, в данном случае «Объекты».

В данном случае активизация фрейма «Где…» произойдет только в том случае, если запрос пользователя на естественном языке будет включать слова из фрейма «Где…», а также из одного из фреймов фреймсета «Объекты».

Например, запрос «А **где** этот **кубик**?» активизирует фрейм «Где…» и «Кубик…». Причем параметром «что» фрейма «Где…» будет являться фрейм «Кубик…».

Фреймы «Как тебя зовут» и «Как твое имя» являются синонимами, т.к. выполняют одно и то же программное следствие.

Фрейм «Кубик…» имеет слово с синонимами, перечисленными через знак «|». Любое из слов-синонимов может стоять на данном месте фразы.

Следует отметить, что знак «|» распространяется только на одно слово.

Если синонимом является последовательность слов, как например, у фрейма «на столик…», то все слова данной последовательности должны быть объединены через знак «&».

У фрейма «на столик…» имеется нетиповизированный параметр «n». Если тип параметра не указан, то он вбирает в себя все слова запроса, стоящие от предыдущего до следующего слова словесной предпосылки. Если следующего слова словесной предпосылки нет, то параметр берет на себя все слова до конца фразы.

В данном примере запрос «Иди к столику 5» активизирует фреймы «Иди к…» и «Столик…». Причем, параметром «куда» фрейма «Иди…» будет являться фрейм «Столик…», а параметром n фрейма «Столик…» будет являться текст «5».

Всю мощь фреймовой структуры показывает фрейм «к <obj:Объекты>». Его может активизировать, например, текстовый запрос: «Подойди к кубику». Здесь активизируется фрейм «Иди|Подойди…», параметром «куда» которого будет являться фрейм «к <obj:Объекты>». Причем параметр «obj» фрейма «к <obj:Объекты>» будет являться фрейм «Кубик».

Несложно посчитать, какое количество комбинаторных перечислений было описано достаточно простым способом. Несложно также представить, насколько еще будет

упрощено написание модели диалогов, если потребуется увеличить число объектов, число мест, а также число команд, оперирующих с ними, например, «Возьми <что>», «Принеси <что> <откуда>» и т.д.

4.9.Текст параметра. Функция TextOf

Программное следствие фрейма может раскрывать текст параметров, указанных в словесной предпосылке фрейма.

Функция **TextOf(***paramName***)** возвращает текст параметра, т.е. ту часть запроса, которая активизировала указанный параметр.

В данном случае «*paramName*» представляет собой строку и названием параметра. Обычно название параметра указываться в виде строковой константы, т.е. записывается в кавычках.

Например:

```
...
frame("Покажи <что:Объекты>")
{
    var objName = TextOf("что");
    print(objName);
}
...
```

Если данный фрейм был активизирован фразой «Покажи кубик», то переменная *objName* примет значение «кубик».

Текст параметра удобно вставлять в часть запроса в функцию LangQuery(…).

4.10. Значения параметра. Функция ValueOf

Программное следствие фрейма может раскрывать значение параметров, указанных в словесной предпосылке фрейма. Функция **ValueOf(***paramName***)** возвращает значение указанного параметра.

При этом производится выполнение программного следствия фрейма, указанного в качестве параметра. Результат, который возвращает фрейм оператором return, является значением, которое возвращает функция ValueOf.

В данном случае «*paramName*» представляет собой строку и названием параметра. Обычно название параметра указываться в виде строковой константы, т.е. записывается в кавычках.

Раскрывать значение нетиповизированного параметра фрейма запрещено.

Например:

```
frameset("Команды", 1)
{
    frame("Покажи <что:Объекты>")
    {
        var objID = ValueOf("что");
        SetBrowserURL("http://content/show.php?id=" + objID);
    }
    ...
}
frameset("Объекты")
{
    frame("Торт") { return 1; }
    frame("Пирожное") { return 2; }
```

```
    frame("Шоколад") { return 3; }
}

while(1) sync();   // главный бесконечный цикл программы
```

В данном примере запрос «**Покажи пирожное**» активизирует фрейм «Покажи…» и фрейм «Пирожное». Причем значением параметра «что» фрейма «Покажи…» будет фрейм «Пирожное».

При вызове функции ValueOf("что") вызывается программное следствие фрейма из параметра «что», т.е. фрейма «Пирожное». При этом программное следствие этого фрейма вернет значение 2.

Таким образом, локальная переменная objID в программном следствии фрейма «Покажи…» будет иметь значение 2.

4.11. Запрос из программного кода

Запрос из программного кода обычно применяется для реализации сложных действий из набора простых.

Для создания запроса служит функция LangQuery(…). В качестве параметров этой функции передается текст запроса на естественном языке. Опционально ей можно передать название фреймсета (не обязательно командного), в котором следует искать ответ.

Функция возвращает значение, возвращаемое оператором return в активизированном фрейме.

Например:

```
frameset("Действия", 1)
{
    frame("Найди <что:Объекты>")
    {
       ...
       return true;
    }

    frame("Подойди к <чему:Объекты>")
    {
       if (!LangQuery("Найди "+TextOf("чему"))) return false;
       ...
       return true;
    }

    frame("Возьми <что:Объекты>")
    {
       if (!LangQuery("Подойди к "+TextOf("что"))) return false;
       ...
       return true;
    }
}

frameset("Объекты")
{
    ...
}
```

```
while(1) sync();   // главный бесконечный цикл программы
```

4.12. Последнее значение фреймсета. Местоимения

У каждого фреймсета сохраняется последний активированный фрейм. В каждом фреймсете могут быть расположены специальные фреймы-местоимения, активация которых приводит к активации последнего фрейма из данного фреймсета.

Фрейм-местоимение имеет вместо программного следствия ключевое слово «pronoum». Например:

```
frameset("Объекты")
{  ...
   frame("она|его|им") pronoum;
   ...
}
```

Пример использования фрейма-местоимения приведен ниже:

```
frameset("действия", 1)
{
   frame("Найди <что:Объекты>")
   {
      print("Ищу объект id:", ValueOf("что"), "\n");
   }
   frame("Возьми <что:Объекты>")
   {
      print("Кладу объект id:", ValueOf("что"), "\n");
   }
}



frameset("Объекты")
{
    frame("Кубик") { return 1; }
    frame("Шарик") { return 2; }
    frame("он|его") pronoun;     // местоимение
}
while(1) sync();   // главный бесконечный цикл программы
```

В приведенном примере появляется возможность формировать следующую последовательность фраз:

«**Найди кубик**»;

«**Возьми его**».

При этом фраза «**Возьми его**» использует местоимение «его». Важно, что система при этом понимает, о чем идет речь.

В тот момент, когда была сформирована фраза «Найди кубик», был активирован фрейм «Найди…», при этом параметром «что» данного фрейма был фрейм «Кубик».

Функцией ValueOf фрейм «Кубик» активизировался. Таким образом, фрейм «кубик» стал последним активизированным фреймом фреймсета «Объекты».

Идущая следом фраза «Возьми его» активирует фрейм «Возьми...», а параметром данного фрейма является фрейм «Он|Его». Т.к. данный фрейм является фреймом-местоимением, то при его активации выполняется программное следствие последнего активированного фрейма фреймсета «Объекты», т.е. фрейма «Кубик».

4.13. Значения последнего текста фреймсета из программного кода

Определить последнее значение активизированного фрейма указанного фреймсета можно с помощью функции LastTextOf:

```
frameset("Действия",1)
{    ...
     frame("отвези <что:Объекты> в мойку")
     {
         ...
     }
     frame("отвези в мойку")
     {
        var txt = LastTextOf("Объекты");
        LangQuery("отвези "+txt+" в мойку");
     }
}
```

4.14. Сложносоставные предложения

Фреймообразные структуры позволяют создавать сложносоставные предложения путем создания отдельного командного фреймсета:

```
frameset("Составные",1)
{
     frame("<действие1:Действия> и <действие2:Действия>")
     {
          if (!ValueOf("действие1")) return false;
          if (!ValueOf("действие2")) return false;
          return true;
     }

     frame("<a1:Действия> <a2:Действия> и <a3:Действия>")
     {
          if (!ValueOf("a1")) return false;
          if (!ValueOf("a2")) return false;
          if (!ValueOf("a3")) return false;
          return true;
     }
}


frameset("Действия",1)
{
     frame("Возьми <что:Объекты>")
     {
          print("Беру ", ValueOf("что"), "\n");
          return true;
```

```
     }

     frame("Раскрась <что:Объекты>")
     {
          print("Крашу ", ValueOf("что"), "\n");
          return true;
     }
}

frameset("Объекты")
{
     frame("кубик") { return 1; }
     frame("шарик") { return 2; }
     frame("он|его") pronoun;
}

while(1) sync();
```

В примере фреймсет «Составные» содержит два типа предложений, состоящих из двух и из трех составных частей.

Таким образом, фраза «**Возьми кубик и раскрась его**» активирует фрейм «действие1 и действие2». Следует отметить, что данная фраза может по отдельности активизировать фреймы «Возьми…» и «Раскрась…», но число слов в предложении, активизированное фреймом «действие1 и действие2» значительно больше.

Программное следствие фрейма «действие1 и действие2» приводит к выполнению сначала действия 1 и, если оно выполнилось удачно, переходит к выполнению действия 2.

4.15. Создание фреймовой структуры в коде программы

Фреймсеты и фреймы могут создаваться из любого места скриптовой программы, даже внутри функций, циклов и условий:

```
// фунция создания фреймсета имен
function GenNameFrames( english )
{
   frameset("Объекты")
   {
      // условие создание фрейма
      if (english)
      {
           frame("Jonh") { return 1; }
           frame("Michael") { return 2; }
      }
      else
      {
           frame("Иван") { return 1; }
           frame("Михаил") { return 2; }
      }
   }
}

// вызов функции
```

```
GenNameFrames(false);
...
```

Следует обратить внимание, что программному следствию фрейма недоступно значение локальных переменных функций.

Если фреймсет с таким именем уже существует, то прошлое реализация фреймсета удаляется.

4.16. Временный фреймсет

В языке iScript существует понятие временный фреймсет, который существует в течение 20 секунд после своего создания. Далее фреймсет удаляется.

Такой фреймсет удобно использовать при создании диалогов. Например, если робот задал пользователю какой-либо вопрос и ждет от него ответа. У пользователя есть 20 секунд, чтобы ответить, в противном случае его ответ будет уже не актуален.

Для создания временного фреймсета достаточно создать фреймсет с названием «temp».

Пример:

```
PlaySpeech("Хотите я расскажу о нашей продукции?");
frameset("temp", 1)
{
    frame("да")
    {
        PlaySpeech("Наша продукция самая...");
    }
    frame("нет")
    {
        PlaySpeech("Зря");
    }
}
```

4.17. Программное создание и удаление фреймсетов и фреймов

Создание и удаление фреймсетов и фреймов доступно из кода скриптовой программы с помощью специальных функций.

Функция AddFrame

Функция **AddFrame**(<*frameset*>, <*словесная предпосылка*>,<*код*>) создает фрейм со словесной предпосылкой <*словесная предпосылка*> во фреймсете <*frameset*> с программным следствием, заданного строкой <*код*>.

Пример:

```
AddFrame("Объекты", "Яблоко",  "return 1;" );
```

Если в качестве кода используется строка с ключевым словом «pronoun», то создается фрейм-местоимение. Пример программного создания фрейма-местоимения:

```
AddFrame("Объекты", "он|его", "pronoun");
```

Данным способом удобно создавать фреймы по элементам базы данных:

```
// подключение к базе данных MySQL
if (mysql_connect())
{  // SQL-запрос
   if (mysql_query("SELECT `id`, `name` FROM objects"))
   {
      // разбор результата в виде ассоциированного объекта
      var row;
      while(row = mysql_fetch_assoc())
      {
         // создание фрейма во фреймсете «Объекты»
         // с названием из базы данных, и возвращающего
         // идентификатор объекта
         AddFrame("Объекты", row.name, "return "+row.id+";");
      }
      // освободить память от результатов запроса
      mysql_free_result();
   }
   // закрыть MySQL
   mysql_close();
}
...
```

Функция AddFrameset

Функция **AddFrameset**(<*название*>, [<*приоритет=-1*>, [<*temp=false*>]]) создает фреймсет с названием <*название*> и приоритетом <*приоритет*>. Признак <*temp*> формирует временный фреймсет.

Также временный фреймсет можно создать, придав ему название «temp».

Параметры <*приоритет*> и <*temp*> являются необязательными. Приоритет по умолчанию равен (-1), что соответствует объектному фреймсету. Признак временного фреймсета по умолчанию не задан.

Пример создания объектного фреймсета из программного кода:

```
AddFrameset("Объекты");
```

Пример создания командного фреймсета с приоритетом 2 из программного кода:

```
AddFrameset("Действия", 2);
```

Пример создания временного фреймсета из программного кода:

```
AddFrameset("Ответы", 1, true);
```

Следует обратить внимание, что если одноименный фреймсет уже существует, то старое его содержимое сохраняется, но изменяется приоритет и признак временного фреймсета. Причем отчет времени существования временного фреймсета отсчитывается от последнего вызова данной функции.

Функция DeleteFrameset

Функция DeleteFrameset(<*название*>) удаляет фреймсет с названием <*название*>, а также все его фреймы.

Например:

```
DeleteFrameset("Ответы");
```

Функция EnableFrameset

Функция **EnableFrameset**(<*название*>, [<*доступность=true*>]) управляет доступностью фреймсета с названием <*название*>. Признак <*доступность*> управляет доступностью указанного фреймсета. Если <*доступность*> равна true, то фреймсет доступен, если <*доступность*> равна false, то фреймсет недоступен.

Если фреймсет не доступен, то активизация его фреймов производиться не будет.

Пример:

```
EnableFrameset("Объекты", false);
```

Функция ClearTasks

Функция **ClearTasks**() удаляет все задачи, стоящие в очереди на исполнение

Функция SetTaskPrior

Функция **SetTaskPrior**(prior) устанавливает приоритет выполнения текущей задачи. Таким образом, появляется возможность управлять последовательностью выполнения фреймов. Например, программный код одного из фреймов выполняет длительные действия, во время которых требуется выполнение других фреймов того же фреймсета. В этом случае данный фрейм может понизить приоритет своей задачи, что даст возможность его прерывания другими фреймами. Например:

```
frameset("Команды", 1)
{
     frame("Привет")
     {
          PlaySpeech("Здравствуйте");
     }

     frame("Покажи каталог")
     {
          SetTaskPrior(-1);   // снизить приорите. Теперь
                              // если во время выполнения
                              // сказать «Привет», то он
                              // выполниться мгновенно, не
                              // дожидаясь завершения данного
                              // фрейма
          SetBrowserURL("page1.php");
          PlaySpeech("Это страница 1");
          Sleep(5000);

          SetBrowserURL("page2.php");
          PlaySpeech("Это страница 2");
          Sleep(5000);
```

```
            SetBrowserURL("page3.php");
            PlaySpeech("Это страница 3");
            Sleep(5000);
        }
}

while(true) sync();
```

4.18. Получение строки нераспознанного запроса

Если текстовый запрос, поступивший на вход системы, не активизировал ни один из фреймов, то формируется событие «* notFound», а текст этого запроса можно получить с помощью функции **NotFoundText()**.

