Евстигнеев Дмитрий Валерьевич

Язык бортовых скриптов IScript3

Евстигнеев Д.В.

29.11.2025 г.

Оглавление:

1. Введение в язык 9
1.1. Глобальные и локальные переменные. Функции 9
1.2. Операторы языка 10
1.3. Операторы JSON 12
2. Типы данных 13
2.1. Используемые типы данных 13
3. Стандартные функции языка 14
3.1. Функции преобразования типов данных 14
Функция toBool 14
Функция toInt 14
Функция toInt64 14
Функция toFloat 14
Функция toString 14
Функции typeof 14
Оператор copy или Copy 15
3.2. Функции управления объектами 15
Функция object 15
3.3. Функции управления массивами 16
Функция array 16
Функция count 16
Функция array_push 17
Функция array_push_if_not_exists 17
Функция array_pop 17
Функция array_shift 17
Функция array_unshift 18
Функция array_splice 18
Функция array_slice 19
Функция array_indexOf или array_index_of 19
Функция array_compare 19
Функция array_remove 19
Свойство .length и .size для массивов 19
3.4. Функции управления бинарными данными 20
Функция binary 20
Функция set_bin_size 20
Функция get_bin_size 20
Функция get_bin_byte 20
Функция set_bin_byte 20
Функция get_bin_data 21
Функция set_bin_data 22
Функция bin_subrange 23
Функция bin_to_array 23
Свойство .length и .size для бинарных данных 23
3.5. Функции работы со строками 23
Функция strlen 23
Функция strpos 24
Функция WordCase 24
Функция split 25
Функция OneOf 25
Функция join 25
Функция getCharAt 25

Функция getCodeAt....26
Функция substring....26
Функция trim....26
Функция trimleft....26
Функция trimright....26
Функция startwith....27
Функция endwith....27
Функция casestartwith....27
Функция caseendwith....27
Функция match....27
Функция replace....27
Функция indexOf или IndexOf....28
Функция caseIndexOf....28
Функция UUID....28
Функция upcase/uppercase....29
Функция lowercase....29
Функция UTF8ToANSI или UTF8ToWin1251....29
Функция ANSIToUTF8 или Win1251ToUTF8....29
Функция format....29
Функция HexToInt....30
Функция URLEncode/EncodeURL....30
Функция Declination....30
Функция SetCodePage....31
Функция SetDeclinatorLang....32
Функция NumberToDesc....32
Функция DateToDesc....34
Функция GetWeekDay....35
Функция GetMonthName....35
Функция ValuteToDesc....36
Свойство .length для строк....36
3.6. Регулярные выражения....37
Функция preg_match....39
Функция preg_matchAll....39
Функция preg_replace....40
3.7. Математические функции....41
Основные математические функции....41
Функция rand....41
3.8. Объявленные константы....41
4. Элементы векторной алгебры....42
4.1. Встроенные математические операторы векторной алгебры....42
4.2. Функции векторной алгебры....46
Функция VecLength....46
Функция Normalize....46
Функция Distance....46
Функция Matrix или TranslateMatrix....46
Функция RotXMatrix....47
Функция RotYMatrix....47
Функция RotZMatrix....47
Функция ScaleMatrix....48
Функция PlaneMatrix....48
Функция WorldPointToMatrix....49
Функция MatrixToWorldPoint....49

Функция MatrixRow....49
5. Фреймы. Распознавание речи и событий....51
5.1. Введение....51
5.2. Фрейм....51
5.3. Синонимы в словесной предпосылке....51
5.4. Слова с произвольным окончанием во фрейме....52
5.5. Необязательные слова....53
5.6. Синонимы с необязательным словом....53
5.7. Порядок слов в словесной предпосылке....54
5.8. Приоритет активизации фрейма....55
5.9. Приоритет фреймов во временных фреймсетах....55
5.10. Жесткое начало или конец предожения....55
5.11. Автозамены перед сравнением....56
5.12. Параметры в словесной предпосылке....57
Числовые параметры в словесной предпосылке....58
5.13. Фреймсет....59
5.14. Текст и значение параметра....61
Текст параметра. Функция TextOf....61
Значения параметра. Функция ValueOf....62
5.15. Запрос из программного кода....63
Запрос с ожиданием результата. Функция LangQuery....63
Асинхронный запрос. Функция LangQueryAsync....63
5.16. Сложносоставные предложения....64
5.17. Создание фреймовой структуры в коде программы....65
Функция CreateFrameset....66
Функция CreateFrame....66
Функция DeleteFrameset....67
Функция NotFoundText....67
5.18. Дерево ведения диалога. Временный фреймсет....67
Функция ClearTasks....68
Функция SetTaskPrior....68
5.19. Функции управления системой распознавания речи....69
Функция SpeechRecognizer/SpeechRecognition....69
Функция SetSpeechDriver....69
5.20. Специальные события системы распознавания речи....70
Получение строки нераспознанного запроса....70
Получение события начала распознавания....70
6. Стандартные события....71
7. Функции работы с JSON и XML....73
Функция json_decode. Разбор строки в формате JSON....73
Функция json_encode. Перевод объектов в строку JSON....73
Функция xml_decode. Разбора строки в формате XML....74
Функция xml_encode. Формирования строки в формате xml....75
8. Управление синтезатором речью и звуком....77
Функция PlaySpeech/Say/say....77
Функция GetLastSpeech....78
Функция RobotSilenceTime....78
Функция HumanSilenceTime....78
Функция RobotGender....79
9. Управление системой распознавания лиц....80
9.1. События системы распознавания лиц....80

9.2. Функции системы распознавания лиц 80
Функция FaceRecognizer 80
Функция GetFaceCount 81
Функция IsFace 81
Функция GetFaceRects 81
Функция GetFaceRect 81
Функция GetFacePerson 82
Функция SetSearchPerson или SetSearchPersonId 82
Функция GetFaceAge 82
Функция GetFaceGender 83
Функция GetFaceEmotion 83
Функция GetFacePassedTime 83
Функция PersonName 83
Функция PersonRights 83
Функция PersonInfo 84
Функция GetPersonList 84
Функция AddNewPerson 84
Функция LockAcquaintance 85
10. Функции работы с QR-кодами 87
Функция DetectQRCodes 87
11. Экранные сервисы 89
Функция SnapPhoto (сделать фото) 89
Функция RoboArt (робо-художник) 90
Функция GetRoboArtImageWidth 92
Функция GetRoboArtImageHeight 92
Функция HasMotion (детектор движений) 92
Функция HasContrast (детектор объектов или контраста) 92
Функция GetAvgBrightness (детектор освещенности) 93
Функция SetMotionEvent (задать событие на движения в кадре) 93
Функция SetMinBrightnessEvent 94
Функция SetMaxBrightnessEvent 94
Функция GetVideoAnalyzerGrid 94
12. Функции работы с датой и временем 96
Функция time 96
Функция localtime 96
Функция mktime 96
Функция GetTickCount 97
Функция sleep 97
13. Временные переменные 99
Функция SetTempVar 99
Функция GetTempVar 99
14. Функции ввода/вывода 100
Функция include 100
Функция print 100
Функция show 100
15. Управление скриптами и системные функции 102
Функция sync 102
Функция exit 102
Функция SetTimer 102
Функция KillTimer/ClearTimer 103
Функция Restart 103

16.Функции работы с файлами..... 104
Функция fopen..... 104
Функция fwrite или fputs..... 104
Функция fgets..... 105
Функция fread..... 105
Функция fseek..... 106
Функция feof..... 106
Функция fclose..... 106
Функция scandir..... 107
17.Функции работы COM-портами..... 108
Функция CommOpen..... 108
Функция CommClose..... 108
Функция IsCommOpen..... 109
Функция CommWrite..... 109
Функция CommWriteBE..... 111
Функция CommRead..... 112
Функция CommBeginRead..... 116
Функция CommEndRead..... 116
18.Функции работы с внутренней базой данных параметров..... 117
Функция SetInnerDB..... 117
Функция GetInnerDB..... 117
Функция GetInnerDBInt..... 117
Функция GetInnerDBInt64..... 118
Функция GetInnerDBBool..... 118
Функция GetInnerDBFloat..... 118
Функция DeleteInnerDB..... 118
Функция DeleteAllInnerDB..... 119
19.Функции работы с базой данных объектов..... 120
Функция CreateSubjDB или CreateSubjectDB..... 120
Функция SetSubjDB..... 121
Функция GetSubjDB..... 122
Функция DelSubjDB..... 122
Функция UpdateSubjDBFrameset..... 122
Функция SetSubjDBPath..... 123
Функция CreatePersonFrameset..... 123
Функция CreateSpecialPersonFrameset..... 124
20.Функции работы с экранным контентом и мультимедиа..... 126
Функция SetBrowserURL..... 126
Функция PlayWave..... 126
Функция AudioVolume..... 126
Функция VideoRecorder..... 127
21.Работа с внешними интернет-соединениями..... 128
21.1. Отправка почты (работа с SMTP)..... 128
Функция SendMailTo..... 128
21.2. Управление HTTP-запросами..... 128
Функция HttpRequestPlain..... 128
Функция HttpRequestJSON..... 130
Функция HttpRequestXML..... 131
Функция GetLastCookie..... 133
Функция HTTPEncoding..... 134
Функция GetIPAddressList..... 134