4.19. Получение события начала распознавания

При начале детектирования речи возникает событие «* SPEECH».

5. Стандартные события

Фреймообразные структуры языка iScript используются для обработки запросов и команд управления всем роботом. Все события, формируемые встроенными системами робота, командами сервера управления, с WEB-страниц экранного контента, специальных команд с пульта управления, а также командами с консоли поступают в виде запросов во фреймовую структуру языка iScript.

Каждый такой запрос представляет собой строку.

Тексты запросов всех события начинаются со знака «* », что не дает их спутать с командами, формируемыми системой распознавания речи. Такая система с голоса знак «* » никогда не формирует. Следует обратить внимание, что после знака «*» следует хоты бы один пробел.

Перечень встроенных в робота событий и их описание приведен в Табл. 1.

Табл. 1 – перечень встроенных в робота событий

Событие	***Описание***
* notFound	Текстовый запрос не активизировал ни один из фреймов. Фразу можно получить с помощью функции NotFoundText()
* noRecog	Робот не расслышал голосовую фразу
* faceIn	Система трекинга лиц обнаружила лицо
* faceOut	Система трекинга лиц обнаружила пропадание лица
* beep	Система управления движением робота обнаружила на пути препятствие (возможно человека), и просит проехать.
* obstacle	Система управления движением сформировала на карте препятствие
* navigate	В полноэкранном браузере контента робота открыли новую страницу (URL страницы можно получить с помощью функции GetBrowserURL())
* SPEECH	Событие начала детектирования речи
* ROBOTINPUTS <n>	Изменение состояния датчиков робота. <n> – новая маска состояния

Пример обработки событий:

```
frameset("events", 100)
{
    frame("* faceIn")
    {
        PlaySpeech("Здравствуйте", "Приветствую Вас");
    }
    frame("* faceOut")
    {
        PlaySpeech("До свидания");
    }
}
FaceTracker(true); // включить систему детектирования лиц
while(1) sync();
```

6. Функции управления роботом

6.1. Управление аппаратными средствами

Функция sync

```
function sync();
```

Функция синхронизирует работу системы скриптов, позволяя в момент вызова обработать поступившие входные сообщения и обслужить таймеры.

Данную функцию следует периодически вызывать в длинных циклах.

Пример:

```
PlaySpeech("Привет мир!");
while(PlaySpeech())
{
   sync();
}
```

Функция SetWheelSpeed

Функция приводит шасси робота в движение.

```
function SetWheelSpeed( v, q, s, mode );
```

или

```
function SetWheelSpeed( v, q, s );
```

или

```
function SetWheelSpeed( v, q );
```

или

```
function SetWheelSpeed( v );
```

Где:

v – линейная скорость движения робота -127…127. Значение 0 – нейтральное положение.

q – скорость поворота робота -127…127. Значение 0 – нейтральное положение. По умолчанию q=0.

s – скорость стрейфа робота (если робот поддерживает стрейф). Диапазон -127…127. Нейтральное положение 0. По умолчанию s = 0.

mode – режим шасси (если поддерживает робот). 0 – режим движения (режим по умолчанию). 1 – режим поворота на месте. 2 – режим стрейфа.

Внимание! На некоторых роботах заданные скорости движения автоматически не обнуляется. Если роботу один раз была дана команда движения шасси, то робот будет двигаться, пока не поступит другая команда управления шасси. Рекомендуется для гарантированной остановки робота повторять команду с нулевой скоростью несколько раз.

Функция GetRangefinder

```
function GetRangefinder(number);
```

Функция возвращает показания дальномера *number* в сантиметрах (float), где *number* виртуальный номер дальномера, зарегистрированного в конфигурационном файле. Если препятствий нет, то возвращает дальность 400 см. Рекомендуется ограничиться дальность 150 см.

Функция GetSideDistance

```
function GetSideDistance(side);
```

Функция получает дальность в сантиметрах объектов, в направлении side. Параметр side может принимать следующие значения:

- 0 – слева;
- 1 – прямо;
- 2 – права;
- 3 – назад.

Если препятствий нет, то возвращает дальность 400 см. Рекомендуется ограничиться дальность 150 см.

Функция GetBrowserURL

```
function GetBrowserURL();
```

Возвращает текущую страницу в полноэкранном браузере контента. Страница должна быть открыта не раньше запуска программы dynrobot, т.к. адрес открытой страницы отправляется в робота единожды в момент открытия.

Пример:

```
frameset("События", 100)
{
   frame(* navigate")
   {
        var url = GetBrowserURL();
        if (url == "http://content/show.php")
        {
            ...
        }
   }
}
...
while(1) sync();
```

Функция SetBrowserURL

```
function SetBrowserURL(url);
```

Устанавливает URL в полноэкранном браузере контента.

Пример:

```
SetBrowserURL("http://content/photoservices/photoservice.php");
```

Функция URLEncode

```
function URLEncode(url);
```

Экранирует специальные символы для URL.

Пример:

```
SetBrowserURL("a.php?text="+URLEncode("Привет мир!"));
```

Функция PlayWave

Функция проигрывает указанный wave-файл.

```
function PlayWave(file);
```

Здесь:

file – строка с названием wave-файла. Путь задается относительно основного файла программы.

Функция AudioVolume

Функция устанавливает или получает текущую громкость звука на роботе.

```
function AudioVolume(volume);
```

или

```
function AudioVolume();
```

Здесь:

volume – громкость в процентах (0-100).

Если параметр volume не указан, функция лишь возвращает текущий уровень громкости.

Во всех случаях функция возвращает текущий уровень громкости или -1. В случае использования данной функции с параметром (для установки громкости) возвращается вновь установленная громкость.

Функция SetMimic

Функция устанавливает мимику лица.

```
function SetMimic(n);
```

Здесь:

n – номер мимики:

0 – нет мимики. Лицо нейтрально и может совершать произвольные действия.
1 – нет мимики, но лицо захвачено для произвольных действий.
2 – открыть рот для буквы «А».
3 – открыть рот для буквы «Е».
4 – открыть рот для буквы «О».
5 – открыть рот для буквы «У».
6 – закрыть глаза.
7 – посмотреть глазами влево.
8 – посмотреть глазами вправо.

По окончанию манипуляций с мимикой лицо следует вернуть в стадию 0.

Функция Lift

Функция управления подъемником корпуса (если он есть у робота). В зависимости от параметров выполняет те или иные функции:

Остановить подъемник:

```
function Lift(false);
```

Установить высоту подъемника в сантиметрах:

```
function Lift(height);
```

Определить текущую высоту подъемника (в сантиметрах):

```
function Lift();
```

Во всех случаях функция возвращает высоту подъемника в сантиметрах.

Функция SetLiftSpeed

Функция устанавливает скорость (и направление) движение подъемника корпуса робота (если он есть).

```
function SetLiftSpeed(speed);
```

Где: speech – скорость от -127 до 127. Скорость 0 соответствует остановки подъемника.

Функция Tray

Функция управления подносом робота (если он есть). В зависимости от параметров выполняет те или иные функции:

Установить режим автоматического баланса подноса:

```
function Tray(true);
```

Отключить режим автоматического баланса подноса:

```
function Tray(false);
```

Установить скорость движения привода баланса подноса:

```
function Tray(speed);
```

Здесь: speed – скорость от -127 до 127. Значение скорости 0 соответствует остановке привода.

Функция GetAccelerometer

Функция получает показания акселерометра подноса робота (если он есть). Функцию следует вызывать циклически, чтобы пробудить процесс опроса акселерометра.

```
function GetAccelerometer();
```

Функция возвращает показания акселерометра в условных единицах от -255 до 255. Значение 0 означает ровно. Значение 0x7FFFFFFF соответствует временному или постоянному отсутствию данных.

Функция GetRadioButton

Система поддерживает 4 вида радиокнопки:

- специальный драйвер на COM-порт.
- радиомышка (кнопки 1, 2, 3).
- HTTP-радиокнопка (приложение для смартфона по адресу: http://<IP-робота>:5000/btn.html
- лазерная указка для презентаций Wireless Presenter.

Тип радиокнопки задается в конфигурации робота в файле config.txt параметром: RADIO_BUTTON_DRIVER.

Функция, определяющая нажатия радиокнопки в iScript, имеет следующий формат:

```
function GetRadioButton();
```

Функция возвращает статус (int) нажатия радиокнопки, если ее драйвер подключен. Возвращаемые значения представляют собой битовую маску, в которой:

- бит 0 – признан нажатия кнопки 1 (маска 1).
- бит 1 – признак нажатия кнопки 2 (маска 2).
- бит 2 – признак нажатия кнопки 3 (маска 4).

– бит 3 – признак нажатия кнопки 4 (маска 8).

Значение 0 соответствует всем отжатым кнопкам.

Например:

```
// ждать нажатия кнопки 1
while((GetRadioButton() & 1)==0) sync();

// ждать отпускания всех кнопок
while(GetRadioButton()) sync();

...

// ждать нажатия кнопки 3
while((GetRadioButton() & 4)==0) sync();

// ждать отпускания кнопки 3
while(GetRadioButton() & 4) sync();
...

// ждать нажатия любой кнопки
while(GetRadioButton()==0) sync();

// определить какая кнопка нажата
if (GetRadioButton() & 1)  // кнопка 1
{
   ...
}
else
if (GetRadioButton() & 2) // кнопка 2
{
   ...
}
else
if (GetRadioButton() & 4) // кнопка 3
{
   ...
}
else
if (GetRadioButton() & 8) // кнопка 4
{
   ...
}
```

Функция GetBattery

```
function GetBattery();
```

Функция возвращает текущий заряд батареи в процентах. Если робот не поддерживает данную функцию, функция возвращает (-1).

Функция GetChargeSignal

```
function GetChargeSignal();
```

Функция возвращает уровень сигнала наведения на зарядное устройство (если функция поддерживается роботом). Показания находятся в диапазоне от 0 до 1023, однако, реальный уровень фонового излучения и максимального уровня сигнала может

быть для каждого робота уникальным. Сигнал тем мощнее, чем меньше возвращаемое значение. Если сигнала нет вообще, то возвращаемое значение близко к 1023.

Функция IsChargeConnected

```
function IsChargeConnected();
```

Функция возвращает true, если робот подключен к зарядному устройству (если эта функция поддерживается роботом).

Функция GetSensor

Функция возвращает показание сенсора *n* робота.

```
function GetSensor(sensor);
```

Где:

sensor – (int) номер сенсора.

Функция возвращает показание в формате float или null, если сенсора не существует.

У некоторых роботов (например, у TrueBot) имеется набор дополнительных сенсоров, напрямую не используемых самой программой «ДинРобот». Для чтения показания этих датчиков используется данная функция. Конкретный перечень номеров датчиков зависит от конкретной модели робота. Для робота TrueBot, использующего контроллер KatyaBoard, этот перечень следующий:

- 0 – датчик прикосновения 0 (0 нет касания; 1, есть касание).
- 1 – датчик прикосновения 1 (0 нет касания; 1, есть касание).
- 2 – относительная влажность воздуха в процентах или 0, если ошибка или нет данных.
- 3 – температура воздуха по датчику 1 (°C) или -300, если ошибка или нет данных.
- 4 – температура точки росы (°C) или -300, если ошибка или нет данных.
- 5 – модуль разницы между показаниями датчика 3 и 4 при условии отсутствия ошибки в их показаниях. В случае ошибки возвращает 300. По сути, получается значение, показывающее, как близко температура воздуха приблизилась к точке образования росы, что может быть страшно для эксплуатации робота.
- 6 – атмосферное давление в мм. рт. ст.
- 7 – температура воздуха по датчику 2 (°C).

Пример:

```
lastWarningTime = 0; // время последнего предупреждения

// главный цикл программы
while(1)
{
  // ...  - выполнение чего-то полезного
  if (GetSensor(5) < 3.0)
  {
    // каждые 20 секунд выдавать предупреждение
    if (GetTickCount() - lastWarningTime > 20000)
    {
      lastWarningTime = GetTickCount();
      PlaySpeech("Опасность образования росы, выключайте меня");
    }
```

```
  }
}
```

Функция SetSensorEvent

Функция назначает событие на то или иное показания сенсора робота.

```
function SetSensorEvent(sensor, value, op, event);
```

Где:

sensor – (int) номер сенсора робота (см. Функция GetSensor).

value – (float) значение датчика, с которым производится сравнение.

op – (string) строка с операцией сравнения:

- "<" – показания датчика меньше value.
- "<=" – показания датчика меньше или равно value.
- ">=" – показания датчика больше или равно value.
- ">" – показания датчика больше value.
- "!=" или "<>"– показания датчика не равно value.
- "==" или "="– показания датчика равно value.

event – (string) строка с событием, генерируемым во фреймовую структуру обработчики событий. Строка должка начинаться с символов «* » (звездочка, пробел). Например: «* TOUCH 1».

Функция возвращает собственный дескриптор, используемый при удалении данного назначения с помощью функции ClearSensorEvent (см. далее).

При назначении события система автоматически отслеживает показания сенсоров робота на каждом такте расчета робота и при возникновении условий срабатывания события выставляет *признак запроса события*, а также *признак запрета на повторное возникновение события*.

Как только iScript входит в стадию ожидания (вызов функций синхронизации sync, delay и т.п.), производится опрос *признаков запросов событий*. Если такой признак есть, то генерируется назначенное событие во фреймовую структуру обработки событий.

Признак запрета на повторное возникновение события позволяет исключить повторное возникновение события, пока условия возникновения события остаются справедливы.

Как только условия возникновения события становятся ложными, признак запрета на повторное возникновение события сбрасывается.

Следует отметить, что между вызовами функции синхронизации sync, delay и т.п. может многократно возникать и пропадать условия возникновения события. Однако событие в этом случае будет сгенерировано лишь один раз. Чтобы событие возникло заново, необходимо чтобы оно возникло вновь уже после его обработки.

Допускается установка нескольких событий на разные показания одного и того же датчика.