Функция IsInternet....134
Функция HasUpdates....134
Функция InstallUpdates....135
22.Управление роботом....136
22.1. Управление шасси....136
Функция SetWheelSpeed или SetWheelsSpeed....136
Функция GetZoneDanger....136
22.2. Управление интеллектуальным движением и картой....137
Функция Go....137
Функция GoToPlace....138
Функция WheelMovingBy....138
Функция WheelMovingByAndWait....140
Функция KickNavigation....140
Функция GetCurrentZone....141
Функция GetCurrentPlace....141
Функция SetCurrentPlace....142
Функция IsPlaceExists....142
Функция GetPlaceDesc....142
Функция GetZoneDesc....142
Функция GetPlaceCoord....143
Функция GetRobotCoord....143
Функция SetRobotCoord....143
Функция GetPlaceList....144
Функция GetZoneList....144
Функция GetZoneType....144
Функция IsNarrowZone....144
Функция GetLighttraffic....144
Функция Park....145
Функция FollowFace....145
Функция GetEmg....146
Функция SetLighttraffics....146
22.3. Управление головой и подъемником....147
Функция Head....147
Функция HeadRel....148
Функция GetHeadAngle....148
Функция GetHeadPitch....148
Функция GetHeadRoll....149
Функция Lift....149
Функция SetLiftSpeed....149
Функция LockFaceTracking или LockFaceTracker....149
22.4. Управление отдельными моторами....150
Функция SetMotorPos или SetMotorPosition....150
Функция GetMotorPos или GetMotorPosition....150
Функция SetMotorSpeed....150
Функция SetMotorPWM....151
Функция BeginGroupCtrl....151
Функция EndGroupCtrl....152
Функция CalibMotor....153
Функция IsMotorMoving....153
22.5. Управление системой комманд и входных событий....154
Функция SendCommand....154
Функция GetEvents....154

22.6. Управление системой питания ... 155
Функция GetBattery ... 155
Функция MinPowerLevel ... 155
Функция IsChargeConnected ... 155
22.7. Управление сенсорами ... 155
Функция GetSensor ... 156
Функция SetSensorTrigger или CreateSensorTrigger ... 156
Функция DeleteSensorTrigger, ClearSensorTrigger или RemoveSensorTriggger ... 158
22.8. Управление дальномерами ... 158
Функция GetRangefinder ... 158
Функция GetSideRFDistance ... 158
Функция GetStairDetectorStopSignal ... 159
22.9. Управление манипуляторами и жестами ... 159
Функция ArmGo ... 159
Функция ArmGoL ... 161
Функция Gesture ... 162
Функция GestureJ ... 163
Функция GestureL ... 164
Функция RunGesture ... 165
Функция RunGestureJ ... 166
Функция RunGestureL ... 167
Функция GetGestureCoord ... 168
Функция AddGesture ... 168
Функция UpdateGesture ... 169
Функция DeleteGesture ... 169
Функция GetGestureList ... 169
Функция RenameGesture ... 170
Функция SetJointSpeed ... 170
Функция SetJointAcc ... 170
Функция SetLinearSpeed ... 171
Функция SetLinearAcc ... 171
Функция IKAngularCritery ... 171
Функция IKMinCritery и MinIKCritery ... 172
Функция IKMaxCritery и MaxIKCritery ... 173
Функция GetArmPose и ArmGetPose ... 173
Функция ArmKinematic ... 173
Функция ArmIK ... 174
22.10. Управление камерами глубины ... 174
Функция PickDepth ... 174
22.11. Управление подсветкой ... 175
Функция SetLED ... 175

1. Введение в язык

Язык IScript3 основан на синтаксисе языка JavaScript. Однако, функции, объекты и методы, используемые в IScript3, отличаются от функций объектов и методов, используемых в JavaScript.

Программа на языке IScript3 для робота должна быть помещена в папку «iscript». Обычно файл имеет расширение «.i3», Например «main.i3». Могут, как наследие предыдущих версий встречаться расширения «.i».

Файлы чат-бота для IScript3 имеют расширения «.chatbot». Например, «masha2.chatbot». С точки зрения самого языка, эти файлы ни чем не отличается от файла с раширением «.i3», однако для таких файлов предусмотрен специальный визуальный редактор, который требует соблюдения определенной структуры файла чат-бота.

Имя главного файла программы определяется в конфигурационном файле робота «config.txt» в параметре FILE секции [SCRIPTS].

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)
```

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 goto:

- 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.
- switch(*условие*)

```
{
    case вариант1:
            действия1;
            break;
    case вариант2:
            действия2;
            break;
    …
    default:
            действия_по_умолчанию;
            break;
}
```

В отличие от C/C++ оператор switch/case работает с любым типом данных, включая строковые, а в «case» можно использовать не только константы, но и выражения.

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);
```

Пример использования оператора switch/case:

```
a = 17;
switch(a)
{
     case 1: print("Один\n"); break;
     case 2: print("Два\n"); break;
     case 3: print("Три\n"); break;
     case 4:
     case 5:
     case 6:
          print("Много\n");
          break;
     default:
          print("По умолчанию много\n");
          break;
     case 7:
          print("Семь\n");
          break;
}

print("Ну как-то так...\n");
```

1.3. Операторы JSON

В язык IScript3 встроены константные выражения на JSON.

```
a={
    id:1312,
    name:"Иванов",
    items:[1,10,21,18],
    place:{x:100, y:200}
  };
```

Выражение на JSON формирует в IScript3 объекты и массивы напрямую, минуя операции создания объектов и массивов.

Однако выражение на JSON должно содержать только константные данные. Т.е. нельзя внутрь JSON вставлять переменные или математические выражения.

Т.е. запрещено делать следующие конструкции:

```
a={id:43+321}; // ОШИБКА! Математическое выражение в JSON!
```

```
a={id:myVar};  // ОШИБКА! Переменная в JSON
```

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
Int64	Целое 64-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
Resource	Указатель на системный ресурс

Типы данных указываются неявно. Они формируются автоматически.

Типы данных Object, Array и Binary являются ссылочными, остальные – простыми.

При операциях присвоении простых типов данных само значение переменных копируется. В результате образуется независимая копия исходных данных.

При операциях присвоения ссылочных типов данных копируется лишь ссылка на данные. В результате образуется зависимая копия данных. Например, если переменная *a* содержала массив. Ее приравняли к переменной *b*. В массиве *b* изменили элемент массива. Этот же элемент массива изменился в переменной *a*.

Например:

```
a = 5; // Int 5
b = a; // простой тип данных, содержимое копируется
a = 10; // a изменили, а b осталось
print("a=", a, " b=",b, "\n"); // a=10 b=5


a = array(10,20,30); // Array с содержимым [10,20,30]
b = a; // ссылочный тип данных,
       // в b скопирована ссылка на массив, а не сам массив
a[1]=50; // заменили 1-й элемент массива
         // он заменился, как в a так и в b.
show(a); // [10,50,30]
show(b); // [10,50,30]
```

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 toInt64

```
function toInt64(a)
```

Преобразует входные данные *a* в тип данных int64.

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
int64	Целое 64-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.
resource	Указатель на системый ресурс

Оператор copy или Copy

Оператор (функция) создает независимую копию значения исходных данных, включая ссылочные типы данных, включая все вложнные массив, объекты и бинарные данные.

```
function copy(src);
```

или

```
function Copy(src);
```

Где:

src – исходные данные.

Возвращает данные того же типа, что и исходные данные.

Например:

```
a = array(10,20,30);
b = a;   // обычное присвоение ссылочного типа.
         // в результате образуется зависимая копия
         // массива
a[1] = 44; // изменяем в массиве a, меняется и в b.
show(a); // [10,44,30]
show(b); // [10,44,30]


b = copy(a); // создаем независимую копию массива a
a[1] = 555; // изменяем в массиве a, в b не меняется
show(a); // [10,555,30]
show(b); // [10,44,30]
```

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_indexOf или array_index_of

Функции array_indexOf и array_index_of являются синонимами.

```
function array_indexOf(arr, value);
```

Функция возвращает индекс вхождения в массив значения value, или (-1), если не найдено.

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]
```

Свойство .length и .size для массивов

Свойство «.length» и ее синоним «.size» для массива возвращает его длину, аналогично функции count().

Например:

```
myArray = array(10,20,30);   // создать массив 1
len = myArray.length;        // получить длину массива
print("Длина массива ",len,"\n");
```

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(bin, size);
```

Устанавливает размер бинарных данных bin. Старые данные в блоке сохраняются.

Функция get_bin_size

Функция является синонимом функции count().

```
function get_bin_size(bin);
```

Функция возвращает размер бинарных данных (если bin 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(bin, ofs);
```

Возвращает значение байта из бинарного блока *bin* 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(bin, ofs, value);
```

Устанавливает значение байта в бинарном блоке *bin* 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 get_bin_data

Возвращает значение слова, сложенного из нескольких последовательных байт в бинарных данных. Тип возвращаемых данных зависит от параметра type.

```
function get_bin_data(bin, ofs, type);
```

Где:

bin – (binary) блок бинарных данных.

ofs – (int) смещение первого байта данных.

type –(string) строка с типом данных (см. таблицу ниже).

Функция возвращает тип данных, в зависимости от параметра type, указанного в таблице ниже. Если ofs + размер_считываемых_данных больше размера бинарных данных, то функция возвращает null.

type (значение строки), регистровонезависимо	*Описание*	*Возвращаемый тип данных*
bool	Булевский тип данных, 1 байт	bool
char	Символ строки, 1 байт	string
wchar	Символ строки в Unicode, конвертируется в ANSI, 2 байта	string
int8	целое 8-битное со знаком, 1 байт	int
uint8	целое 8-битное без знака, 1 байт	int
int16 или short	целое 16-битное со знаком, Little Endian, 2 байта	int
uint16 или word	целое 16-битное без знака, Little Endian, 2 байта	int
int16be	целое 16-битное со знаком, Big Endian, 2 байта	int
uint16be	целое 16-битное без знака, Big Endian, 2 байта	int
int, int32 или long	целое 32-битное со знаком, Little Endian, 4 байта	int
uint32	целое 32-битное без знака, Little Endian, 4 байта	int64
int32be	целое 32-битное со знаком, Big Endian, 4 байта	int
uint32be	целое 32-битное без знака, Big Endian, 4 байта	int64
int64 или uint64	целое 64-битное со знаком, Little Endian, 8 байта	int64
int64be или uint64be	целое 64-битное со знаком, Big Endian, 8 байта	int64
float, float32	С плавающей запятой в формате IEEE, 32-битное, Little Endian, 4 байта	float

double, float64	С плавающей запятой в формате IEEE, 64-битное, Little Endian, 8 байта	float
floatbe, float32be	С плавающей запятой в формате IEEE, 32-битное, Big Endian, 4 байта	float
doublebe, float64be	С плавающей запятой в формате IEEE, 64-битное, Big Endian, 8 байта	float
string*N* Имеется в виду: string2, string8,…	Строка в 8-битной кодировке из N символов, считывается N байт	string
wstringN Имеется в виду: wstring2, wstring8,…	Строка в Unicode из N символов. После считывания конвертируется в ANSI. Считывается N*2 байт	string
HEX2	Два байта, интерпретируемые как строка в формате HEX. Значение переводится в int без знака	int
HEX4	4 байта, интерпретируемые как строка в формате HEX. Значение переводится в int без знака	int
HEX6	6 байт, интерпретируемые как строка в формате HEX. Значение переводится в int без знака	int
HEX8	8 байт, интерпретируемые как строка в формате HEX. Значение переводится в int со знаком	int

Например:

```
bin = ...  // формировать бинарные данные

// считать два байта со смещения 50 и интепретировать их
// как int16
var a = get_bin_data(bin, 50, "int16");

// считать строку из 12 байт со смещения 52
var s = get_bin_data(bin, 52, "string12");
```

Функция set_bin_data

Записывает в бинарный блок данных значение, состоящее из нескольких последовательных байт.

```
function set_bin_data(bin, ofs, data, type);
```

или

```
function set_bin_data(bin, ofs, data);
```

Где:
bin – (binary) блок бинарных данных.
ofs – (int) смещение первого байта данных.
data – записываемые данные, тип зависит от параметра type, если он указан.

type –(string) строка с типом данных (см. таблицу функции get_bin_data).

Функция записывает данные в бинарный блок, начиная со смещения ofs. Если данные выходят за пределы блока данных, то размер бинарных данных расширяется.

Функция возвращает смещение следующих данных, записываемых в бинарный блок, или null, если не удается переразместить бинарный блок данных памяти.

Если не указан тип данных, то функция записывает в память типы данных bool (1 байт), int (4 байта), int64 (8 байт), float (8 байт), string (количество байт определяется по длине строки).

Функция bin_subrange

```
function bin_subrange(bin, ofs)
```

или

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

Функция возвращает новый блок бинарных данных, полученный путем копирования *len* байт из исходного бинарного блока *bin*, 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(bin);
```

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

Свойство .length и .size для бинарных данных

Свойство «.length» и ее синоним «.size» для бинарных данных возвращает размер бинарных данных в байтах, аналогично функции get_bin_size().

Например:

```
bin = binary(100);   // создать массив 1
len = bin.length;        // получить размер
print("Размер данных ",len," байт\n");
```

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 или float) 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(n, "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 OneOf

Функция возвращает одно из значений, выбранных случайным образом.

```
function OneOf(v1, v2, v3, ...);
```

или

```
function OneOf(arr);
```

Где:

v1, v2, v3, … – значения произвольного типа.

arr – массив значений произвольного типа.

В случае передачи отдельных значений функция выбирает случайным образом одно из них и возвращает в качестве результата.

В случае передачи функции массива в качестве единственного аргумента, функция возвращает значение одного из элементов массива, выбранного случайным образом.

Тип результата зависит от типа аргумента.

Пример:

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

s = OneOf("Красный", "Зеленый", "Синий"); // s = "Зеленый";
```

В данном примере значение «Зеленый» было выбрано случайным образом.

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 startwith

```
function startwith(string, subStr)
```

Функция возвращает true, если строка string начинается со строки subStr. Иначе возвращает false.

Функция endwith

```
function endwith(string, subStr)
```

Функция возвращает true, если строка string заканчикается строкой subStr. Иначе возвращает false.

Функция casestartwith

```
function startwith(string, subStr)
```

Функция возвращает true, если строка string начинается со строки subStr. Иначе возвращает false. Функция регистронезависимая (игнорирует различие заглавных и строчных букв).

Функция caseendwith

```
function endwith(string, subStr)
```

Функция возвращает true, если строка string заканчикается строкой subStr. Иначе возвращает false. Функция регистронезависимая (игнорирует различие заглавных и строчных букв).

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 indexOf или IndexOf

Функции indexOf и IndexOf являются синонимами.

Функция ищет первое вхождение подстроки subStr в строку string, начиная с позиции startPos.

```
function indexOf(string, subStr, startPos);
```

или

```
function indexOf(string, subStr);
```

Где:

string – (string) строка, в которой производится поиск.

subStr – (string) подстрока, поиск которой осуществляется.

startPos – (int) начальная позиция поиска (нумеруется с нулевого символа).

Функция возваращает (int) позицию первого найденного вхождения подстроки subStr в строку string, или (-1), если подстрока не найдена.

Функция caseIndexOf

Функция ищет первое вхождение подстроки subStr в строку string, начиная с позиции startPos. Функция регистронезависимая (игнорирует различие заглавных и строчных букв).

```
function caseIndexOf(string, subStr, startPos);
```

или

```
function caseIndexOf(string, subStr);
```

Где:

string – (string) строка, в которой производится поиск.

subStr – (string) подстрока, поиск которой осуществляется.

startPos – (int) начальная позиция поиска (нумеруется с нулевого символа).

Функция возваращает (int) позицию первого найденного вхождения подстроки subStr в строку string, или (-1), если подстрока не найдена.

Функция UUID

Функция формирует строку с уникальным идентификатором в формате UUID.

Функция переводит символы строки в верхний регистр.

```
function UUID();
```

Возвращает строку с уникальным идентификатором в формате:
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxx

Функция upcase/uppercase

Функция переводит символы строки в верхний регистр.

```
function upcase(s);
```

или

```
function uppercase(s);
```

Где:

s – входная строка.

Функция возвращает строку в верхнем регистре. Исходная строка при этом не изменяется.

Функция переводит строку в верхний регистр, использую правила текущей кодировки, установленной в config.txt.

Пример:

```
print(upcase("робот")); // будет выведено "РОБОТ"
```

Функция lowercase

Функция переводит все заглавные символы строки в строчные.

```
function lowercase(s);
```

Где:

s – входная строка.

Функция возвращает строку в нижнем регистре. Исходная строка при этом не изменяется.

Функция UTF8ToANSI или UTF8ToWin1251

```
function UTF8ToANSI(stringUTF8);
```

Функция переводит строку из кодировки UTF-8 в кодировку Windows 1251 и возвращает строку с результатом.

Функция ANSIToUTF8 или Win1251ToUTF8

```
function ANSIToUTF8(string1251);
```

Функция переводит строку из кодировки Windows-1251 в кодировку UTF-8 и возвращает строку с результатом.

Функция format

```
function format(formatStr, arg1, arg2, arg3,...);
```

Функция форматирует строку подобно функции printf в языках C/C++.

Где:

formatStr – (string) строка формата, содержащая параметры %d, %f, %c, %s, %04X, %s, вместо которых подставляются аргументы, переданные функции.

arg1, arg2, arg3, … – аргументы, подставляемые в параметры. Тип данных должен соответствовать параметру.

Функция возвращает (string) отформатированную строку.

Например:

```
m = 5;
b = 200
s = format("Мы собрали %d грибов и %d ягод\n", m, b);
print(s);
```

Функция HexToInt

```
function HexToInt(hex);
```

Функция переводит строку в формате hex в формат int.

Где:

hex – (string) строка, в которой записано число в формате HEX (16-ричное).

Функция возвращает (int) значение числа, которое было задано параметром hex.

Например:

```
s = "23FA";
n = HexToInt(s);
print(n);
```

Функция URLEncode/EncodeURL

```
function URLEncode(s);
```

или

```
function EncodeURL(s);
```

Функция кодирует строку для передачи ее в качестве HTTP-запроса. Кодировка строки не изменяется.

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:

- Е – единственное число
- М – множественное число.
- * – любой число.

Пример:

«РЕ» – родительный падеж, единственное число.

«*Е» – 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", "ВЕ"); // результат «РЕКУ»
s = Declination("Река", "В*"); // результат «РЕКУ | РЕКИ»
s = Declination("Глаз", "РМ"); // 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 SetCodePage

Функция задает кодовую страницу языка для 8-битной кодировки.

```
function SetCodePage(codePage)
```

Где:

codePage – (int) код кодировки:

- 1251 – Обычная кодировка Windows-1251.

- 1252 – Западно-европейская кодировка.
- 1250, 1253, 1254 – см. таблицы кодировок Windows.
- 65001 – UTF-8.

Кодировка определяет то, в какой кодировке задан текст скрипта, включая символы языка для синтезатора речи. При этом синтезатор речи должен понимать символы этой кодировки.