Пример:

```
// фреймсет обработчики событий
frameset("events", 100)
{
    // обработчик события пользователя
    frame("* My Event")
    {
        PlaySpeech("Не надо меня трогать");
        while(PlaySpeech()) sync();
    }
}
```

```
// установить событие на срабатывание датчика прикоснования
SetSensorEvent(1, 0.0, "!=", "* My Event");

while(true) sync();// главный цикл программы
```

Функция ClearSensorEvent

Функция удаляет назначение события, назначенного с помощью функции SetSensorEvent.

```
function ClearSensorEvent(handle);
```

Где:

handle – дескриптор, полученный при вызове функции SetSensorEvent.

Пример:

```
// фреймсет обработчики событий
frameset("events", 100)
{
    // обработчик события пользователя
    frame("* My Event")
    {
        PlaySpeech("Не надо меня трогать");
        ClearSensorEvent(myEvent); // удалить назначение события
    }
}
// установить событие на срабатывание датчика прикоснования
myEvent = SetSensorEvent(1, 0.0, "!=", "* My Event");

while(true) sync();// главный цикл программы
```

Функция CheckTray

Функция проверяет наличие предметов на подносе (если это поддерживается роботом).

```
function CheckTray();
```

Функция возвращает:

-1 – система обнаружения объектов на подносе еще не готова.

0 – объектов на подносе нет.

1 – на подносе есть какие-то объекты.

2 – на подносе есть движения.

Функцию следует вызывать многократно. Каждый вызов данной функции запускает на 3 секунды камеру подноса робота. Однако изображение с камеры приходит не мгновенно, а первые ее кадры могут быть плохого качества. Поэтому функция первые секунды после вызова возвращает -1 (система не готова).

Пример:

```
// ждать готовности системы
while(CheckTray()==-1) sync();
// проверить наличие объектов на подносе
if (CheckTray())
   print("На подносе что-то есть\n");
else
   print("Поднос пуст");
```

Функция ScanQRCodes

Функция сканирует QR-коды с указанной камеры.

```
function ScanQRCodes(cameraIndex);
```

Функция возвращает массив объектов с распознанными QR-кодами или null в случае ошибки. Каждый элемент массив содержит объект со следующими полями:

.text – строка с распознанным текстом.

.angleX – направление на центр QR-кода по оси X в градусах относительно визуальной оси камеры. Положительное направление – направо. Ноль – по центру изображения.

.angleY – направление на центр QR-кода по оси Y в градусах относительно визуальной оси камеры. Положительное направление – вверх. Ноль – по центру изображения.

.size – размер в процентах от экрана.

.rot – угол поворота QR-кода в градусах. 0 – вверх.

Функция должна вызываться периодически. При этом первые вызовы функции, скорее всего, ничего вернут, т.к. первый вызов функции лишь подключит детектор QR-кодов к камере, затем камера будет включаться и давать первые нечеткие кадры. Рекомендуется вызывать функцию в цикле течение 2-3 секунд.

Если прекратить вызывать функцию, то через 2 секунды сканер QR-кодов отключается от камеры.

Например:

```
// цикл за 3 секунды
var startT = GetTickCount();
while(GetTickCount() - startT < 3000)
{
     // получить QR-коды с камеры 0
     var qrArray = ScanQRCodes(0);
     if (qrArray && count(qrArray) > 0)
     {
          // вывести список обнаруженных QR-кодов
          for(var i=0; i < count(qrArray); i++)
          {
             print("--------- ", i , "----------\n");
             print("text:", qrArray[i].text, "\n");
             print("angleX:", qrArray[i].angleX, "\n");
             print("angleY:", qrArray[i].angleY, "\n");
          }
     }
     sync(); // синхронизация
}
```

Функция Head

Функция устанавливает угол поворота головы. В зависимости от параметров выполняет различные действия.

Установить угол поворота головы:

```
function Head( angle );
```

Где:

angle – угол поворота головы в градусах.

Определяет текущий угол поворота головы:

```
function Head();
```

Функция при любых параметрах возвращает текущий заданный угол поворота головы в градусах.

Функция LockFaceTrackerHead

Функция блокирует поворот головы системы слежения за лицами. Актуально для роботов, имеющих функцию фотографирования, и камера которых расположена на голове.

```
function LockFaceTrackerHead( lock );
```

или

```
function LockFaceTrackerHead();
```

Где:

lock – true – заблокировать голову. false – разблокировать голову. Если параметр не указан, то функция лишь определяет состояние статуса блокировки головы.

Функция при любых параметрах возвращает текущий статус блокировки головы (bool).

Функция CanUseLiftByFaceTracker

Функция определяет состояние или устанавливает состояние признака разрешения использования подъемника системой трекинга лиц.

```
function CanUseLiftByFaceTracker( canUse );
```

или:

```
function CanUseLiftByFaceTracker();
```

Где:

canUse – (true/false) разрешение использования подъемника.

Функция при любых параметрах возвращает текущий признак использования подъемника.

Функция применяется только для роботов, у которых есть подъемник, как специально обозначенное устройство.

Функция CalibMotor

Функция запускает калибровку привода робота.

```
function CalibMotor(motor);
```

или

```
function CalibMotor(motor, big);
```

Здесь:

motor – номер привода:

1-9 –колесо 1, 2, 3…

10 – подъемник (лифт).

11 – поднос.
12–15 – оси головы (шеи) 1,2,3….
40-59 – оси левой руки 1,2,3….
60-79 – оси правой руки 1,2,3…

big – (bool) полная калибровка (true) или быстрая калибровка (false).

Функция возвращает true, если привод существует и можно запустить его калибровку. Иначе возвращает false.

Функция CalibrationInprogress

Функция возвращает true, если робот находится в режиме калибровки.

```
function CalibrationInprogress();
```

Функция SetMotorPWM

Функция запускает движение заданного мотора с заданным значением ШИМ.

```
function SetMotorPWM(motor, pwm);
```

Здесь:

motor – номер привода:
1-9 – колесо 1, 2, 3…
10 – подъемник (лифт).
11 – поднос.
12–15 – оси головы (шеи) 1,2,3….
40-59 – оси левой руки 1,2,3….
60-79 – оси правой руки 1,2,3…

pwm – (int) значение ШИМ в диапазоне -127…+127.

Функция возвращает true, если привод существует и можно запустить. Иначе возвращает false.

Функция Servo

Функция задает сырое значение угла поворота сервопривода motor. Функция применяется только для степеней робота, оснащенных сервоприводами, управляемых по положению.

```
function Servo(motor, value);
```

В данном случае функция возвращает true в случае успеха и false в противном случае.

Или:

```
function Servo(motor);
```

В данном случае функция возвращает текущую уставку или показание датчика сервопривода (в зависимости от модели робота). При одиночном вызове на некоторых роботах может возвращать неактуальные показания датчика.

Здесь:

motor – (int) номер привода, зависит от робота.
value – (int) задание на сервопривод (сырое значение).

Функция LockServo

Функция захватывает управление групповыми сервомоторами, для тех роботов, где имеет место такое управление. Работает совместно с функцией Servo.

```
function LockServo(lock);
```

Здесь:

value – (bool) захват (true) или отпускание (false).

В некоторых роботах (например, для рук робота модели «Настя») на аппаратном уровне нет команды управления одним единственными приводом. Есть команда установки положения, например, сразу всех приводов одной руки. В то время, как функция Servo имеет формат установки положения только для одного отдельного привода.

Чтобы управлять такими приводами при помощи функции Servo, необходимо произвести захват управления такими групповыми сервомоторами с помощью LockServo(true). Затем несколько раз вызвать функцию Servo для каждого звена, после чего отпустить управление групповыми сервомоторами с помощью функции LockServo(false). В этом случае команда управления будет выслана один раз при отпускании управления.

Следует отметить, что блокировка тех или иных приводов зависит от драйвера робота. Некоторые приводы будут управляться независимо от наличия или отсутствия такого захвата. Например, на роботах модели «Настя» под групповой захват попадают только приводы рук, а управление сервоприводом головы и подъемника не подпадает под такой захват.

Также использование данной функции актуально для роботов модели TrueBot, если нужно синхронно отправить команду нескольким сервомоторам.

Пример:

Пример:

```
// захватить групповое управление сервомоторами
LockServo(true);

// установить положение рук (для робота «Настя»)
Servo(40, 2000);
Servo(41, 15);
Servo(42, 100);

Servo(60, -300);
Servo(61, 0);
Servo(62, 150);

// отпустить управление и отправить команду управления
LockServo(false);
```

Функция WriteRobotGesture

Функция запоминает текущее положение звеньев робота в качестве жеста с определенным номером.

```
function WriteRobotGesture(gesture);
```

Здесь:

gesture – номер жеста от 0 до 65535.

Функция возвращает true, если жест был записан.

Звенья, жесты которых запоминаются, определяются конкретной моделью робота. Впервые введено в роботе модели «Настя», в которой записываются положения рук робота.

Функция RunRobotGesture

Функция запускает движение звеньев робота для принятия им указанного жеста.

```
function RunRobotGesture(gesture);
```

Здесь:

gesture – номер жеста от 0 до 65535.

Функция возвращает true, если жест существует.

Номера жестов робота запоминаются либо с помощью функции WriteRobotGesture, либо с помощью пульта управления роботом.

Звенья, жесты которых запоминаются, определяются конкретной моделью робота. Впервые введено в роботе модели «Настя», в которой записываются положения рук робота.

Функция Cyclogram

Функция запускает циклограммы и проверяет статус их выполнения.

```
function Cyclogram(fileName, startTime, endTime);
```

или

```
function Cyclogram(fileName, startTime);
```

или

```
function Cyclogram(fileName);
```

или

```
function Cyclogram();
```

Здесь:

fileName – полное имя файла циклограммы, включая путь и расширение файла (путь задается либо абсолютный, либо относительно папки DynRobot). Если имя файла имеет пустое значение или значение null, то циклограмма завершается.

startTime – время начала фрагмента циклограммы, с которого следует начать выполнение (мс). По умолчанию 0.

endTime – время конца фрагмента циклограммы, на котором следует начать завершить выполнение циклограммы (мс). Значение 0 соответствует выполнению всей циклограммы целиком. По умолчанию 0.

При вызове без параметров функция проверяет статус циклограммы (запущена или нет). Функция возвращает true, если в данный момент циклограмма выполняется.

Если на момент вызова функции с заданным параметром fileName выполнялась какая-либо циклограмма, то ее выполнение останавливается.

Следует отметить, что если циклограмма имеет признак зацикливания (см. редактор циклограмм), то она не завершается никогда, независимо от параметров startTime и endTime. Остановить такую циклограмму можно только вызовом функции Cyclogram с указанием пустой строки или названием другой циклограммы.

Пример:

```
// запустить циклограмму
Cyclogram("my.cyc");

// ждать ее завершения
while(Cyclogram()) sync();

// запустить циклограмму poly.cyc, начиная с 3-ей секунды
Cyclogram("poly.cyc", 3000);
```

```
// ждать ее завершения
while(Cyclogram()) sync();

// запустить циклограмму poly.cyc, начиная до 4-ой секунды
Cyclogram("poly.cyc", 0, 4000);

// ждать ее завершения
while(Cyclogram()) sync();

// запустить циклограмму repeat.cyc
Cyclogram("repeat.cyc");
sleep(5000);    // подождать 5 секунд
Cyclogram("");  // остановить циклограмму
```

6.2.Функции управления речью робота

Функция PlaySpeech

```
function PlaySpeech(text1,…, textn);
```

или для определения состояния синтезатора речи:

```
function PlaySpeech();
```

или для остановки воспроизведения:

```
function PlaySpeech(false);
```

Произносит с помощью синтезатора речи одну их перечисленных фраз. Фраза выбирается случайным образом.

Пример:

```
PlaySpeech("Привет", "Здравствуйте");
```

Функция не дожидается окончания фразы и выходит сразу же после отправки на синтезатор речи очередной фразы. При этом, если синтезатор речи занят, то фраза ставится в очередь произнесения.

Вызов функции без параметров возвращает true, если синтезатор речи в данный момент произносит фразу, или false, если синтезатор речи освободился. Например:

```
PlaySpeech("Привет мир!");  // начать говорить
while(PlaySpeech()) sync(); // дождаться окончания фразы
```

Для управления голосом синтезатором Microsoft TTS в тексте можно использовать знаки:

«>» – поднять тон голоса;

«<» – опустить тон голоса;

«~» – нормальный тон голоса.

Кроме этого можно использовать знаки:

«#n» – установить жест *n*.

«#:Циклограмма» – запустить циклограмму (см. ниже)

«:)» – улыбнуться в конце фразы (если аватарка поддерживает улыбку).

":(» – загрустить в конце фразы (если аватарка это поддерживает).

"<3» – сделать глаза сердечками в конце фразы (если аватарка это поддерживает).

«*» – замедлить произнесение слова, заключенного между знаками звездочки, например: «*Ой* . Ну и денёк».

«(R:вариант1|вариант2|…|вариант$_n$)» – вставить один из вариантов (случайно выбранный).

«(G:вариантМ|вариантЖ)» – вставить один из вариантов в зависимости от пола робота, заданного через config.txt параметром ROBOT_GENDER или через функцию RobotGender(…) (соответственно «вариантМ» для мужского пола, «вариантЖ» – для женского).

Дополнительные пробелы в тексте задают длину паузы между словами.

В синтезаторе голоса ЦРТ управление тоном голоса не поддерживается.

В текст также можно вставлять название переменных в фигурных скобках, например:

```
PlaySpeech("Меня зовут {robotName}");
```

Где robotName – переменная, в которой хранится строка с именем робота.

Выбор случайного варианта:

```
PlaySpeech("(R:Хорошо|Понятно|Как скажите), обнулино");
```

Выбор вариант в зависимости от пола робота:.

```
PlaySpeech("(G:Понял|Поняла), развернуться");
```

По аналогии с текстом, набираемым оператором с пульта управления, в тексте можно использовать символы управление жестами «#».

```
PlaySpeech("#4 Приветствую Вас #3, раз вы ещё здесь");
```

Функции PlaySpeech вместо одной или нескольких фраз может запустить циклограмму. Для этого вместо фразы следует указать “#:fileName”, где fileName – название файла циклограммы без расширения. Например:

```
PlaySpeech("#:cyclogram1 ", "#:cyclogram2","Привет");
```

В данном примере будет производиться случайный выбор между запуском циклограммы хранящейся в файле «cyclogram1.cyc» или в файле «cyclogram2.cyc» или воспроизводится фаза «Привет».