Важно, что файлы скрипта понимают BOM-сигнатуру UTF-8, и по умочанию преходят на UTF-8, если файл скрипта в этой кордировке. Автоматического возврата к Win-1251 не предумотрено. Нужно перезапускать ДинРобот2.

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
```

Свойство .length для строк

Свойство «.length» для строк возвращает длину строки в сиволах.

Например:

```
s = "Hello world";
len = s.length;         // получить длину строки
print("Длина строки ",len," символов\n");
```

3.6. Регулярные выражения

Регулярные выражения – достаточно известный среди программистов механизм поиска и замены строк текста по специальным шаблонам, называемыми регулярными выражениями.

Если разработчик скрипта не обладает достаточной квалификацией, лучше не использовать регулярные выражения в своём коде, т.к. написание шаблонов регулярных выражений – достаточно сложный механизм, сродни собственному языку программирования.

Цель регулярного выражения – найти в тексте подстроку текста, соответствующую заданному шаблону, и (в зависимости от используемой функции) разобрать ее по шаблону или произвести в ней замену текста по шаблону.

Далее будут приведены основные элементы шаблонов регулярных выражений. Полный перечень возможностей следует читать в Интернете. В «ДинРобот-3» подключена библиотека регулярных выражений, соответствующих грамматике «ECMAScript».

Выражение	Описание
.	Любой символ, кроме переноса строк
\n	Символ переноса строк
\r	Символ возврата каретки
\w	Любая латинская печатная буква или цифра
\W	Любой символ, не являющийся латинской печатной буквой или цифрой
\d	Любая цифра
\D	Любой символ, не являющийся цифрой
\s	Пробел, табуляция или иной пробельный символ
\S	Любой не пробельный символ
[A-Fa-f0-9]	Любой символ из обозначенных диапазонов символов. В данном случае указан диапазон от «A» до «F» (включительно), от «a» до «f», от «0» до «9»
[abcgh_]	Любой символ из указанного перечня символов
[A-F456]	Любой символ из диапазона от «A» до «F», а также цифры «4», «5», «6».
[^abc]	Любой символ, кроме перечисленных
[^A-F]	Любой символ, кроме символов из диапазона от «A» до «F».
\"	Экранирование кавычек
\[	Экранирование специальных знаков. Экранировать нужно круглые, фигурные и квадратные скобки, а также знаки «.», «,», «\», «^», «*»,«-»,«+», «?»
\\\\	Экранирование знака «\». Здесь тонкость: знак «\» при написании строк в IScript3 нужно экранировать «\\», но и в самих регулярных выражениях знак «\» нужно экранировать «\\». Отсюда при написании строки появляется учетверение «\\\\». Если не брать в расчет экранирование строк в IScript3 (например, строка была загружена откуда-нибудь), то в ней символ «\»

	должен быть экранирован только один раз «\\».

Повторы

Под повторами понимается, сколько раз должен в шаблоне повторяться одна и та же шаблонная последовательность. Повторы указывают сразу за последовательностью.

Выражение	Описание
*	0 или любое количество раз
+	1 и более раз
?	0 или 1 раз
{1,3}	Строго от 1 до 3 раз
{4}	Строго 4 раза

Например:

\d+ – любая цифра хотя бы один раз.
\s* – пробельные символы от 0 до любого количества раз.
[A-F0-9]+ – буквы от «A» до «F» и цифры от «0» до «9» хотя бы один раз.
\d{5} – строго 5 цифр подряд.
[^\"]* – любой символ, кроме кавычки, любое количество раз (включая 0).
\"? – опционально кавычка.

Возвращаемые поддиапазоны

При разборе текста важно найти в тексте искомые подстроки. Их важно вернуть или сохранить при замене. Для этого возвращаемые поддиапазоны символов нужно заключить в круглые скобки. Первые круглые скобки соответствуют первому поддиапазону, вторые – второму и т.д.

Например, пусть имеется строка текста «Мы собрали 5 грибов и 18 ягод». Требуется определить, сколько указано в строке грибов и ягод.

Для этого составляется регулярное выражение:

«(\d+) гриб.* и (\d+) ягод».

Первая последовательность «\d+» будет искать первое совпадение с цифрами. При этом последовательность «\d+» заключена в скобки, это означает, что найденную последовательность цифр нужно вернуть в качестве первого результата поиска. При этом При этом далее в тесте должно идти слово, начинающееся с символов «гриб». Далее пропускается любое количество символов, любое число раз, но главное, что после этого должна идти буква «и» а следом за ней последовательность цифр «\d+». Эта последовательность тоже заключена в скобки, это означает, что найденную последовательность цифр нужно также вернуть в качестве второго результата поиска.

Переключение «жадности»

Для переключения «жадности» поиска служит знак «?», который нужно указать после указания повтора.

Жадный поиск в спорных ситуациях двух шаблонов забирает себе большее количество искомой информации. Нежадный поиск дает возможность следующим элементам шаблона захватить нужную им часть текста. По умолчанию поиск жадный.

Например, строка текста «Мы собрали 5 грибов и 18 ягод».

Регулярное выражение:

«(\d+) гриб.*(\d+) ягод».

Обратим внимание, что шаблон «.*» должен захватить любое количество символов любое число раз. Но ведь часть текста «18 ягод» – это же ведь тоже, как бы, любые символы. Но, видимо, ожидалось, что последовательность «18 ягод» должна быть обработана шаблоном «(\d+) ягод». Совсем не дать отработать шаблону «(\d+) ягод» нельзя, но поиск по умолчанию жадный. Поэтому шаблон «.*» захватит часть текста «ов и 1», а шаблону «(\d+) ягод», останется часть текста «8 ягод».

Если же переключить жадность шаблона «.*», то шаблону «(\d+) ягод» достанется больше символов из искомой строки:

«(\d+) гриб**.*?**(\d+) ягод».

В этом случае шаблон «.*» будет захватит часть текста «ов и », а шаблону «(\d+) ягод» достанется часть текста «18 ягод».

Функция preg_match

Функция preg_match ищет совпадение текста по регулярному выражению и возвращает массив результатов совпадений.

```
function preg_match(re, str)
```

Где:

re – строка с регулярным выражением (без обрамления в «/» и флагов).

str – строка в которой производится поиск.

Функция возвращает массив (array) из строк (string) совпадений. Нулевой элемент массива – подстрока, совпавшая со всем шаблоном регулярного выражения. Первый, второй и третий и т.д. элементы массива – подстроки, соответствующие первой, второй, третьей и т.д. скобке в регулярном выражении.

Если совпадений нет, то функция возвращает null.

Пример:

```
text = "Мы собрали 5 грибов и 18 ягод";
g = preg_match("(\d+) гриб.* и (\d+) ягод",text);
if (g)
{
      print("Грибов: ",g[1],"\n");
      print("Ягод: ",g[2],"\n");
}
```

Функция preg_matchAll

Функция preg_matchAll ищет все совпадения текста по регулярному выражению и возвращает массив, в который вложен еще один массив по каждому результату совпадений.

```
function preg_matchAll(re, str)
```

Где:

re – строка с регулярным выражением (без обрамления в «/» и флагов).

str – строка в которой производится поиск.

Функция возвращает массив (array) из массивов (array) строк (string) совпадений.

Первый массив, это массив из всех результатов поиска. Вложенные массивы, это результаты поиска по каждому совпадению. Нулевой элемент вложенного массива – подстрока, совпавшая со всем шаблоном регулярного выражения. Первый, второй и третий и т.д. элементы вложенного массива – подстроки, соответствующие первой, второй, третьей и т.д. скобке в регулярном выражении.

Если совпадений нет, то функция возвращает массив нулевой длины

Пример:

```
text = "Мы собрали 5 грибов и 18 ягод. А после мы нашли 22 гриба и 40
ягод";

g = preg_matchAll("(\d+) гриб.*? и (\d+) ягод",text);
for(var i=0; i < g.length; i++)
{
      print(i+1,"й раз:\n");
      print("  Грибов: ",g[i][1],"\n");
      print("  Ягод: ",g[i][2],"\n");
}
```

В данном случае будет выведено в консоль:

```
1й раз:
  Грибов: 5
  Ягод: 18
2й раз:
  Грибов: 22
  Ягод: 40
```

Функция preg_replace

Функция preg_replace ищет все совпадения текста по регулярному выражению и заменяет их на указанный текст.

```
function preg_replace(re, to, str)
```

Где:

re – строка с регулярным выражением (без обрамления в «/» и флагов).

to – строка, на которую будет произведена замена. В этой строке можно использовать «$1» для подстановки значения из первой скобки регулярного выражения, «$2» – из второй скобки и т.д.

str – строка в которой производится поиск.

Функция возвращает строку (string) с произведенной заменой (исходная строка не изменяется)

Пример:

```
text = "Мы собрали 5 грибов и 18 ягод. А после мы нашли 22 гриба и 40
ягод";

text = preg_replace("(\d+) гриб[а-я]*", "$1 mushrooms", text);
print(text,"\n");
```

В данном случае будет выведено в консоль:

```
Мы собрали 5 mushrooms и 18 ягод. А после мы нашли 22 mushrooms и 40
ягод
```

В данном случае производился поиск выражения «(\d+) гриб[а-я]*». Это означает, что искались цифры любое количество раз большее одного, причем цифры заключены в скобки, что означает, что само число будет первым результатом поиска. Далее должно идти слово, начинающееся на «гриб» и опционально имеющее любое окончание из русских строчных букв. Найденная последовательность заменяется на «$1 mushrooms», при этом вместо «$1» подставляется значение из первой скобки регулярного выражения.

3.7. Математические функции

Основные математические функции

```
function abs(a);       // модуль
function fabs(a);      // то же самое
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 floor(a);     // округляет к наименьшему целому
function ceil(a);      // округляет к наибольшему целому
function round(a);     // округляет к ближайшему целому
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).

3.8. 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 (3.14159265358979323846264338327795)

4. Элементы векторной алгебры

4.1. Встроенные математические операторы векторной алгебры

Элементы векторной алгебры востребованы при расчетах точек и координат звеньев манипулятора или движения робота по карте.

Скрипт оперирует понятиями «скаляр», «вектор» и «матрица».

Скаряр – это обычное число. Название пошло от слова scale (масштаб), потому что скаляр в векторной алгебре обычно только и используется для масштабного коэффициента.

Вектором является массив координат или объект с полями {x,y,z}.

Например:

```
a = [10,20,30]; // вектор в виде массива
b = {x:10, y:20, z:30}; // вектор в виде объекта
```

Математически вектор, определяющий координаты точки в пространстве, и вектор направления определяются одним понятием вектор, однако интерпретируются по-разному.

Координаты точки в виде вектора содержат запись о координате X, Y, Z этой точки в некой системе координат (в базовой или локальной системе координат какого-то объекта).

Вектор направления образуется за счет вычитания координат из точки конца вектора, координат точки начала этого вектора. Вектор направления определяет не координаты точки, а координаты направления. Если вектор направления сложить с координатами точки в пространстве, то будет получены координаты точки, полученные путем отступления от этой точки в направлении вектора направления.

Вектор направления обычно требуется нормированным, т.е. единичной длины. Для нормализации вектора направления все его координаты требуется поделить на его длину. Но в IScript3 для нормализации вектора имеется отдельная функция.

Матрица – это двухмерный массив 4x4. В скриптах используются исключительно матрицы преобразования координат.

Например, объявление единичной матрицы с помощью JSON:

```
m = [[1,0,0,0],
     [0,1,0,0],
     [0,0,1,0],
     [0,0,0,1];
```

Последний столбец матрицы преобразования всегда 0,0,0,1.

Матрица преобразования обычно используется для перевода координат точек и векторов направления из локальной системы координат, заданной этой матрицей, в базовую систему координат, относительно координат которой задается эта матрица преобразования.

Первая строка матрицы определяет направление оси X локальной системы координат в базовой системе координат. Всегда дополняется нулем.

Вторая строка матрицы определяет направление оси Y локальной системы координат в базовой системе координат. Всегда дополняется нулем.

Третья строка матрицы определяет направление оси Z локальной системы координат в базовой системе координат. Всегда дополняется нулем.

Последняя строка матрицы определяет координаты начала координат локальной системы координат в базовой системе координат. Всегда дополняется единицей.

Для преобразования координат точки из локальной системы координат в базовую систему координат достаточно умножить координаты точки в пространстве, дополненной в конце единицей до 4-х компонентного вектора, на матрицу преобразования. В результате матричного умножения будут получены координаты точки, в базовой системе координат. В конце результирующий вектор также будет дополнен единицей в конце, которую можно отбросить (IScript3 отбрасывает эту единицу).

Для преобразования координат вектора направления из локальной системы координат в базовую систему координат достаточно умножить его координаты, дополненные в конце нулем до 4-х компонентного вектора, на матрицу преобразования. В результате матричного умножения будут получены координаты вектора направления, в базовой системе координат. В конце результирующий вектор будет также дополнен нулем в конце, который можно отбросить (IScript3 отбрасывает этом ноль).

Внимание! Вектора и матрицы – это массивы или объекты, т.е. являются ссылочными типами данных. Поэтому при операциях присвоения образуется ЗАВИСИМАЯ копия данных. И внесение изменений в один вектор, изменяет и его копию. Поэтому при необходимости сделать независимую копию данных, используется оператор copy.

IScript3 поддерживает следующие векторные и матричные операции.

Сложение векторов

```
a = [10,20,30];
b = [30,40,50];
c = a + b;
show(c);  // выведет: [40,60,80]
```

Обычно используется для сложения координат точки с вектором направления для получения координат точки, отложенной в направлении этого вектора.

Может работать с векторами произвольной длины.

Вычитание векторов

```
a = [10,20,30];
b = {x:30,y:40,z:50};
c = a - b;
show(c);  // выведет: {x:-20,y:-20,z:-20}
```

Обычно используется для получения вектора направления, заданного точкой конца и точкой начала этого вектора.

Может работать с векторами произвольной длины.

Векторное произведение векторов

```
a = [1,0,0];
b = [0,1,0];
c = a * b;
show(c);  // выведет: {x:0,y:0,z:1}
```

В результате векторного умножения формируются координаты вектора, перпендикулярного двум перемножаемым векторам направления. Длина результирующего вектора будет соответствовать площади трапеции, образованной

из двух перемножаемых векторов. Но обычно длина вектора не так важна, а важно само направление этого вектора.

Обратите внимание, в векторном произведении важен порядок множителей:
a * b ≠ b * a

Скалярное произведение векторов

```
a = [1,0,0];
b = [0.3,0.8,0];
c = a | b;
show(c);  // выведет: 0.3
```

Обратите внимание, что операция «|» имеет меньший приоритет, чем операция «*», поэтому выражение «a | b», лучше брать в скобки и писать «c = (a | b)», особенно, если это сложное выражение, типа: «c = p0 + (a | b) * k»;

Скалярное произведение дает величину проекции одного вектора на другой.

Может работать с векторами произвольной длины.

Умножение вектора на скаляр:

```
a = [0,1,0];
k = 5;
c = a * k;
show(c);  // выведет: [0,5,0]
```

Обычно используется для умножения единичного вектора направления на заданную величину отступа в направлении этого вектора.

Обратите внимание, что здесь важен порядок множителей. Нужно умножать вектор на скаляр, а не наоборот.

Изменение знака вектора:

```
a = [10,20,30];
b = -a;
show(c);  // выведет: [-10,-20,-30]
```

Обычно используется для умножения единичного вектора направления на заданную величину отступа в направлении этого вектора.

Преобразование координат точки по матрице (умножение вектора, дополненного единицей, на матрицу):

```
m = [[1,0,0,0], [0,1,0,0], [0,0,1,0], [10,20,30,1]];
a = [1,2,3];

c = a * m;
show(c);  // выведет: {x:11,y:22,z:33}
```

С точки зрения векторной алгебры, вектор *a* должен быть дополнен компонентом со значением 1, что соответствует преобразованию точки. Т.е. перед умножением на матрицу, вектор был расширен до *a* = [1,2,3,1] . После умножения из результирующего вектора единица в конце отбрасывается.

Обратите внимание, важен порядок умножения. Нужно умножать вектор на матрицу, а не наоборот.

Преобразование координат вектора направления по матрице (умножение вектора, дополненного нулем, на матрицу):

```
m = [[1,0,0,0], [0,1,0,0], [0,0,1,0], [10,20,30,1]];
a = [1,0,0];
```

```
c = a ^ m;
show(c);  // выведет: {x:1,y:0,z:0}
```

С точки зрения векторной алгебры, вектор *a* должен быть дополнен компонентом со значением 0, что соответствует преобразованию вектора направления. Т.е. перед умножением на матрицу, вектор был расширен до *a* = [1,2,3,0] . После умножения из результирующего вектора единица в конце отбрасывается.

Обратите внимание, важен порядок умножения. Нужно умножать вектор на матрицу, а не наоборот.

Умножение матрицы на скаляр

```
m = [[1,0,0,0], [0,1,0,0], [0,0,1,0], [10,20,30,1]];
m2 = m * 2;
show(m3); // выведет:
// [[2,0,0,0],[0,2,0,0],[0,0,2,0],[20,40,60,1]]
```

По сути, масштабирует матрицу.

Произведение матриц преобразования

```
m1 = [[1,0,0,0], [0,1,0,0], [0,0,1,0], [10,20,30,1]];
m2 = [[1,0,0,0], [0,1,0,0], [0,0,1,0], [2,3,4,1]

m3 = m2 * m1;
show(m3); // выведет:
// [[1,0,0,0],[0,1,0,0],[0,0,1,0],[12,23,34,1]]
```

С точки зрения векторной алгебры перемножение матриц преобразования формирует результирующую матрицу преобразования нескольких вложенных друг в друга систем координат.

Например, если в базовой системе координат есть объект 1 со своей локальной системой координат, заданной матрицей преобразования m1. В системе координат объекта 1 имеется объект 2 со своей локальной системой координат, заданной матрицей преобразования m2, заданной в системе координат объекта 1. Для получения матрицы преобразования из локальной системы координат объекта 2 в базовую систему координат необходимо умножить слева матрицу преобразования m2 на матрицу преобразования m1:

m = m2 * m1

Важен порядок умножения: матрица следующей в иерархии системы координат должна умножаться слева.

Обратная матрица (инверсия матрицы)

```
m = [[1,0,0,0], [0,1,0,0], [0,0,1,0], [10,20,30,1]];
m2 = !m;
show(m2); // выведет:
// [[1,0,0,0],[0,1,0,0],[0,0,1,0],[-10,-20,-30,1]]
```

Обратная матрица преобразования служит для перевода координат точки или вектора направления из базовой системы координат, в локальную систему координат, заданную исходной матрицей.

4.2. Функции векторной алгебры

Функция VecLength

Возращает длину вектора.

```
function VecLength(vec);
```

Где:

vec – вектор, заданный в виде массива произвольной длины, или в виде объекта {x,y,z}.