Функция RobotGender

Функция возвращает или устанавливает пол робота.

```
function RobotGender(gender);
```

или

```
function RobotGender();
```

Где:

gender – (int) пол робота. Рекомендуется 0 – Муж. 1 – Женский.

Функция всегда возвращает текущий номер пола робота.

Пол робота по умолчанию прописан в файле config.txt. параметром ROBOT_GENDER.

Пример:

```
RobotGender(1); // установить женский пол
// скажать в зависимости от пола робота
//    «Рад видеть вас» или «Рада видеть вас»
PlaySpeech("(G:Рад|Рада) видеть Вас");

// ожидать окончания
while(PlaySpeech()) sync();
```

Функция RobotSilenceTime

Функция возвращает время в миллисекундах, прошедшее с последней фразы робота.

```
function RobotSilenceTime();
```

Функция возвращает время (int) в миллисекундах.

При запуске данное время инициализируется 200 секундами (200 000 мс).

Функция HumanSilenceTime

Функция возвращает время в миллисекундах, прошедшее с момента распознавания последней фразы пользователя

```
function HumanSilenceTime();
```

Функция возвращает время (int) в миллисекундах.

При запуске данное время инициализируется 200 секундами (200 000 мс).

При распознавании речи данный таймер сбрасывается в 0 дважды: один раз при обнаружении признаков начала речи, второй раз при окончании процесса распознавания речи.

Функция GetLastSpeech

```
function GetLastSpeech();
```

Функция возвращает последнюю строку, произнесенную синтезатором речи робота. Причем произнесенную, неважно каким способом (через скрипт, через оператора, отправленную с WEB-страницы экранного контента).

Использование функции актуально в диалогах, типа «Повтори».

Пример:

```
frameset("dialogs", 1)
{
    ... // другие фреймы
    frame("что")
    frame("чего")
    frame("как")
    frame("повтори")
    frame("что ты сказал")
    {
          PlaySpeech(GetLastSpeech());
    }
}
```

Функция SpeechRecognizion

Включает или выключает работу системы распознавания речи.

```
function SpeechRecognizion(onOff);
```

синоним:

```
function SpeechRecognition(onOff);
```

синоним:

```
function SpeechRecognizer(onOff);
```

или

```
function SpeechRecognizion(onOff, alternativeOnly);
```

синоним:

```
function SpeechRecognition(onOff, alternativeOnly);
```

синоним:

```
function SpeechRecognizer(onOff, alternativeOnly);
```

Где:

onOff – состояние системы распознавания речи (true – включить, false – выключить);

alternativeOnly – использовать (true) только альтернативную систему распознавания речи или использовать основную и альтернативную систему распознавания речи (false). Если параметр опущен или равен null, то состояние не менять.

Следует отметить, что система распознавания речи автоматически приостанавливает свою работу во время воспроизведения голоса, формируемого синтезатором речи. В противном случае робот бы слушал сам себя.

Система распознавания речи останавливается при остановке скриптов.

При остановке скриптов состояние, выставляемое признаком alternativeOnly становиться равным false.

Пример:

```
SpeechRecognizion(true);
```

Функция SetLangQueryCodePage

```
function SetLangQueryCodePage(codePage);
```

Устанавливает кодовую страницу языковых запросов. Доступные значения: 1250, 1251, 1252, 1253, 1254. Кодовая страница актуальна для смены языка речи, она влияет на преобразование прописных букв в заглавные, а также на порядок предварительной подготовке слов к распознаванию.

Пример:

```
SetLangQueryCodePage(1251);
```

Функция SetSpeechSynthVoice

```
function SetSpeechSynthVoice(driver, version, voice);
```

Устанавливает драйвер синтезатора речи и голос для него.

Здесь:

driver – драйвер синтезатора речи. Доступные значения:

SAPI – Microsoft TTS.

STC – синтезатор речи от Центра Речевых Технологий.

Wave – синтезатор речи в виде таблицы звуковых файлов.

version – версия драйвера. Если driver имеет значение SAPI, то version может принимать значения: «SPEECH» или «SPEECH_SERVER». Для driver=STC, параметр version имеет значения номер языка (начиная с нуля). Для driver = Wave параметр version игнорируется.

voice – голос. Для driver=SAPI параметр voice является частью строки названия голоса (по принципу вхождения части строки в строку с названием голоса). Для driver=Wave параметр voice является названием файла с таблицей wave-файлов. Для driver=STC параметр voice является номером голоса (начиная с 0).

Пример:

```
SetSpeechSynthVoice("SAPI", "SPEECH_SERVER", "Elena");
```

Или:

```
SetSpeechSynthVoice("STC", 0, 0);
```

Функция SetSpeechSynthLanguage

```
function SetSpeechSynthLanguage(lang);
```

Устанавливает язык для синтезатора речи. Функция аналогична функции SetSpeechSynthVoice, за исключением того, что все параметры для нее берутся из конфигурационного файла config.txt из параметров:

*lang*_LANGUAGE_DRIVER
*lang*_LANGUAGE_DRIVER_VERSION
*lang*_LANGUAGE_VOICE
Здесь:
lang – параметр, переданный функции SetSpeechSynthLanguage.

Пример:

```
SetSpeechSynthLanguage("RU");
```

6.3.Функции состояния аппаратных ошибок

Функция CalibrationInprogress

Функция возвращает true, если робот находится в процессе калибровки.

```
function CalibrationInprogress();
```

Функция IsRobotBlocked

Функция возвращает true, если робот заблокирован. Периодически некоторые роботы формируют данный сигнал, когда одно или несколько их колес не может двигаться, вероятнее всего, из-за того, что робот во что-то уперся.

```
function IsRobotBlocked();
```

Функция IsRobotEmergency

Функция возвращает true, если на роботе нажата кнопка авария. Большинство роботов определяет это по факту долговременного отсутствия изменения показаний датчиков исполнительных механизмов, в то время, как команды управления на них идут.

```
function IsRobotEmergency();
```

Функция IsRobotHardwareError

Функция возвращает true, если на роботе произошла какая-либо серьезная аппаратная ошибка. Подразумевается, что саму проблему специалист может выявить в консоли или в журнале ошибок, поэтому в скрипт выносится лишь сам факт появления этой проблемы, чтобы, например, голосом сообщить об этом.

```
function IsRobotHardwareError();
```

Функция IsCommandServerError

Функция возвращает true, если наблюдается двукратная (или более) ошибка связи с командным сервером управления. Ошибка возникает только, если связь сервером активна и не запрещена лицензией на Дин-Робот.

```
function IsCommandServerError();
```

6.4.Функции интеллектуального движения

Функция Go

Функция управляет движением робота. В зависимости от параметров выполняет различные функции.

Начало движения в место с номером *place*, используя навигационную карту.

```
function Go(place);
```

Функция не дожидается завершения движения. Само движение производится асинхронно.

Остановка движения:

```
function Go(false);
```

Проверить наличие движения.

```
function Go();
```

При всех вариантах возвращает признак наличия движения:

- (-1) – движение завершилось неудачей.
- 0 – движение закончилось.
- 1 – робот в процессе движения.

Использование функции должно быть разрешено в лицензии на Дин-Робот.

Пример:

```
Go(2);    // начать движение в место номер 2

// цикл пока робот двигается
while(Go()==1)
{
   if (newCmd)
   {
      // остановить движение
      Go(false);
      ...
   }
   sync();
}

// проверить ошибку по завершению движения
if (Go()==-1) print("Ошибка\n");
```

Функция GoToPlace

Функция отправляет робота в указанное место *place* на карте, дожидаясь завершения движения.

```
function GoToPlace(place);
```

Возвращает true – в случае удачного прихода робота в указанное место, иначе возвращает false.

Использование функции должно быть разрешено лицензией Дин-Робот.

Пример:

```
if (!GoToPlace(2))   // движение в место номер 2
{
     print("Ошибка");
}
```

Функция Wandering

Функция включает или выключает режим случайных блужданий (должно быть разрешено в лицензии на Дин-Робот).

```
function Wandering(onOff);
```

или

```
function Wandering();
```

Где:

onOff – параметр, определяющий включение (true) или выключение (false) системы.

Функция возвращает состояние (bool) режима случайных блужданий.

В данном режиме движение робота осуществляется исключительно по дальномерам (почти, как робот-пылесос). Функция самостоятельно не детектирует лица или иную информацию. Пользователь должен самостоятельно параллельно данному режиму детектировать лица или события.

Пример:

```
FaceTracker(true); // включить распознавание лиц
Wandering(true);   // включить случайные блуждания робота
while(true)
{
   // если лицо то остановиться
   if (IsFace())
   {
      Wandering(false); // остановить робота

      PlaySpeech("Добрый день");

      // пока есть лицо
      while(IsFace())
      {
          sync(); // синхронизация
      }
      PlaySpeech("До свидания");
      Wandering(true); // включить случайные блуждания робота
   }
   sync(); // синхронизация
}
```

Функция Park

Функция управляет режимом парковки робота на зарядное устройство (если поддерживается роботом). В зависимости от параметров выполняет различные действия:

Запускает процесс автоматической парковки.

```
function Park(true);
```

Останавливает процесс автоматической парковки.

```
function Park(false);
```

Определяет статус завершение процесса автоматической парковки.

```
function Park();
```

Функция во всех вариантах возвращает 0 – парковка завершена или не активна. 1 – парковка не завершена. (-1) – не удалось завершить парковку.

Например:

```
Park(true); // начать парковку

// пока парковка активна не делать ничего
while(Park()>0) sync();

// парковка завершилась
if (Park() < 0) print("Ошибка!");
```

Функция FollowQR

Функция включает или выключает режим «Следуй за QR-кодом» (движение должно быть разрешено в лицензии на «Дин-Робот»).

```
function FollowQR(onOff, QRText);
```

или

```
function FollowQR(onOff);
```

Где:

onOff – параметр, определяющий включение (true) или выключение (false) режима.

QRText – текст, который следует распознавать на QR-коде. Если в качестве QRText задается null или пустая строка, то используется QR-текст, заданный в config.txt параметром FOLLOW_QR_TEXT.

Функция возвращает:

- 0 – режим выключен.
- 1 – режим включен.
- 2 – режим активен (QR-код зафиксирован или потерян совсем недавно).

В данном режиме движение робота осуществляется за QR-кодом с соответствующим текстом. Чем дальше QR-код от робота, тем больше его скорость (нужно только чтобы робот его не потерял). Вблизи QR-кода осуществляются повороты на месте.

Для роботов, поддерживающих режим стрейфа, переворот QR-кода верхней частью вниз переводит к переводу робота в режим стрейфа на месте. Передвижение вправо-влево QR-кода при этом приводит к движению робота влево-вправо.

Пример:

```
FollowQR(true);   // включить режим «следуй за QR-кодом»

// ждать появления QR-кода
while(FollowQR()!=2) sync();

// ждать пропадания QR-кода
while(FollowQR()==2) sync();

// допустим, оператор проводил робота QR-кодом в место номер 5
SetCurrentPlace(5);
```

6.5.Функции управления навигацией

Функция GetDiffLevel

Функция получает текущий уровень различий (уровень среднеквадратичного отклонения) между текущим изображением и изображением на карте в данной точке (чем больше значение, тем меньше сходство). Если изображения похожи, то данное значение примерно меньше 500-600 ед. Максимальное значение задаётся параметром конфигурационного файла MAX_DIFF_LEVEL, умноженного 2.

```
function GetDiffLevel();
```

Функция IsUserZone

Функция возвращает true, если робот проходит по зоне 3, обозначенной на карте. В противном случае возвращает false.

```
function IsUserZone();
```

Значение функции изменяется только при автоматическом движении робота. В противном случае значение функции сохраняет свое предыдущее значение.

Пример:

```
first = true;
Go(2);          // отправить робота в точку 2
while(Go()==1)  // пока робот едет
{
    sync();
    if (IsUserZone())  // въехал ли робот в зону 3
    {
        if (first)  // если впервые въехал...
        {
            first = false;
            PlaySpeech("Всем привет");
        }
    }
}
// проверить ошибку по завершению движения
if (Go()==-1) print("Ошибка\n");
```

Функция GetUserZoneParam

Функция возвращает параметр, присвоенный зоне 3, обозначенной на карте. В случае, если робот не находится в зоне 3, ей не задан параметр или если автономной навигации не происходит.

```
function GetUserZoneParam();
```

Значение функции изменяется только при автоматическом движении робота. В противном случае значение функции сохраняет свое предыдущее значение.

Пример:

```
first = true;
Go(2);          // отправить робота в точку 2
while(Go()==1)  // пока робот едет
{
    sync();
```

```
    var uzParam = GetUserZoneParam();
    if (uzParam != -1)  // Если мы находимся или находились в
зоне с параметром
    {
        if (first)  // если впервые въехал...
        {
            first = false;
            PlaySpeech("Добро пожаловать в зону " + uzParam);
        }
    }
}
// проверить ошибку по завершению движения
if (Go()==-1) print("Ошибка\n");
```

Функция GetRobotCoords

```
function GetRobotCoords();
```

Функция возвращает координаты робота в виде объекта с полями:

.x – координата x.
.y – координата y;
.a – угол ориентации (градусы);
.h – высота поднятия корпуса робота.

Пример:

```
var coords = GetRobotCoords();
if (coords.x < 0.0) { ... }
```

Функция SetRobotCoords

```
function SetRobotCoords(x,y,a);
```

Функция устанавливает координаты робота:

.x – координата x.
.y – координата y;
.a – угол ориентации (градусы);

Функция GetPlaceCoords

```
function GetPlaceCoords(place);
```

Функция возвращает координаты места *place* на карте местности или null, если место не найдено. Результат формируется в виде объекта со следующими полями:

.x – координата x.
.y – координата y;
.a – угол ориентации (градусы);
.h – высота поднятия корпуса робота.

Пример:

```
var coords = GetPlaceCoords(4);  // получить координаты места 4
if (coords.x < 0.0) { ... }
```

Функция peleng

```
function peleng(place);
```

или

```
function peleng(coords);
```

или

```
function peleng(x,y);
```

Функция возвращает пеленг точки, указанной либо номером места (*place*) на карте местности, либо объектом (*coords*), содержащим координаты объекта местности, либо путем указания координат *x*, *y*.

Функция возвращает пеленг указанной точки в градусах в диапазоне от -180 до +180 градусов. Функция возвращает null, если место не найдено или объект *coords* не содержит полей с именами “*x*” и “*y*”.

Под пеленгом понимается угол направления относительно текущего положения и ориентации робота.

Здесь:

x – (float) координата x.

y – (float) координата y;

coords – (тип object) объект, содержащий поля “x” и “y” с координатами точки (наличие других полей у объекта игнорируется). Поля “x” и “y” объекта интерпретируются, как тип данных float.

place – (int) номер места на карте местности.