Функция возвращает (float) длину вектора, рассчитанную, как корень из суммы квадратов компонентов вектора.

Например:

```
a = [10,20,30,30];
len = VecLength(a);
print(len,"\n"); // выведет 47.958315
```

Функция Normalize

Функция нормализует вектор (т.е. приводит к длине 1 путем покомпонентного деления вектора на его длину).

```
function Normalize(vec);
```

Где:

vec – вектор, заданный в виде массива произвольной длины, или в виде объекта {x,y,z}.

Функция возвращает нормализованный вектор. Причем, если исходный вектор был задан в виде массива, то и результирующий вектор будет в виде массива. Если исходный вектор был задан в виде объекта {x,y,z}, то и результирующий вектор будет возвращен в виде объекта того же формата.

Пример:

```
a = [10,20,30];
b = Normalize(a);
show(b); // [0.267261, 0.534522, 0.801784]
```

Функция Distance

Функция возврашает расстояние расстояние между двумя точками, заданными координатами векторов.

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

Где:

a, b – векторы, заданные в виде массива произвольной длины, или в виде объектов {x,y,z}.

Функция возвращает (float) расстояние между двумя точками a и b.

Функция Matrix или TranslateMatrix

Функция создает единичную матрицу или матрицу перемещения. Функции Matrix и TranslateMatrix являются синонимами.

```
function Matrix();
```

или

```
function Matrix(x,y,z);
```

или

```
function Matrix(vector);
```

Где:

x,y,z – координаты перемещения для матрицы перемещения.

vector – массив (array) вида [x,y,z] или объект (object) вида {x,y,z} с координатами пемещения для матрицы перемещения.

Функция возвращает матрицу вида:

```
[[1,0,0,0],
 [0,1,0,0],
 [0,0,1,0],
 [x,y,z,1]];
```

Если не указан вектор vector или координаты x,y,z, то координаты x,y,z приравниваются к нулю, а фунция возвращает единичную матрицу.

Функция RotXMatrix

Функция создает матрицу поворота вокруг оси X.

```
function RotXMatrix(a);
```

Где:

a – угол поворота в радианах.

Функция возвращает матрицу вида:

```
[[1, 0,       0,          0],
 [0, cos(a), -sin(a),  0],
 [0, sin(a),  cos(a),  0],
 [0,      0,         0,   1]
```

Если не указан параметр *a*, то угол приранимается к нулю, а функция возвращает единичную матрицу.

Функция RotYMatrix

Функция создает матрицу поворота вокруг оси Y.

```
function RotYMatrix(a);
```

Где:

a – угол поворота в радианах.

Функция возвращает матрицу вида:

```
[[cos(a), 0, -sin(a),   0],
 [0,        1,        0,   0],
 [sin(a),  0,   cos(a), 0],
 [0,      0,         0,    1]
```

Если не указан параметр *a*, то угол приранимается к нулю, а функция возвращает единичную матрицу.

Функция RotZMatrix

Функция создает матрицу поворота вокруг оси Z.

```
function RotZMatrix(a);
```

Где:

a – угол поворота в радианах.

Функция возвращает матрицу вида:

```
[[cos(a), -sin(a),  0,  0],
 [sin(a),  cos(a),  0,  0],
 [0,           0,       1,  0],
 [0,       0,           0,  1]
```

Если не указан параметр *a*, то угол приранимается к нулю, а функция возвращает единичную матрицу.

Функция ScaleMatrix

Функция создает матрицу масштабирования.

```
function ScaleMatrix(s);
```

или

```
function ScaleMatrix(sx,sy,sz);
```

или

```
function ScaleMatrix(vector);
```

Где:

s – (float) масштабный коэффициент, единый по всем осям.

sx, sy, sz – (float) масштабные коэффициенты по сооветствующим осям.

vector – (array [x,y,z] или объект {x,y,z}) с масштабными коэффициентами по осям.

Функция возвращает матрицу вида:

```
[[sx, 0,  0,     0],
 [0,   sy,   0,   0],
 [0,    0,    sz,  0],
 [0,    0,     0,  1]
```

Где: sx, sy, sz – соответствующие масштабные коэффициенты по осям.

При вызове без параметров возвращает единичную матрицу.

Функция PlaneMatrix

Функция создает матрицу плоскости с точкой начала координат.

```
function PlaneMatrix(point0, pointX, pointY);
```

Где:

point0 – (array [x,y,z] или объект {x,y,z}) точка начала координат.

pointX – (array [x,y,z] или объект {x,y,z}) любая точка на оси X плоскости, отличная от point0.

pointY – (array [x,y,z] или объект {x,y,z}) любая точка на оси Y плоскости, отличная от point0 и от pointX.

Функция возвращает матрицу плоскости, в которой начало координат находится в точке point0. Ось X направлена в направлении точки pointX и имеет единичную длину. Ось Y направлена в направлении точки pointY и имеет единичную длину. Ось Z перпендикулярна плоскости, образованной осями X и Y, рассчитанную, как нормированное векторное произведение оси X и оси Y. Т.о.

если ось X направлена вправо, а ось Y вверх, то ось Z будет направлена «на себя». Если ось X направлена вправо, а ось Y вниз, то ось Z будет направлена «от себя».

Методика расчета:

X = normalize(pointX - point0);
Y = normalize(pointY - point0);
Z = normalize(X×Y)
result = [[X.x, X.y, Z.z, 0],
[Y.x, Y.y, Y.z, 0],
[Z.z, Z.y, Z.z, 0],
[point0.x, point0.y, point0.z, 1.0]

Функция WorldPointToMatrix

Функция переводит координаты точки и ее ориентации в матрицу преобразования.

```
function WorldPointToMatrix(worldPoint);
```

Где:

worldPoint – объект (object) с полями {x,y,z,ox,oy,oz} или {x,y,z,roll,pitch,yaw}.

Здесь: x, y, z – координаты точки. ox,oy,oz,roll,pitch,yaw – углы Эйлера ориентации точки в градусах.

Функция вовзращает матрицу в виде:

[[m00,m01,m02,m03],
[m11,m11,m12,m13],
[m20,m21,m22,m23],
[m30,m31,m32,m33]];

Объект worldPoint обычно используется в функциях управления руками и манипуляторами робота.

Функция MatrixToWorldPoint

Функция переводит матрицу в координаты точки и ее ориентации.

```
function MatrixToWorldPoint(mat4);
```

Где:

mat4 – матрица преобразования вида:
[[m00,m01,m02,m03],
[m11,m11,m12,m13],
[m20,m21,m22,m23],
[m30,m31,m32,m33]];

Функция возвращает объект (object) с полями {x,y,z,ox,oy,oz}. Компоненты ox,oy,oz заданы в градусах.

Объект worldPoint обычно используется в функциях управления руками и манипуляторами робота.

Функция MatrixRow

Функция получает или заменяет строку матрицы 4x4.

Установка строки:

```
function MatrixRow(mat4, rowIndex, value);
```

или получение строки:

```
function MatrixRow(mat4, rowIndex);
```

Где:

mat4 – матрица 4x4 в виде двухмерного массива.

rowIndex – номер строки от 0 до 3 включительно.

value – значение строки в виде массива или в виде объекта {x,y,z}

В случае установки строки функция не возвращает ничего. В случае получения строки, функция возвращает значение строки в виде объекта {x,y,z}.

Например:

```
m = [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]];

v = [2,3,4];
MatrixRow(m, 3, v); // заменить строку 4. теперь:
// m = [[1,0,0,0],[0,1,0,0],[0,0,1,0],[2,3,4,1]];

a = MatrixRow(m,0); // получить строку 0
// a = {x:1,y:0,z:0};
```

5. Фреймы. Распознавание речи и событий

5.1. Введение

В скриптовый язык IScript3 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.

5.2. 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 IScript3.

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.

5.3. Синонимы в словесной предпосылке

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

[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.

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

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

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

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

или

```
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")
   {
      PlaySpeech("Меня зовут Маша", "Мое имя Маша");
   }
...
```

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

5.4. Слова с произвольным окончанием во фрейме

В ряде случаев при формировании фреймов очень не удобно указывать слова во всех возможных падежах. Поэтому можно указать знак «~» в конце слова, например: «компани~». Этому слову будет соответствовать все слова, которые начинаются с «компани» и могут иметь произвольные окончания. Напрмер: «компания», «компании», «компанией».

Пример:

```
...
   frame("Расскажи/скажи о/про компани~")
   {
      ...
   }
...
```

Такой фрейм может быть активизирован фразой: «**Скажи** что-нибудь **про** вашу **компани**ю».

5.5. Необязательные слова

Во фрейме могут присутствовать необязательные слова. Присутствие этих слов в сказанной человеком фразе повышает приоритет выбора данного фрейма по сравнению с другими аналогичными фреймами за счет того, что в нем будут активировано больше слов.

Чтобы задать такие слова их нужно заключить в квадратные скобки.

Например:

```
...
   frame("пойдем гулять [на улицу]")
   {
   }

   frame("Расскажешь про мероприятие")
   {
      ...
   }
...
```

В данном случае первый фрейм будет активизироваться фразой «Пойдем гулять». А фраза «Расскажешь про мероприяние» активизирует второй фрейм.

Однако фраза «**Пойдем гулять на улицу**, и там **расскажешь** мне **про мероприятие**» активизирует именно первый фрейм, потому что в чем фраза активизировала 4 слова, а во втором фрейме только 3.