У функции имеется синоним «Peleng» (с заглавной буквы).

Пример:

```
var p = object();   // создать объект p
p.x = 432.1;
p.y = 43.1

// определить пеленг на объект с координатами p
var a1 = peleng(coords);

// определить пеленг место с номером 4
var a4 = peleng(4);

// определить координаты точки 5 и определить пеленг
var coords = GetPlaceCoords(5);
var a5 = peleng(coords);
```

Функция GetCurrentPlace

```
function GetCurrentPlace();
```

Функция возвращает номер текущего места, куда последний раз пришел робот. По умолчанию возвращает (-1).

Пример:

```
if (GetCurrentPlace() == 4)
{
   PlaySpeech("Это фабрика конфет");
}
```

Функция SetCurrentPlace

```
function SetCurrentPlace(place);
```

Функция перемещает на виртуальной карте местности отметку с роботом в указанное место с номером *place*. Функция обычно используется для установки начальной точки местоположения робота.

Пример:

```
SetCurrentPlace(10);
```

Функция CreateNewPlace

```
function CreateNewPlace();
```

или

```
function CreateNewPlace(number);
```

Функция создает на виртуальной карте местности новое место.

Если задан номер *number*, то производится поиск места с указанным номером, найденному месту присваиваются текущие координаты робота.

Если место не задано или не найдено по номеру, то создается новое место на карте. Данному месту присваивается номер *number*, если он задан, в противном случае номер места присваивается автоматически. Новое место принимает значение текущих координат робота.

Функция возвращает номер нового места или null в случае ошибки.

Пример:

```
frameset("Действия",1)
{
   frame("Здесь твоя начальная точка")
   {
       CreateNewPlace(1);
   }
}
```

Функция WakeUpNavigation

```
function WakeUpNavigation();
```

Функция пробуждает систему навигации робота. В противном случае во время стоянки робота навигация «засыпает», спустя 5 секунды.

Функция SetEndAngularPercision

Функция устанавливает точность позиционирования робота по углу в конечной точке маршрута.

```
function SetEndAngularPercision( angle );
```

Где:

angle – точность в градусах (double). При указании отрицательного значения используется точность по умолчанию (заданная в конфигурационном файле параметром TARGET_PELENG).

Функция имеет синоним: SetTargetPeleng.

Функция SetEndLinearPercision, SetPlacePercision

Функция устанавливает линейную точность позиционирования робота в конечной точке маршрута.

```
function SetEndLinearPercision( radius );
```

или

```
function SetEndLinearPercision( epsX, epsY );
```

или

```
function SetPlacePercision( radius );
```

или

```
function SetPlacePercision ( epsX, epsY );
```

Где:

radius – точность в сантиметрах (double). При указании отрицательного значения используется точность по умолчанию (заданная в конфигурационном файле в параметрах PLACE_PRECISION_X, PLACE_PRECISION_Y).

epsX, epsY – точность в сантиметрах (double), разложенная на ось X и Y робота. При указании отрицательного значения используется точность по умолчанию (заданная в конфигурационном файле в параметрах PLACE_PRECISION_X, PLACE_PRECISION_Y)

Функция имеет синоним: SetPlacePercision.

Следует понимать, что робот может обеспечить точность позиционирования в конечной точки порядка 1 см. Однако для достижения такой точности робот будет в течение достаточно длительного времени совершать много различных маневров в окрестности целевой точки, пытаясь встать точно. Поэтому пользователю следует выбрать здоровый компромисс между точностью и временем позиционирования робота. Возможно некоторые точки маршрута не требуют такой высокой точности позиционирования.

Функция SetEndPointRadius

Функция устанавливает радиус зоны вокруг конечной точки маршрута, в которой роботу следует принять ориентацию конечной точки маршрута, а не ориентацию в сторону ее пеленга.

При движении по маршруту робот принимает ориентацию по направлению к промежуточной или конечной точки маршрута. Однако при подходе к конечной точке маршрута следует принимать ориентацию конечного пункта маршрута.

В основном данная зона актуальна для роботов, поддерживающих стрейф. Внутри этой зоны робот принимает целевую ориентацию и для подхода использует уже стрейф и движение.

Для роботов, не поддерживающих стрейф, данную зону следует задавать очень небольшого радиуса – меньше требуемой точности позиционирования, т.к. робот должен придти в целевую точку с заданной точностью, а в ней уже принимать целевую ориентацию.

```
function SetEndPointRadius( radius );
```

Где:

radius – радиус зоны в сантиметрах (double). При указании отрицательного значения используется точность по умолчанию (заданная в конфигурационном файле в параметре END_POINT_RADIUS).

Функция SetMaxSpeed, SetFastSpeed

Функция устанавливает максимальную линейную скорость движения робота. Функции SetMaxSpeed и SetFastSpeed являются синонимами.

```
function SetMaxSpeed( speed );
```

или

```
function SetFastSpeed( speed );
```

Где:

speed – максимальая скорость движения от 0 до 127. При использовании значения -1 используется скорость по умолчанию, заданная в конфигурационном файле config.txt в параметре FAST_SPEED.

Функция UseBackMoving

Функция устанавливает или возвращает состояние признака использования обратного возврата при интеллектуальном движении.

```
function UseBackMoving( use );
```

или

```
function UseBackMoving();
```

Где:

use – (bool) признак использования режима обратного возврата (по умолчанию состояние признака не меняется).

Функция возвращает текущее состояние признака обратного возврата.

Обратный возврат используется при интеллектуальном движении по карте местности. Если маршрут робота от точки, где он находится, проходит назад робота, то при использовании данного признака, робот сначала отступает назад, а затем уже разворачивается.

В основном введено для роботов-официантов, которые стоят у столика гостей. Отход от столика для них целесообразнее делать путем небольшого отъезда назад от столика, а лишь затем разворота.

Функция MovingUseLift

Функция устанавливает или возвращает состояние признака использования высоты подъемника робота, записанного на карту.

```
function MovingUseLift ( use );
```

или

```
function MovingUseLift ();
```

Где:

use – (bool) признак использования подъемника

Функция возвращает текущее состояние данного признака.

6.6.Фото и видео и прочие сервисы робота

Функция StartVideoRecord

Функция начинает запись видео в указанный файл с указанной камеры. Функция автоматически определяет тип аргументов и выбирает одну из следующих реализация:

```
function StartVideoRecord(fileName, camera, isAudio);
```

или

```
function StartVideoRecord(fileName, camera);
```

или

```
function StartVideoRecord(fileName, isAudio);
```

или

```
function StartVideoRecord(camera, isAudio);
```

или

```
function StartVideoRecord(fileName);
```

или

```
function StartVideoRecord(camera);
```

или

```
function StartVideoRecord(isAudio);
```

Здесь:

fileName – (строка) имя файла, в который производится запись. Если строка с именем файла содержит абсолютный путь, то запись производится в указанный файл. Если путь не указан, то файл записывается в папку, заданную в конфигурационном файле config.txt параметром VIDEORECORDER_DEF_PATH (по умолчанию «video/»). Если имя файла не задано или равно пустой строке, то система автоматически подбирает имя файла, используя для его названия timestamp.

camera – (целое) номер камеры, с которой производится запись. Если камера не задана или равна (-1), то используется камера, заданная в конфигурационном файле config.txt параметром VIDEORECORDER_DEF_CAMERA (по умолчанию 0).

isAudio – (bool) признак записи аудио. По умолчанию true.

Запись видео в системе DynRobot может производиться автоматически (если в config.txt выставлен параметр VIDEORECORDER_AUTOSTART), или по команде из iScript.

Время записи одного видеоролика ограничено параметром VIDEORECORDER_MAX_RECORD_TIME конфигурационного файла config.txt. По истечению этого времени система автоматически переходит к записи следующего видеофайла, формируя его название по следующему принципу: от имени предыдущего файла удаляются все цифры перед расширением файла, а затем к имени файла приписывается номер, совпадающий с timestamp.

Например: пользователем был задан файл «myVideoRecord.avi». Система формирует название следующего файла «myVideoRecord1462741050.avi». Следующий файл будет иметь название «myVideoRecord1462741110.avi» и т.д.

Пользователю рекомендуется так составить программу на iScript, чтобы производить остановку записи видео функцией StopVideoRecord() до истечения максимального времени записи.

Не рекомендуется ставить время записи очень большим, т.к. размер одного avi-файл согласно формату AVI, ограничен 1 ГГб. Кроме того, незавершенный видеофайл в формате avi не имеет ни заголовка, ни подвала с таблицей размещения кадров, поэтому он не может быть воспроизведен системой. Поэтому полезно периодически завершать запись одного видеофайла и переходить к другому.

Формат видеофайла MJPEG, формат звука PCM 16, mono 44100 (звук без компрессии). Качество сжатия JPEG устанавливается в конфигурационном файле параметром VIDEORECORDER_QUALITY. Рекомендуется значение 75.

При невозможности воспроизведения видеофайла на сторонней машине следует установить набор кодеков ffmpeg или k-lite codec. Для записи видеофайла DynRobot использует свой собственный независимый от системы кодек.

Если видеозапись была запущена с помощью iScript, то при завершении работы iScript запись автоматически прекращается.

Пример использования:

```
PlaySpeech("Задайте ваш вопрос"); // произнести фразу
while(PlaySpeech()) sync();  // дождаться окончания произношения

// начать видеозапись
StartVideoRecord("question.avi");

var startTime = GetTickCount(); // время начала записи
var lastFaceTime = startTime; // время появления лица в кадре
while(1)
{
    var t = GetTickCount(); // текущее время

    // прервать записи через минуту
    if (t - startTime > 60000) break;

    // засечь время появления лица в кадре
    if (IsFace()) lastFaceTime = GetTickCount();

    // если лицо исчезло, через 3 секунды прекратить запись
    if (t - lastFaceTime > 3000) break;

    sync();  // синхронизация
}
StopVideoRecord();  // остановить видеозапись
PlaySpeech("Запись завершена"); // сказать фразу
```

Функция StopVideoRecord

Функция завершает запись видео.

```
function StopVideoRecord();
```

Функция SetAvatar

Функция устанавливает аватар роботу из XML-файла аватара.

```
function SetAvatar(xmlFileName);
```

Название файла аватарки эквивалентно строке AVATAR_MODEL в конфигурационном файле config.txt.

Функция возвращает true в случае успеха.

Например:

```
SetAvatar("AVATAR/Masha/model.xml");
```

Функция SetEmotion

Функция эмоцию роботу.

```
function SetEmotion(n);
```

Где:

n – номер эмоции:

- 0 – нейтральные эмоции;
- 1 – грусть;
- 2 – улыбка;
- 3 – глаза-сердечки;
- 4 – подмигнуть влево;
- 5 – подмигнуть право.

Эмоция, установленная данной функции аналогична эмоциям, отправляемым с пульта управления.

Время эмоции зависит от типа эмоции. Так грусть и улыбка устанавливаются на 12 секунд, глаза-сердечки – на 5 секунд, подмигивания на 1,5 секунды. После подмигиваний и глаз-сердечек автоматически устанавливается эмоция улыбки на 12 секунд. Грусть и улыбка, спустя 12 секунд, переходит в нейтральную эмоцию.

Функция возвращает true в случае успеха.

Например:

```
SetEmotion(2);
```

Функция SetRoboavatar

Функция устанавливает робоаватар из указанного файла с изображением лица или отменяет режим робоаватара.

```
function SetRoboavatar(fileName);
```

или

```
function SetRoboavatar("");
```

При вызове данной функции с указанием имени графического файла, функция устанавливает роботу робоаватар из файла. Поддерживаются форматы JPG, GIF, PNG, BMP.

При вызове функции с пустой строкой режим робоаватар отменяется.

Функция возвращает true в случае успеха.

Например:

```
SetAvatar("c:/photos/myface.jpg");
```

Функция SnapPhoto

Функция делает снимок и сохраняет его в файл, опционально, накладывая рамку (виньетку).

```
function SnapPhoto(camera, fileName);
```

или

```
function SnapPhoto(camera, fileName, maskFileName);
```

Здесь:

camera – номер камеры.

fileName – имя файла, куда следует сохранить фото после съемки.

maskFileName – имя файла с рамкой. Если не указан, то рамка не накладывается.

Например:

```
SnapPhoto(0, "c:/photo/photo234.jpg", "c:/mask/mask1.png");
```

Функция RoboavatarService

Функция управления сервисом Roboavatar для съемки лица с камеры.

```
function RoboavatarService(snap);
```

Здесь:

snap – параметр типа bool. Принимает значение false для режима поиска лица и значение true, для снимка и завершения режима поиска лица для робоаватара.

Функцию следует вызывать в цикле с параметром false. При этом робот будет производить поиск лица. По окончанию следует вызывать функцию с параметром true. При этом последний кадр с лицом станет робоаватаром.

Работа функции повторяет сервис «робоаватар», реализованный через WEB-интерфейс контента робота.

Если функция не вызвана с параметром true, то сервис автоматически отключается после 2 секунд бездействия.

Например:

```
PlaySpeech("Посмотрите в камеру");

// снимать в течение 3 секунды
var startTime = GetTickCount();
while(GetTickCount() - startTime < 3000)
{
    RobotavatarService(false);
    sync(); // синхронизация
}
RobotavatarService(true);
PlaySpeech("Посмотрите на мое лицо. Вы стали роботом");
```

6.7.Функции управления скриптами

Функция exec

```
function exec(command, arguments);
```

Выполняет системную команду *command*, передавая ей аргументы *arguments*. Под Linux *arguments* не используется, все параметры передаются через командную строку *command*.

Пример:

```
exec("winword", "C:\\DOCS\\doc1.docx");
```

Функция exit

```
function exit(type);
```

Выполняет выход из скрипта, программы или выключение бортового компьютера командой. В зависимости от параметра type выполняет следующие действия:

- type = 0 – выход из скриптов.
- type = 1 – завершение работы программы DynRobot.
- type = 2 – выключение бортовой ЭВМ.

Функция SetTimer

Функция устанавливает таймер, который будет с заданным интервалом выполнять указанный код.

```
function SetTimer(code, interval);
```

Здесь:

code – строка кода на iScript, который требуется выполнить по таймеру.

interval – интервал таймера (мс).

Функция возвращает идентификатор таймера. Удалить таймер можно с помощью функции KillTimer, передав ей в качестве параметра идентификатор таймера.

Внимание! Таймер может работать только в интервалах синхронизации кода, формируемого функциями sync() и sleep(…).