5.6. Синонимы с необязательным словом

Во фрейме могут присутствовать синонимы (альтернативы) с необязательным словом. Присутствие этих слов в сказанной человеком фразе повышает приоритет выбора данного фрейма по сравнению с другими аналогичными фреймами за счет того, что в нем будут активировано больше слов.

Для этого достаточно в середине слов-синонимов указать одну пустую альтернативу.

Например:

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

```
    {
        ...
    }
...
```

Иной способ указания альтернативы с необязательным словом, это заключение ее в квадратные скобки. 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».

5.7. Порядок слов в словесной предпосылке

По умолчанию слова в словестной предпосылки могут присутствовать в предложении в любом порядке. Например, фрейм:

```
...
    frame("Расскажи про мероприятие")
    {
        ...
    }
...
```

Может активизироваться фразой: «Про мероприятие расскажи».

Однако, если порядок слов в предложении важен необходимо использовать знак равно «=» перед началом фрейма. Например:

```
...
    frame("=Расскажи про мероприятие")
    {
        ...
    }
...
```

Знак «=» запрещает перестановку слов в предложении. Такой фрейм может быть активирован фразой «Расскажи про мероприятие», однако фраза «Про мероприятие расскажи» данный фрейм активизировать не будет.

Порядок слов в предложении актоматически становится важен при использовании фреймов-событий, начинающихся со знака «* ». Т.е. во фрейме:

```
...
    frame("* my event")
    {
        ...
    }
...
```

Будет важен порядок слов.

5.8. Приоритет активизации фрейма

Если требуется повысить приоритет распознавания фрейма необходимо использовать в нем один или несколько знаков «!». Каждый знак «!» повышает приоритет распознавания данного фрейма на 2 слова. Например:

```
...
   frame("Хочу кушать")
   {
   }

   frame("!Да")
   {
      ...
   }
...
```

В данном случае фраза «Да, возможно я хочу кушать», актизирует именно второй фрейм, так как у него приоритет распознавания выше, несмотря на то, что в нем было активизировано лишь одно слово «Да».

5.9. Приоритет фреймов во временных фреймсетах

Про временные фреймсеты см. далее. Однако следует отметить, что приоритет распознавания фреймов во временных фреймсетах на 0.5 больше, чем у обычных фреймов. Например:

```
frameset("dialogs",1)
{
   frame("Да")
   {
      ...
   }
}
...
frameset("temp",1)
{
    frame("Да")
    {
        ...
    }
}
```

В данном случае, сказанное «Да» активизирует фрейм во фреймсете «temp», пока этот фреймсет существует (существует он 20 секунд или активизации какого либо фрейма).

5.10. Жесткое начало или конец предожения

Знак «:» в начале текста фрейма запрещает присутствие других слов в начале предложения пользователя. А знак «:» в конце текста фрейма запрещает

присутствие в предложении пользователя других слов, после слов, указанных в тексте фрейма.

Например:

```
...
   frame(":пока")
   {
   }

...
```

Данный фрейм может быть активирован фразой «**Пока**, робот». Но фраза «Мы стоим **пока** можем» данный фрейм не активизирует, т.к. в ней перед словом «пока» есть другие слова.

Другой пример:

```
...
   frame(":пока:")
   {
   }

...
```

Данный фрейм может быть активизирован только фразой «Пока». Но фраза «Пока мы живы, вы не умрем» данный фрейм не активизирует, т.к. к ней после слова пока присутствуют другие слова. Также фраза «Мы постоим пока», тоже не активизирует данный фрейм, т.к. в ней есть слова перед словом «пока».

5.11. Автозамены перед сравнением

Чтобы упростить составление фреймов, в ДинРобот применяется система предварительной подготовки слов, позволяющая не задавать множество синоноимов слов. Применяются следующая система:

1. Все числительные заменяются цифрами. Например: «Первый», будет замен на «1». Включая слова «Во-первых», а также «один», «одним» и т.п.
2. Удаляется частика «-ка» в конце слова.
3. Удаляются все знаки «-». Например: «гуляй-поле» эквивалетно «гуляйполе».
4. Союзы «во», «изо», «со», «подо», «надо» заменяются на «в», «из», «с», «под», «над» и т.д.

Например:

```
...
   frame("Убери с дивана|стола|стула")
   {
   }

...
```

Данные фрейм может быть активизирован фразой: «Убери**-ка со** стола».

В данном случае частица «-ка» будет удалена, а союз «со» будет заменен на «с». Поэтому данный фрейм все равно будет активирован, несмотря на то, что в нем указаны не совсем те слова, что были в предложении пользователя.

5.12. 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> короб~ ")
   {
       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.

Следует отметить, что нетиповизированные параметры имеют приоритет распознавания 0.75 слова. Например:

```
...
   frame("Принеси <n> короб~ ")
   {
       var n = TextOf("n");
       ...
   }
   frame("Принеси 5 коробок")
   {
       var n = TextOf("n");
       ...
   }
```

```
...
```

Фраза «Принеси 5 коробок» будет активизировать именно второй фрейм. А фраза «Принеси 6 коробок» активизирует первый фрейм.

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», «Number1», ««Number2», «Number3», «Number4» или «Digits» (большие или малые буквы не важны).

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

Под числами (Number1) подразумевается слово, состоящее строго из одной цифры.

Под числами (Number2) подразумевается слово, состоящее строго из двух цифр.

Под числами (Number3) подразумевается слово, состоящее строго из трех цифр.

Под числами (Number4) подразумевается слово, состоящее строго из четырех цифр.

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.

5.13. 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>") {...}
}

SpeechRecognizer(true); // включить распознавание речи
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.

5.14. Текст и значение параметра

Текст параметра. Функция TextOf

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

Для получения текста параметра используется функция TextOf.

```
function TextOf(paramName);
```

Где:

paramName – (string) название параметра фрейма.

Функция **TextOf** 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(…).

Значения параметра. Функция ValueOf

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

```
function ValueOf(paramName);
```

Где:

paramName – (string) название параметра фрейма.

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

При этом производится выполнение программного следствия фрейма, активизированного указанным параметром. 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");
        // objID будет 1, 2 или 3 в зависимости от объекта
        SetBrowserURL("http://content/show.php?id=" +
objID);
    }
    ...
}
frameset("Объекты")
{
    frame("Торт") { return 1; }
    frame("Пирожное") { return 2; }
    frame("Шоколад") { return 3; }
}

SpeechRecognizer(true); // включить распознавание речи
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, активированного параметром «что», в данном случае это фрейм «Пирожное». При этом программное следствие фрейма «Пирожное» вернет значение 2.

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

5.15. Запрос из программного кода

Запрос с ожиданием результата. Функция LangQuery

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

Для создания запроса служит функция LangQuery(…).

```
function LangQuery(query);
```

Где:

query – (string) текст запроса на естественном языке.

В качестве параметров этой функции передается текст запроса на естественном языке.

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
```

Асинхронный запрос. Функция LangQueryAsync

Функция аналогична функции LangQueryAsync, за исключением того, что функция отправляет запрос в базу знаний на естественном человеческом языке без ожидания результата выполнения.

```
function LangQueryAsync(query);
```

Где:

query – (string) текст запроса на естественном языке.

В качестве параметров этой функции передается текст запроса на естественном языке.

Функция всегда возвращает null.

Запрос, сделанный функцией, ставится в очередь запросов, а задача, рожденная фреймом, ставится в очередь выполнения задач, согласно своему приоритету. Выполнение задач начинается с того момента, как только скрипт доходит до выполнения какой-либо функции с ожиданием (sync, Sleep, GoToPlace и т.д.).

В целом функция делает те же действия, что и система распознавания речи с распознанным текстом.

5.16. 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.

5.17. 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.

Функция CreateFrameset

Функция создает фреймсет, аналогично оператору «frameset». Однако созданный таким образом фреймсет должен быть заполнен фреймами только с помощью функции CreateFrame. Кроме того, если фреймсет уже сушестовал, то он начинает дополняться, а не очищаться.

```
function CreateFrameset(name, prior);
```

или:

```
function CreateFrameset(name);
```

Где:

name – (string) название фреймсета. Может содержать подстроки «TEMP» и «DUTY», аналогично оператору «framset».

prior – (int) приоритет выполнения или (-1) (предметный фреймсет). Если параметр опущен, а фреймсет уже существовал, то его приоритет не меняется.

Возвращает true в случае удачи или false в случае ошибки.

Если фреймсет с аналогичным названием уже существовал, то он не чищается, а начинает дополняться новыми фреймами.

Пример:

```
suffix = "A";
CreateFrameset("myFrameset"+suffix);
CreateFrame("Малина","return 1;");
CreateFrame("Клубника",2);
CreateFrame("Клюква",3);
```

Функция CreateFrame

Функция создает фрейм в последнем фреймсете, созданном с помощью функции CreateFrameset.

```
function CreateFrame(condition, script);
```

или:

```
function CreateFrame(condition, retValue);
```

или

```
function CreateFrame(condition);
```

Где:

condition – (string) словесная предпосылка фрейма.

script – (string) текст скрипта, которой должен выполнится при активации фрейма. ***Скрипт не должен уничтожать сам себя путем удаления фреймсета, которому принадлежит, или очистки фреймов из него. Иначе это привет к аварийному завершению «ДинРобот»!***

retValue – (любой тип, отличный от string) значение, которое должен вернуть фрейм.

Возвращает true в случае успеха или false в случае ошибки. Ошибка может возникать в случае невозможности разбора условия condition или в очень редких случаях нарушения работы с памятью.

Вызов функции «CreateFrame("малина", "return 5;")» аналогична ее вызову с параметрами: «CreateFrame("малина", 5)».

В случае вызова без второго параметра создается фрейм, который при активации будет всегда возвращать true.

Пример см. выше функцию CreateFrameset.

Функция DeleteFrameset

Функция удаляет фреймсет.