Например:

```
// функция, вызываемая по таймеру
function OnTimer()
{
    print("Я тут!\n");
}

// взвести таймер на 1 секунду
SetTimer("OnTimer();", 1000);

// основной цикл программы
while(1)
{
   sync();
}
```

Функция KillTimer

Функция удаляет таймер, созданный функцией SetTimer.

```
function KillTimer(timerID);
```

Здесь: timerID – идентификатор таймера, созданный функцией SetTimer.

Например:

```
// функция, вызываемая один раз по таймеру
function OnTimeout()
{
     print("Ку-ку\n");
     KillTimer(gTimerID);  // уничтожить таймер
}
// создать таймер
gTimerID = SetTimer("OnTimeout();", 60000);
// основной цикл программы
while(1)
{
   sync();
```

Функция KillAllTimers

Функция удаляет все таймеры, созданные с помощью SetTimer.

```
function KillAllTimers();
```

Функция ResetAll

Функция удаляет все фреймы и таймеры.

```
function ResetAll();
```

Функция Restart

Функция завершает работу текущих скриптов (аналогично завершению работы), а затем перезапускает скриптовую систему с «чистого листа», начиная с файла iFileName.

```
function Restart(iFileName);
```

К файлу iFileName можно указать как абсолютный путь, так и путь, относительно папки scripts папки DynRobot.

Функция аналогична команде «@RESTART *iFileName*» (без кавычек), переданной из браузера.

7. Система выявления скрытых рисков

7.1. Введение

Программный комплекс «ДинРобот» имеет встроенную систему, позволяющую роботу производить собеседование с человеком и по его ответам и его голосу выявлять скрытые особенности поведения этого человека. Система запатентована ООО «Фора Диджитал».

Для обеспечения работы системы необходимо подготовить файл вопросов, которые будет задавать робот. Файл подготавливается с помощью специального редактора «Редактор RiskControl». Список вопрос, а также сырые данные результата обследования храняться в зашифрованном виде. Для использования данном системы в составе робота необходимо приобретать лицензию у ООО «Фора Диджитал».

Перед использованием системы на роботе необходимо создать папку для хранения результатов обследования.

В конфигурационном файле config.txt следует прописать следующие параметры:

- PSY_RISK_CONTROL_OUTPUT_PATH – папка для хранения результатов обследования (слеш в конце не обязателен).
- PSY_RISK_CONTROL_DEF_QUESTION_FILE – файл со списком вопросов, используемый по умолчанию.
- PSY_RISK_CONTROL_SECONDS_PER_ANSWER – количество секунд, отводимых человеку для ответа на вопрос.
- PSY_RISK_CONTROL_START_PHRASE – фраза перед началом тестирования.
- PSY_RISK_CONTROL_END_PHRASE – фраза в конце тестирования.
- PSY_RISK_CONTROL_REPEAT_PHRASE – фраза типа, «говорите только да или нет».
- PSY_RISK_CONTROL_PAUSE_PHRASE – фраза приостановки тестирования.
- PSY_RISK_CONTROL_RESUME_PHRASE – фраза возобновления тестирования.

7.2. Функции системы выявления скрытых рисов

Функция PsyRiskControl

Функция управления встроенным модулем «RiskControl», позволяющим производить собеседование с человеком и проводить психоаналитический анализ его голоса для выявления скрытых особенностей поведения этого человека:

```
function PsyRiskControl(patient, info, questionFile);
```

или

```
function PsyRiskControl(patient, info);
```

или

```
function PsyRiskControl(patient);
```

или

```
function PsyRiskControl(state);
```

или

```
function PsyRiskControl();
```

Здесь:

patient – (string) имя обследуемого.

info – (string) дополнительная информация об обследуемом (может быть пустой строкой). По умолчанию – пустая строка.

questionFile – (string) полный путь к файлу вопросов, по которому проводиться обследование. По умолчанию используется последний заданный файл, а если и он не задан, то используется файл, заданный через конфигурационный файл **config.txt** параметром «PSY_RISK_CONTROL_QUESTION_FILE».

state – (int) устанавливаемое состояние:

- 0 – выключить систему.
- 1 – включить систему (он же снять с паузы).
- 2 – поставить на паузу.

Во всех вариантах функция возвращает (int) текущее состояние системы:

- (-1) – произошла ошибка.
- 0 – выключена.
- 1 – включена.
- 2 – на паузе.
- 3 – обследование завершено.

При вызове функции с параметром patient, info или questionFile, функция запускает новое обследование (если старое было завершено, выключено или находится в состоянии ошибки). При этом функция дожидается окончания запуска процесса обследования, и выходит сразу же, как только обследование началось. Во всех остальных случаях функция лишь устанавливает тот или иной режим без ожидания его применения.

На время работы системы система распознавания речи принудительно отключается. Однако после установки обследования на паузу или при выключении процесса обследования состояние системы распознавания речи восстанавливается.

Для проверки окончания тестирования разработчик скрипта должен периодически вызывать функцию без параметров для определения текущего состояния обследования.

В процессе обследования можно определить текущую стадию с помощью функции GetPsyRiskControlStage. С помощью функции GetPsyRiskControlFileName по завершению тестирования можно определить имя файла с результатами обследования.

В результате тестирования в папке с результатами обследования (папке, заданной параметром PSY_RISK_CONTROL_OUTPUT_PATH в файле config.txt) образуются файлы:

- *xxx*.rsl – файл с результатами обследования.
- *xxx*.mp3 – MP3-файл с записью обследования.
- *xxx*.pdf – отчет об обследовании в формате pdf.

Здесь: *xxx* – номер обследования (определяется по TIMESTAMP).

Кроме этого формируется файл: «__list__.tbl» – файл таблицы сведений об всех проведенных обследованиях. Все файлы (за исключением mp3 и pdf) храняться в зашифрованном формате. Для их расшифровки и просмотра требуется программа PsyVoiceEditor, входящую в состав основного программного комплекса АПК «RiskControl».

Пример:

```
// начать обследование
if (PsyRiskControl("Петя","", "default.qlst") != 1)
{
   print("Error!\n”);
}
else
{
   // ждать окончания обследования
```

```
   while(PsyRiskControl() != 3) sync();

   // получить pdf-файл с результатами обследоения
   var pdfFile = GetPsyRiskControlFileName(".pdf");
}
```

Функция GetPsyRiskControlStage

Функция возвращает стадию процесса обследования, проводимого с помощью модуля «RiskControl».

```
function GetPsyRiskControlStage();
```

Функция возвращает стадию (int) процесса обследования:

- 15 – ожидание завершения речи, сказанной перед запуском обследования
- 0 – пауза 2,5 секунды перед началом обследования (определение уровня шума микрофона).
- 1 – ожидание окончания стартовой фразы.
- 2 – задание вопроса.
- 3 – ожидания, пока вопрос будет задан.
- 4 – ожидание ответа пользователя.
- 10 – завершение тестирования (фраза завершения обследования).
- 11 – ожидание окончания фразы о завершении обследования.
- 12 – ожидание фразы «говорите только Да или Нет», произносимой, если ответ не распознан.

Вызов данной функции имеет смысл только, если процесс обследования был запущен с помощью функции PsyRiskControl

Функция GetPsyRiskControlFileName

Функция возвращает файл с результатами обследования.

```
function GetPsyRiskControlFileName(ext);
```

Где:

ext – (string) требуемое расширение файла, которые может принимать следующие значения:

- «.mp3» – файл аудиозаписи обследования с расширением «mp3».
- «.pdf» – файл отчета с расширением «pdf».
- «.rsl» – файл сырых данных обследования с расширением «rsl».

Функция возвращает (string) имя файла процесса обследования:

В случае ошибки функция возвращает null.

Функцию можно вызывать, только по окончанию обследования (если функция PsyRiskControl возвращает значение 3).

Пример см. функцию PsyRiskControl.

8. Связь с центральным командным сервером управления

8.1. Введение

Через конфигурационный файл config.txt можно разрешить роботу производить обмен данными с центральным сервером управления. Для этого следует задать параметр COMMAND_SERVER_HOST и COMMAND_SERVER_IP. При этом робот с интервалом, заданным в конфигурационном файле config.txt с помощью параметра COMMAND_SERVER_INTERVAL, будет производить обмен данными с этим сервером по протоколу HTTP методом POST.

Запрос на сервер производится либо по истечению указанного интервала времени, либо при появлении у робота очередных данных для отправки на сервер.

Некоторые команды на сервер (например, управления виртуальными светофорами) робот поддерживает автоматически. Отправку прочих команд следует производить из скрипта управления роботом. Прием команд сервера осуществляется в виде событий во фреймы.

Строки команд передаются через HTTP-запрос как параметр cmd. Номер робота передается в параметре robot HTTP-запроса. Если роботу нечего передать на сервер, то формируется пустой запрос с пустым текстом по протоколу HTTP. Если серверу нечего передать, то сервер формирует пустой ответ по протоколу HTTP. Например:

Запрос:

```
POST /robot.php HTTP/1.0
Host: 192.168.1.2
Content-Length: 12

robot=1&cmd=
```

Ответ сервера:

```
HTTP/1.1 200 OK
Content-Length: 0


```

Полный перечень команд обмена с сервером зависит от задачи и определяется самим пользователем и разработчиком серверного программного обеспечения.

8.2. Автоматически оправляемые команды

Если обмен с центральным сервером управления разрешен, то робот автоматически отправляет на сервер следующие команды.

Начало автоматического движения по карте

При автоматическом движении робота по карте при покидании роботом места (при начале автоматического движения).

```
ROBOT AT PLACE 0↵
```

При этом на сервер формируется следующий HTTP запрос:

```
POST /robot.php HTTP/1.0
Host: 192.168.1.2
Content-Type: application/x-www-form-urlencoded
Content-Length: 37

robot=1&cmd=ROBOT%20AT%20PLACE%200%0A
```

В ответ на запрос сервер должен передать перечень закрытых виртуальных светофоров, перечисленных через знаки пробела:

```
HTTP/1.1 200 OK
Content-Length: 21
Content-Type: text/plain; charset=UTF-8

@LIGHTTRAFFICS 1 10 8
```

Если виртуальных светофоров нет, то ответ @LIGHTTRAFFICS должен все равно формироваться. Перед началом движения (если обмен с сервером включен) робот будет ожидать ответа сервера в течение нескольких секунд.

Запрос номер места по его типу

На виртуальной карте в памяти робота сохраняются только номера мест, а информация о назначении этих мест должна храниться на центральном сервере управления. Робот может запросить центральный сервер управления о номере какого-нибудь места по его назначению с помощью функции скрипта GetPlaceNumberOf(type).

Например, робот может спросить сервер, к какому номеру места нужно подъехать для встречи гостей.

При вызове функции GetPlaceNumberOf на центральный сервер управления отправляется команда:

```
GET PLACE NUMBER OF <type>↵
```

Здесь <type> – параметр, переданный функции GetPlaceNumberOf. Разработчик серверного программного обеспечения и разработчик скрипта для робота должны самостоятельно договориться о типах мест.

При этом на сервер формируется следующий HTTP запрос (здесь type=«ADMIN»):

```
POST /robot.php HTTP/1.0
Host: 192.168.1.2
Content-Type: application/x-www-form-urlencoded
Content-Length: 48

robot=1&cmd=GET%20PLACE%20NUMBER%02OF%02ADMIN%0A
```

В ответ на запрос сервер должен передать номер запрашиваемого места:

```
HTTP/1.1 200 OK
Content-Length: 14
Content-Type: text/plain; charset=UTF-8

@PLACENUMBER 8
```

Функция GetPlaceNumberOf описана ниже

Команда перезапуска скрипта

Центральный сервер управления при необходимости может перезапустить скрипт на роботе, отправив на любой запрос робота ответ:

```
@RESTART <script>
```

Здесь: <script> – название скрипта.

Например, робот на сервер отправляет пустой запрос:

```
POST /robot.php HTTP/1.0
Host: 192.168.1.2
Content-Type: application/x-www-form-urlencoded
Content-Length: 12

robot=1&cmd=
```

В ответ на запрос сервер сообщает о необходимости перезапуска скрипта:

```
HTTP/1.1 200 OK
Content-Length: 18
Content-Type: text/plain; charset=UTF-8

@RESTART newtask.i
```

8.3.Функции языка для связи с центральным сервером управления

Функция SendToCommandServer

```
function SendToCommandServer(a1,…,an);
```

Функция отправляет команду (команды) на центральный HTTP-сервер управления.

Параметры сервера, а также имя документа, в который отправляется запрос, задаются в конфигурационном файле config.txt. Для отправки используется метод POST, а строки команд передаются как параметр cmd. Номер робота передается в параметре robot HTTP-запроса.

Команды могут собираться из подстрок $a_1,\ldots,a_n$. В конце каждой команды должен быть знак перевода строки «\n».

Пример:

```
SendToCommandServer("ROBOT AT PLACE ",table,"\n");
```

HTTP-запрос, сформированный данной командой, будет следующим:

```
POST /robot.php HTTP/1.0
Host: 192.168.1.2
Content-Type: application/x-www-form-urlencoded
Content-Length: 37

robot=1&cmd=ROBOT%20AT%20PLACE%205%0A
```

В ответ сервер должен выдать набор команд для робота, которые будут отправлены на вход фреймовой структуры обработки событий.

Например, если сервер формирует следующий ответ:

```
HTTP/1.1 200 OK
Content-Length: 37
Content-Type: text/plain; charset=UTF-8

* GO TO PLACE 5 FOR PUT DISH FROM 100
```

То на роботе запрос должен быть обработан с помощью следующего кода:

```
frameset("commands", 100)
{
     frame("* GO TO PLACE <n> FOR <reason> FROM <m>")
     {
          var n = TextOf("n");
          var reason = TextOf("reason");
          var from = TextOf("from");
          ...
     }
}
...
```

Функция GetRobotNumber

```
function GetRobotNumber();
```

Функция возвращает номер робота заданный, в конфигурационном файле параметром (COMMAND_SERVER_ROBOT). Используется, в основном, для обмена данными с центральным сервером управления.

Функция GetPlaceNumberOf

```
function GetPlaceNumberOf(placeName);
```

Функция запрашивает у центрального сервера управления номер специального места, заданного строкой placeNumber. Если место существует, функция возвращает положительное число с номером места. Если место не найдено, возвращает ноль. Возвращает null в случае отсутствия сервера управления.

Функция задерживает управление, пока не получит ответ сервера, однако процесс ожидания ответа могут прерывать события (фреймы) и таймеры.

Название мест для центрального сервера управления робототехническим комплексом «Робот-официант» приведен в таблице:

Строка placeNumber	***Назначение***
WASHING	Место мойки.
MENU	Место, куда следует отвозить меню (место администратора, с флагом «Меню»)
PAYMENT	Место, куда следует отвозить оплату (место администратора, с флагом «Оплата роботом)
MEETING	Место встречи гостей
CHARNING	Место для зарядки. Место на сервере должно быть обозначено атрибутом с номером робота. В противном случае робот едет на первую зарядку из списка.

Например:

```
var n = GetPlaceNumberOf("CHARGING");
if (n > 0)
{
    if (!GoToPlace(n)) print("Error");
}
```

9. Управление HTTP-запросами

Система iScript имеет встроенные функции управления запросами к внешнему WEB-серверу по протоколу HTTP или HTTPS, возвращающему информацию в формате JSON, XML или планарным текстом (в т.ч. html).

Все функции синхронны с таймаутом 2-3 секунды.

Ни одна из нижеприведенных функцией не поддерживает ответ сервера на перенаправление.

Функция HttpRequestPlain

Функция формирует запрос к внешнему WEB-серверу по протоколу HTTP/HTTPS или file, возвращающему ответ в виде планарного текста.

```
function HttpRequestPlain(URL, postData, cookie, headers);
```

или

```
function HttpRequestPlain(URL, postData, cookie);
```

или

```
function HttpRequestPlain(URL, postData);
```

или

```
function HttpRequestPlain(URL);
```

Здесь:

URL – URL-адрес сайта, включая указание протокола http, https или file.

postData – данные, передаваемые методом POST. Если данные не заданы или равны null, то формируется запрос методом GET.

cookie – строка COOKIE (в формате HTTP-запроса) или null.

headers – дополнительные заголовки HTTP-запроса. Заголовки должны представлять собой набор строк, разделенных символами “\r\n”. Строка заголовков должна также заканчиваться символами «\r\n».

Функция возвращает планарный текст документа, который вернул сервер, или null в случае ошибки. Ответ формируется только в том случае, если код HTTP-статуса равен 200 (OK).

Например:

```
html = HttpRequestPlain("http://www.dynsoft.ru");
if (html)
{
     var p = strpos(html, "<title>");
     if (p >= 0)
     {
          print("Title found!\n");
     }
}
```

Пример использования дополнительных заголовков:

```
html = HttpRequestPlain("http://www.dynsoft.ru",
               null,
               null,
               "Token: 54443\r\nAuthKey: 55555\r\n");
if (html)
{
     ...
}
```

Пример использования метода POST:

```
html = HttpRequestPlain("http://www.dynsoft.ru",
                "login=Dima&password=123");
if (html)
{
     ...
}
```

Функция возвращает текст в оригинальной кодировке, никаких преобразований кодировок не производится. Это же касается параметров, передаваемых в запросе.

Функция HttpRequestJSON

Функция формирует запрос к WEB-серверу по протоколу HTTP/HTTPS или file, возвращающему ответ в виде JSON.

```
function HttpRequestJSON(URL, postData, cookie, headers);
```

или

```
function HttpRequestJSON(URL, postData, cookie);
```

или

```
function HttpRequestJSON(URL, postData);
```

или

```
function HttpRequestJSON(URL);
```

Здесь:

URL – URL-адрес сайта, включая указание протокола http или https.

postData – данные, передаваемые методом POST. Если данные не заданы или равны null, то формируется запрос методом GET.

cookie – строка COOKIE (в формате HTTP-запроса) или null.

headers – дополнительные заголовки HTTP-запроса. Заголовки должны представлять собой набор строк, разделенных символами “\r\n”. Строка заголовков должна также заканчиваться символами «\r\n».

Функция возвращает объект, представляющий собой разобранный json-ответ сервера, или null в случае ошибки. Ответ формируется только в том случае, если код HTTP-статуса равен 200 (OK).

Запросы, формируемые данной функцией, аналогичны запросам функции HttpRequestPlain.

Например, если сервер возвращает следующий json-файл:

```
{
  "coords"   : { "x": 100, "y": 200.0 },
  "name"     : "My JSON Test",
  "objArray" :
        [
          {"x":300, "y":400 },
          {"x":500, "y":600 },
          {"x":700, "y":800 }
        ]
}
```

Запрос может быть разобран следующим кодом:

```
json = HttpRequestJSON("file:///C:/Projects/test.json");
if (json)
{
     print("json.coords.x=", json.coords.x, "\n");
     print("json.coords.y=", json.coords.y, "\n");
     print("json.name=", json.name, "\n");
     print("json.objArray[1].x=", json.objArray[1].x, "\n");
     print("json.objArray[1].y=", json.objArray[1].y, "\n");
}
```

Функция возвращает текст в оригинальной кодировке, никаких преобразований кодировок не производится. Это же касается параметров, передаваемых в запросе.

Функция HttpRequestXML

Функция формирует запрос к WEB-серверу по протоколу HTTP/HTTPS, возвращающему ответ в виде XML.

```
function HttpRequestXML(URL, postData, cookie, headers);
```

или

```
function HttpRequestXML(URL, postData, cookie);
```

или

```
function HttpRequestXML(URL, postData);
```

или

```
function HttpRequestXML(URL);
```

Здесь:

URL – URL-адрес сайта, включая указание протокола http или https.

postData – данные, передаваемые методом POST. Если данные не заданы или равны null, то формируется запрос методом GET.

cookie – строка COOKIE (в формате HTTP-запроса) или null.

headers – дополнительные заголовки HTTP-запроса. Заголовки должны представлять собой набор строк, разделенных символами “\r\n”. Строка заголовков должна также заканчиваться символами «\r\n».

Функция возвращает дерево из объектов, представляющий собой узлы XML-файла или null в случае ошибки. Ответ формируется только в том случае, если код HTTP-статуса равен 200 (OK).

Каждый узел XML-файла представляет собой объект, со следующими полями:

.name – (строка) название тэга узла.

.child – (объект) указатель на первый дочерний тэг данного узла.

.next – (объект) указатель на следующий тэг того же уровня.

.value – (строка) текст простого тэга (например: <config>My Test</config>)

Все атрибуты XML-тэга также формируются в виде полей объекта.

Например, пусть сервер возвращает следующих XML:

```
<?xml encoding="UTF-8"?>
<config>
     <coords x="100" y="200" />
     <name>My Test</name>
     <objects>
          <object x="200" y="500"/>
          <object x="210" y="560"/>
          <object x="220" y="590"/>
     </objects>
</config>
```

Запрос может быть разобран следующим кодом:

```
root = HttpRequestXML("file:///C:/Projects/test.xml");

xml = root;
while(xml)
{
  if (xml.name == "config")
  {
     xml2 = xml.child;
     while(xml2)
     {
          if (xml2.name == "coords")
          {
               print("X:", xml2.x, "\n");
               print("Y:", xml2.y, "\n");
          }
          else
          if (xml2.name == "name")
          {
               print("name:", xml2.value, "\n");
          }
          else
          if (xml2.name == "objects")
          {
             xml3 = xml2.child;
             while(xml3)
             {
               print("OBJ.X:", xml3.x, " OBJ.Y:", xml3.y, "\n");
               xml3 = xml3.next;
             }
          }
          xml2 = xml2.next;
     }
  }
  xml = xml.next;
}
```

Функция возвращает текст в оригинальной кодировке, никаких преобразований кодировок не производится. Это же касается параметров, передаваемых в запросе.

Функция GetLastCookie

Функция возвращает cookie из последнего HTTP-запроса.

```
function GetLastCookie();
```

Например:

```
html = HttpRequestPlain("http://ipusn.dynsoft.ru/forum.php");
if (html)
{
     // получить cookie, возвращенные последним HTTP-запросом
     var myCookie = GetLastCookie();
     ...
}
```

10. Функции измерения кодировки строк

Функция UTF8ToWin1251

```
function UTF8ToWin1251(stringUTF8);
```

Функция переводит строку из кодировки UTF-8 в кодировку Windows 1251 и возвращает строку с результатом.

Функция Win1251ToUTF8

```
function Win1251ToUTF8(string1251);
```

Функция переводит строку из кодировки Windows-1251 в кодировку UTF-8 и возвращает строку с результатом.

Функция Win1251ToKOI8

```
function Win1251ToKOI8();
```

Функция переводит строку из кодировки Windows-1251 в кодировку KOI8-R и возвращает строку с результатом.

Функция KOI8ToWin1251

```
function KOI8ToWin1251(stringKOI8R);
```

Функция переводит строку из кодировки KOI8-R в кодировку Windows-1251 и возвращает строку с результатом.

Функция URLEncode

```
function URLEncode(s);
```

Функция кодирует строку для передачи ее в качестве HTTP-запроса. Кодировка строки не изменяется.

11. Система распознавания лиц

11.1. Общие сведения о системе распознавания лиц

Программа dynrobot имеет возможность работы с системой распознавания лиц по средствам одной из библиотек: OpenCV , Neurotechnology VeriLook или VAT.

Выбор системы осуществляется в конфигурационном файле config.txt параметром FACE_TRACKING_DRIVER:

FACE_TRACKING_DRIVER = OPENCV – для распознавания с помощью OpenCV

FACE_TRACKING_DRIVER = VERILOOK – для распознавания с помощью Neurotecnology VeriLook.

Система выделения лиц на базе OpenCV не требует никаких дополнительных лицензий, но данная система не позволяет распознавать (идентифицировать) личность, не позволяет определять его возраст и пол.

При использовании Neurotecnology Verilook имеется возможность распознавать (идентифицировать) личность лица, определять пол и примерный возраст. Однако для использования Neurotecnology Verilook необходимо наличие лицензии от компании Neurotechnology на ее продукт VeriLook 5.7. Стоимость лицензии (на февраль 2016 года) составляет 330 евро. Приобрести лицензию можно на сайте www.neurotechnology.com. Имеется возможность использования trial-версии на 30 дней.

11.2. Настройка базы данных лиц

База данных лиц создается на основе файловой структуры.

Для создания базы данных необходимо:

1. На бортовой ЭВМ робота создать любую папку, в которую будет помещена база данных лиц.
2. В конфигурационном файле config.txt в параметре FACE_DB_DIRECTORY необходимо указать абсолютный или относительный путь к данной папке БЕЗ символа “\” или “/” в конце.
3. В данной папке необходимо создать для каждой личности отдельную папку. Название папки должно совпадать с именем или идентификатором личности.
4. В папку личности необходимо поместить одну или несколько фотографий данной личности в виде изображений в формате BMP, JPEG или PNG. Файлы должны быть четкими, разрешение не менее 640x480, расстояние между глаз на фотографии должно быть не менее 60 пикселей. На фотографии личность должна быть сфотографирована в фас.
5. Если при запуске dynrobot в папке личности нет файла person.templ, то программа по изображениям личности строит ее шаблон и сохраняет в файл person.templ. При наличии файла person.templ программа загружает шаблон из данного файла.
6. При добавлении новых фотографий личности или для пересоздания шаблона достаточно удалить файл person.templ и перезапустить программу dynrobot.

11.3. Функции управления системой распознавания лиц

Функция FaceTracker

Функция управляет системой трекинга лиц.

```
function FaceTracker( state, detectDetails);
```

или

```
function FaceTracker( state );
```

или

```
function FaceTracker();
```

Где:

state – (bool) состояние работы системы трекинга лиц. Если state = true, то система трекинга лиц включается, если false – отключается.

detectDetails – включить определение дополнительных деталей (пол, возраст), сильно замедляет работу. Работает только с Neurotechnology VeriLook.

Функция возвращает текущее состояние системы трекинга лиц.

Система автоматического трекинга лиц осуществляет периодические повороты головой в поиске человеческого лица. Если лицо попадает в камеру, то робот поворачивает в сторону человека голову. О найденном лице в систему скриптов отправляется событие «* faceIn», а при исчезновении «* faceOut».

Следует отметить, что для возникновения события «* faceIn» размер лица на изображении должен быть больше величины HELLO_FACE_WIDTH, указанного в конфигурационном файле.

Следует отметить, что для возникновения события «* faceOut», лицо должно пропасть на время, больше чем время, указанное в параметре FACE_OUT_PERIOD в конфигурационном файле.

Функция IsFace

```
function IsFace();
```

Функция возвращает признак наличие лица (bool), обнаруживаемого системой трекинга лиц.

Лицо может быть обнаружено на видео, но не попадать по размерам в систему событий «* faceIn» и «* faceOut». В отличие от системы событий, функция IsFace сообщает о наличии лица, детектируемого системой, независимо от его размеров.

Функция GetFaceAngle

```
function GetFaceAngle();
```

Функция возвращает угол (float, в градусах) пеленга лица. Положительное направление – в правую сторону. Если лицо по центру, угол до него 0º. Имеет смысл вызывать только, если функция IsFace вернула true.

Функция GetFaceWidth

```
function GetFaceWidth();
```

Функция возвращает ширину лица (float), обнаруживаемого системой трекинга лиц в процентах от ширины изображения. Имеет смысл вызывать только, если функция IsFace вернула true.

Функция GetFaceGender

```
function GetFaceGender();
```

Функция возвращает пол лица:

-1 – не определено.

0 – мужской.

1 – женский.

Функция работает только при распознавании лиц с помощью Neurotecnology VeriLook, и только если данная функция включена при вызове FaceTracker()

Имеет смысл вызывать только, если функция IsFace вернула true.

Функция GetFaceAge

```
function GetFaceAge();
```

Функция возвращает возраст лица.

-1 – не определено

В противном случае примерный возврат.

Функция работает только при распознавании лиц с помощью Neurotecnology VeriLook, и только если данная функция включена при вызове FaceTracker()

Имеет смысл вызывать только, если функция IsFace вернула true.

Функция FaceRecognizer

```
function FaceRecognizer(state);
```

или

```
function FaceRecognizer();
```

Функция включает или выключает систему распознавания (идентификации) лиц.

Если state = true, то система распознавания лиц включается, иначе выключатся. При запуске без параметров просто возвращает текущий статус системы распознавания лиц.

Функция возвращает true, если система распознавания лиц включена, иначе возвращает false.

Функция работает только при распознавании лиц с помощью Neurotecnology VeriLook.

Т.к. данная функция требует большое число системных ресурсов, не следует держать ее постоянно включенной.

Функция GetPerson

```
function GetPerson();
```

Функция возвращает строку с именем или идентификатором последнего распознанного лица или пустую строку, если личность не распознана или система идентификации лиц недоступна.

Идентификатор совпадает с названием папки личности (без полного пути).

Функция работает только при распознавании лиц с помощью Neurotecnology VeriLook.

Пример:

```
FaceRecognizer(true); // включить систему распознавания лиц

// главный цикл программы
while(1)
{
   sync();      // синхронизация

   // получить имя распознанного лица
   var person = GetPerson();
```

```
   if (person!="")
   {
       // поздороваться с конкретной личностью
       PlaySpeech("Здравствуйте "+person);
       Sleep(3000);
   }
}
```

Функция AddPerson

```
function AddPerson(personName, temp);
```

или

```
function AddPerson(personName);
```

Функция добавляет последнее выделенное лицо в базу данных лиц под именем personName.

Лицо может быть добавлено в базу данных временно (в этом случае temp = true), на время текущего сеанса работы программы, или постоянно (temp = false), в этом случае под личность создается соответствующая папка в директории базы данных. По умолчанию личность добавляется временно.

Функция возвращает true, если личность была успешно добавлена, иначе false.

Функция работает только при распознавании лиц с помощью Neurotecnology VeriLook. Без наличия данной системы функция возвращает null.

Пример:

```
frameset("Команды",1)
{
   // фрейм с нетиповизированным параметром "имя"
   frame("Меня зовут <имя>")
   frame("Мое имя <имя>")
   {
      if (IsFace())
      {
         AddPerson(TextOf("имя"));
         PlaySpeech("Очень приятно");
      }
      else
         PlaySpeech("Где вы, я вас не вижу");
   }
}

// главный цикл программы
while(1) sync();
```

Функция GetFaceRect

Функция возвращает координаты прямоугольной области лица на изображении.

```
function GetFaceRect();
```

Функция возвращает объект со следующими полями:

.centerX – координаты центра лица по оси X (в пикселях).

.centerY – координаты центра лица по оси Y (в пикселях).

.width – ширина лица в пикселях.
.height – высота лица в пикселях.
.imageWidth – полная ширина кадра видеоизображения, на котором производился поиск, в пикселях.
.imageHeigt – полная ширина кадра видеоизображения, на котором производился поиск, в пикселях.

Функцию имеет смысл вызывать, только, если функция IsFace() вернула true.

Пример:

```
FaceTracker(true);  // включить распознавание лиц

// организовать вечный цикл
while(1)
{
     sync();  // синхронизация

     if (IsFace())  // если есть лицо
     {
          // получить координаты лица
          rect = GetFaceRect();

          // вывести в консоль
          print("centerX=" + rect.centerX + "\n",
                "centerY=" + rect.centerY + "\n",
                "width=" + rect.width + "\n",
                "height=" + rect.height + "\n",
                "================================\n");
     }
}
```

Функция GetFaceStatistic

```
function GetFaceStatistic(period, usePerson);
```

или

```
function GetFaceStatistic(period);
```

или

```
function GetFaceStatistic();
```

Функция копит статистику по лицу в течение периода *period* и выдает усредненные данные по лицу.

Где:

period – период ожидания при накоплении (миллисекунды), по умолчанию 800 мс.

usePerson – (true/false) параметр, определяющий необходимость накопления статистики по распознанной персоне. По умолчанию false.

Функция возвращает объект со следующими полями:

.age – (int) средний возврат персоны в годах.

.gender – (int) пол персоны. 0 – муж. 1 – жен. -1 – не определено.

.person – (string) имя персоны или пустая строка. Формируется, только если установлен признак usePerson.

Функция не возвращается в течение всего времени накопления статистики, при этом события и таймеры могут выполняться, пока работает функция.

Работа функции возможна только при использовании системы распознавания лиц Neurotechnology VeriLook или VAT. При этом необходимо включить детектирование пола

и возраста в функции FaceTracker, а также FaceRecognizer (при использовании параметра usePerson).

Пример:

```
frameset("commands",1)
{
   // событие появления лица
   frame("* faceIn")
   {
      // в течение 800 мс накапливать информацию по лицу
      var face = GetFaceStatistic(800, true);
      if (face.person!="")
      {
          // известная роботу персона. Выбрать вариант
          // приветствия в зависимости от возраста
          if (face.age < 18)
               PlaySpeech("Здравствуй "+face.person);
          else
               PlaySpeech("Здравствуйте "+face.person);
     }
     else
     if (face.gender == 0)
     {
          // мужчина
          if (face.age < 18)
              PlaySpeech("Привет, мальчик");
          else
          if (face.age < 45)
              PlaySpeech("Здравствуйте, молодой человек");
          else
              PlaySpeech("Здравствуйте, мужщина");
     }
     else
     if (face.gender == 1)
     {
          // женщина
          if (face.age < 18)
               PlaySpeech("Здравствуй, девочка");
          else
          if (face.age < 35)
               PlaySpeech("Здравствуйте, девушка");
          else
               PlaySpeech("Приветствую, Мадам");
     }
     else
          // пол не определен
          PlaySpeech("Здравствуйте");
   }
}
```

Функция ClearFaceIn

```
function ClearFaceIn();
```

Функция сбрасывает все внутренние триггеры, блокирующие возникновению события «* faceIn».
Так, например, если не истек таймаут исчезновением лица в кадре, повторное появление лица не формирует событие «* faceIn». Однако событие будет вновь формироваться при появлении лица, если предварительно вызвать функцию ClearFaceIn.
При использовании системы распознавания лиц робот не формирует событие «* faceIn», если в течение 1-2 минут появлялось лицо того же пола и возраста или та же персона. Однако вызов функции ClearFaceIn разрешает формирование события вновь.

12. Функции работы с базой хранения параметров

Для хранения значений некоторых параметров между сеансами запуска программы «ДинРобот» в языке iScript имеется набор функций работы с базой данных хранения внутренних параметров. Все параметры из этой базы данных загружаются в оперативную память и по мере их модификаций асинхронно записываются в файл. Физическая запись в файл осуществляется, спустя 4 секунды после внесения в базу данных последнего изменения, или после штатного завершения программы «ДинРобот».

Файл базы данных для хранения этих параметров указывается в конфигурационном файле «config.txt» параметром «INNER_DB_FILE». По умолчанию это файл "InnerDB.txt".

Данный файл текстовый, параметры в нем храняться в формате:

<параметр1> = <значение1>

<параметр2> = <значение2>

…

<параметр$_n$> = <значение$_n$>

При необходимости файл может быть вручную отредактирован при выключенной программе «ДинРобот» (или при включенной, но не производящей никаких модификаций этого файла).

Ниже представлен набор функций работы с этими параметрами. Все функции чувствительны к регистру (большой и малой букве) названия параметра.

Функция GetInnerDB

```
function GetInnerDB(param);
```

или

```
function GetInnerDB(param, def);
```

Получает значение из базы данных хранения внутренних параметров в **строковом виде**.

Здесь:

param – (string) название параметра.

def – (string) значение по умолчанию. Если данное значение не указано, то в качестве значения по умолчанию используется пустая строка.

Если указанный параметр указан функция возвращает строку со значением этого параметра. Если параметр не найден, то возвращается значение по умолчанию.

Функция GetInnerDBInt

```
function GetInnerDBInt(param);
```

или

```
function GetInnerDBInt(param, def);
```

Получает значение из базы данных хранения внутренних параметров в **целочисленном виде (параметр типа int)**.

Здесь:

param – (string) название параметра.

def – (int) значение по умолчанию. Если данное значение не указано, то в качестве значения по умолчанию используется 0.

Если указанный параметр указан функция возвращает число со значением этого параметра. Если параметр не найден, то возвращается значение по умолчанию.

Функция GetInnerDBFloat

```
function GetInnerDBFloat(param);
```

или

```
function GetInnerDBFloat(param, def);
```

Получает значение из базы данных хранения внутренних параметров в **виде значения с плавающей точкой (параметр типа double)**.

Здесь:

param – (string) название параметра.

def – (double) значение по умолчанию. Если данное значение не указано, то в качестве значения по умолчанию используется 0.0.

Если указанный параметр указан функция возвращает число со значением этого параметра. Если параметр не найден, то возвращается значение по умолчанию.

Функция GetInnerDBBool

```
function GetInnerDBBool(param);
```

или

```
function GetInnerDBBool(param, def);
```

Получает значение из базы данных хранения внутренних параметров в **виде значения типа bool**.

Здесь:

param – (string) название параметра.

def – (bool) значение по умолчанию. Если данное значение не указано, то в качестве значения по умолчанию используется false.

Если указанный параметр указан функция возвращает число со значением этого параметра. Если параметр не найден, то возвращается значение по умолчанию.

Функция SetInnerDB

```
function SetInnerDB(param, value);
```

Записывает в базу хранения внутренних параметров параметр с указанным значением. Если параметр существует, то функция заменяет его значение, если параметр не существует, то функция добавляет в базе данных новый параметр. После этих изменений файл базы данных записывается на диск спустя 4 секунды после последних изменений.

Здесь:

param – (string) название параметра.

value – значение параметра.

Тип параметра определяется типом данных параметра value.

Функция возвращает true, если запись прошла удачно. Значение false бывает только, если недостаточно памяти.

Функция DeleteInnerDB

```
function DeleteInnerDB(param);
```

Удаляет из базы хранения внутренних параметров указанный параметр. После этих изменений файл базы данных записывается на диск спустя 4 секунды после последних изменений.

Здесь:

param – (string) название параметра.

Функция возвращает true, если параметр существовал и был удален.

Функция DeleteAllInnerDB

```
function DeleteInnerDB(startWith);
```

Удаляет из базы хранения внутренних параметров все параметры, названия которых начинается с startWith. Если startWith равно пустой строке, то удаляются все параметры из базы данных.

После этих изменений файл базы данных записывается на диск спустя 4 секунды после последних изменений.

Здесь:

startWith – (string) название параметра.

Функция возвращает количество удаленных параметров.

Пример:

```
// записать базу данных параметры
t0 = ...;
t1 = ...;
t2 = ...;
SetInnerDB("personTime0", t0);
SetInnerDB("personTime1", t1);
SetInnerDB("personTime2", t2);
...

// удалить из базы данных все параметры, название которых
// начинается с personTime
DeleteAllInnerDB("personTime");
```

13. Функции работы с MySQL

Параметры доступа к базе данных MySQL задаются из конфигурационного файла. Для простоты использования дескриптор базы данных и дескриптор результатов запроса хранятся внутри системы и безопасным образом открываются, закрываются и уничтожаются при завершении работы скриптов.

При работе с базой данных MySQL следует использовать кодировку Windows-1251.

Функция mysql_connect

```
function mysql_connect();
```

Функция подключается к базе данных MySQL. Параметры доступа к базе данных задаются в конфигурационном файле системы.

Функция возвращает true в случае удачного подключения и false в случае ошибки.

Функция mysql_close

```
function mysql_close();
```

Функция завершает сеанс работы с базой данных MySQL. Очищает результаты запроса, если они не были очищены функцией **mysql_free_result**.

Функция mysql_query

```
function mysql_query( SQL );
```

Функция выполняет SQL-запрос в базу данных.

Здесь: SQL – строка с SQL-запросом.

Функция возвращает true в случае успеха и false в случае ошибки.

Функция mysql_fetch_row

```
function mysql_fetch_row();
```

Функция возвращает следующую строку результата SQL-запроса в виде массива или null в случае ошибки.

При первом вызове данной функции после функции **mysql_query** данная функция загружает результаты выполнения запроса в память. Для освобождения памяти от этих результатов следует вызывать функцию **mysql_free_result**.

Например:

```
// подключиться к MySQL
if (mysql_connect())
{
     // запрос данных из mysql
     if (mysql_query("SELECT `id`,`name` FROM `robots`"))
     {
          // разбор результатов запроса
          while(row = mysql_fetch_row())
          {
               // вывести в консоль
               print(row[0] + ") " + row[1]+"\n");
          }
          // освободить результаты
          mysql_free_result();
     }
     // отключиться от MySQL
     mysql_close();
}
```

Функция mysql_fetch_assoc

```
function mysql_fetch_assoc();
```

Функция возвращает следующую строку результата SQL-запроса в виде объекта, название полей которого совпадает с названиями колонок в запрошенной таблице или null в случае ошибки.

При первом вызове данной функции после функции mysql_query данная функция загружает результаты выполнения запроса в память. Для освобождения памяти от этих результатов следует вызывать функцию mysql_free_result.

Например:

```
// подключиться к MySQL
if (mysql_connect())
{
     // запрос данных из mysql
     if (mysql_query("SELECT `id`,`name` FROM `robots`"))
     {
          // разбор результатов запроса
          while(row = mysql_fetch_assoc())
          {
               // вывести в консоль
               print(row.id + ") " + row.name +"\n");
          }
          // освободить результаты
          mysql_free_result();
     }
     // отключиться от MySQL
     mysql_close();
}
```

При формировании SQL запросов не создавайте таких названий колонки, которые невозможно преобразовать в название поля объекта. Используйте оператор SQL «as», чтобы переименовать колонки в SQL-запросе.

Например, вместо SQL-запроса:

```
SELECT
    orders.`id`,
    items.`name`
FROM
  orders, items
WHERE ...
```

Следует использовать:

```
SELECT
     orders.`id` as `id,
     items.`name` as `name`
FROM
   orders, items
WHERE ...
```

Функция mysql_free_result

```
function mysql_free_result();
```

Функция очищает результаты SQL-запроса, размещенные в памяти функциями **mysql_fetch_row** или **mysql_fetch_assoc**.

Функция mysql_escape_string

Функция имеет синоним mysql_escape.

```
function mysql_escape_string(string);
```

Функция экранирует специальные символы строки string, подготавливая строку для записи в базу данных.

Пример:

```
// строка имени пользователя, составленная хакером
var myName = "Дима'; DROP TABLE users; #";

...
mysql_query("SELECT id FROM users WHERE name='" +
                                    mysql_escape_string(myName)
+"'");
...
```

Обратите внимание, строка *myName* была экранирована. Поэтому, когда ее текст попадет в функции mysql_query, SQL-запрос будет следующий:

```
SELECT
  id
FROM
  users
WHERE name='Дима\' DROP TABLE users; #';
```

При этом, если бы строка *myName* не была экранирована, то запрос удалил бы таблицу users.

Функция mysql_insert_id

Функция возвращает идентификатор ключа последней операции INSERT.

```
function mysql_insert_id();
```

Пример:

```
...
if (mysql_query("INSERT INTO users (name) VALUES 'Дима'"))
{
     var userID = mysql_insert_id();
}
```

Функция mysql_affected_rows

Функция возвращает количество строк, затронутых операциями UPDATE и DELETE.

```
function mysql_affected_rows();
```

Пример:

```
...
if (mysql_query("UPDATE users SET `pass`='123' WHERE id=2"))
{
   // если ни одной строки запрос не затронул, то вывести ошибку
   if (mysql_affected_rows() == 0)
        print("Пользователь не найден");
}
```