```
function DeleteFrameset(name);
```

Где: name – название фреймсеса.

Функция удаляет указанный фреймсет.

Функция NotFoundText

Функция возвращает последний нераспознанный текст, сказанный пользователем.

```
function NotFoundText();
```

Функция возвращает строку (string) с последним нераспознанным текстом.

Пример:

```
frameset("my",1)
{
   frame("Привет") // распознанный текст и реакция на него
   {
      say("Привет");
   }
   frame("* notFound")
   {
      // если не распознали, переспросить
      say("Вы сказали, "+GetLastSpeech());
   }
}

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

5.18. Дерево ведения диалога. Временный фреймсет

В языке IScript3 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");
    }
}
```

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 приоритет. 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);
     }
}
SpeechRecognizer(true); // включить распознавание речи
while(true) sync();
```

5.19. Функции управления системой распознавания речи

Функция SpeechRecognizer/SpeechRecognition

Функция включает или отключает систему распознавания речи.

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

или, синоним:

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

или

```
function SpeechRecognition();
```

или, синоним:

```
function SpeechRecognizer();
```

Где:

onOff – включить (true) или выключить (false) систему распознавания речи.

Функция возвращает состояние (bool) активности системы распознавания речи.

При вызове функции без параметров, функция лишь возвращает состояние активности.

Следует отметить, что робот автоматически блокирует распознавание речи во время собственной речи робота, поэтому специально выключать распознавание речи перед использованием функции PlaySpeech не требуется.

Результат, возвращаемый функцией, возвращает лишь состояние активности, установленное с помощью данной функции, но не состояние блокировки распознавания во время воспроизведения речи робота.

При завершении работы скриптов распознавание речи автоматически завершается.

Функция SetSpeechDriver

Функция изменяет язык распознавания речи, установленный по умолчанию.

```
function SetSpeechDriver();
```

или

```
function SetSpeechDriver(lang);
```

или

```
function SetSpeechDriver(driver, voice,
                                 defPitch, pitchK,
defRate);
```

или

```
function SetSpeechDriver(driver, voice,
                           defPitch, pitchK, defRate,
waveRate);
```

Где:

lang	–	язык (string). Например "RU".
driver	–	название драйвера. Поддерживается только: «SAPI» и «SAPIx86») для Windows, и «RHVoice» для Linux.
voice	–	название голоса (используется, как часть названия голоса).
defPitch	–	(int) тон голоса по умолчанию. Нормальное значение 0.

pitchK – (float) коэф.повышения тона на один знак «>». Нормальное значение 1. Однако у разных голосов этот коэффициент может варьироваться.
defRate – (int)скорость воспроизведения. Нормальное значение 0.
waveRate – (int) частота WAVE-файла. По умолчанию 16000, подходит для большинства голосов.

При вызове без параметров производится переключение на драйвер языка, заданный в config.txt.

Язык, на который производится смена, должен поддерживаться текущей системой распознавания речи.

Язык, измененный с помощью данной функции, остается измененным даже после завершения работы скриптов, вплоть до завершения работы программы.

Пример:

```
SetSpeechDriver("SAPI","Elena",0,1,0);
```

5.20. Специальные события системы распознавания речи

Получение строки нераспознанного запроса

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

Получение события начала распознавания

При начале детектирования речи возникает событие «* SPEECH».

6. Стандартные события

Фреймообразные структуры языка IScript3 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 IScript3.

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()
* faceIn	Система трекинга лиц обнаружила лицо. Персона уже может быть определена
* faceOut new	Лицо пропало из кадра, потому что появилось новое лицо.
* faceOut out	Лицо пропало из кадра и вышел тайаут ожидания появления лица в кадре. В кадре больше нет лиц.
* hello	Система трекинга лиц вычислила условия, когда нужно поздороваться. Если GetFacePersonId() возвращает не пустую строку, то нужно здороваться с известной персоной, иначе с неизвестной.
* compliment	Система трекинга лиц требует сказать комплимент при новой встрече. Условия комплимента возникают тогда, когда робот уже видел недавно данную персону, и вот он снова ее увидел. Здороваться повторно в такой ситуации было бы странно, поэтому нужно сказать комлимент. Персону можно определить функций GetFacePersonId()
* acquaintance	Система трекинга лиц захотела познакомиться.
* faceOut	Система трекинга лиц обнаружила пропадание лица
* SPEECH	Событие начала детектирования речи
* LIGHTTRAFFIC *n*	Робот заехал в зону светофора *n*.

Пример обработки событий:

```
frameset("events", 100)
{
    frame("* hello")
    {
        PlaySpeech("Приветствую");
    }


    frame("* faceOut")
    {
        PlaySpeech("До свидания");
    }
}
```

```
FaceRecognizer(true); // включить систему детектирования
лиц
while(1) sync();
```

7. Функции работы с JSON и XML

Функция json_decode. Разбор строки в формате JSON

Функция разбирает строку текста, как JSON-объект.

```
function json_decode(str);
```

или

```
function json_decode(str, isUTF8);
```

Где:

str – текст в формате JSON.

isUTF8 – (bool) автоматический перевод строк из UTF-8 в ANSI (по умолчанию false).

Функция возвращает скалярное значение, строку, массив или объект в зависимости от того, чем является строка str. В случае ошибки возвращает null.

Пример:

```
str = "{x:100, y:200, a:[10,20,30]}";

// разобрать строку, превратить в объект json
json = json_decode(str);
if (json)
{
   // вывести результат
   print("x=", json.x, "  y=", json.y, "\n");
   for(i=0; i<count(json.a);i++)
   {
       print("a[",i,"]=", json.a[i],"\n");
   }
}
```

В результате примера будет выведено:

```
x=100 y=200
a[0]=10
a[1]=20
a[2]=30
```

Функция json_encode. Перевод объектов в строку JSON

Функция формирует строку текста в формате JSON из объектов IScript3.

```
function json_encode(obj);
```

Где:

obj – объект языка IScript3 (значение, строка, массив, объект).

Функция возвращает строку, закодированнную в формате JSON. В случае ошибки возвращает null.

Пример:

```
obj = object();
obj.x = 100;
obj.y = 200;
obj.a = array(10,20,30);

// преобразовать obj в JSON-строку
s = json_decode(obj);
```

```
if (s)
{
   // вывести результат
   print(s);
}
```

В результате примера будет выведено:

```
{
  x:100,
  y:200,
  a:[
    10,
    20,
    30
  ]
}
```

Функция xml_decode. Разбора строки в формате XML

Функция разбирает строку текста, как XML-объект.

```
function xml_decode(str);
```

или

```
function xml_decode(str, isUTF8);
```

Где:

str – текст в формате XML.

isUTF8 – переводить строки из UTF-8 в ANSI.

Функция возвращает объект, содержащий поля XML. При этом XML-теги становятся объектами, название тега возвращается полем **tagName**. Атрибуты тега под своими именами проецируются в поля объекта. Дочерние теги становятся массивом **children**. Простой текст тега становится полем **text**.

Кордировка, указанная в теге <?xml>, игнорируется. Для указания кодировки следует использовать параметр isUTF8 или после разбора переводить кодировку с помощью скрипта.

Пример:

```
xml = '<?xml version="1.0" encoding="windows-1251"?>
      '<rooms scale="1.0">"+
      '  <room name="кухня" x="100" y="100" />'+
      '  <room name="спальня" x="0" y="0">Большая</room>'+
      '</room>';

// разобрать строку превратить в объект
obj = xml_decode(xml);

// вывести на экран
print("obj.tagName=", obj.tagName,"\n"); // выведено
«rooms»
print("obj.scale = ", obj.scale, "\n"); // выведено «1.0»
for(i=0; i < count(obj.children); i++)
{
   // если тег «room»
   if (obj.children[i].tagName=="room")
   {
```

```
      print("room ", i,":\n");
      print("  name=", obj.children[i].name, "\n");
      print("  x=", obj.children[i].x, "\n");
      print("  y=", obj.children[i].y, "\n");

      // вывести простой текст тега
      if (obj.children[i].text)
         print("  простой текст=",
obj.children[i].text,"\n");

   }
}
```

Функция xml_encode. Формирования строки в формате xml

Функция переводит объект в формат JSON

```
function xml_encode(obj);
```

или

```
function xml_encode(obj, isUTF8);
```

Где:

obj – число, строка, массив или объект.

isUTF8 – (bool) переводить ли кодироку в UTF-8.

Функция обратная к функции xml_decode. Объект должен иметь те же поля, что и объект, после фукнкции xml_decode.

Функция возвращает строку, которая содержит описание объекта в формате xml.

Пример:

```
// формирование полей объекта
obj = object();
obj.tagName="root"; // название тега
obj.x = 100; // атрибуты
obj.y = 200;

obj.children = array(); // дочерние теги

obj.children[0] = object();
obj.children[0].tagName="prop";    // название тега
obj.children[0].text = "Большой";  // простой текст

obj.children[1] = object();
obj.children[1].tagName="prop";    // название тега
obj.children[1].text = "Малый";    // простой текст

// превратить объект в строку xml
s = xml_encode(obj);

// вывести на экран
print(s);
```

В результате примера будет выведено:

```
<?xml version="1.0" encoding="windows-1251">
```

```
<root x="100" y="100">
  <prop>Большой</prop>
  <prop>Малый</prop>
</root>
```

8. Управление синтезатором речью и звуком

Функция PlaySpeech/Say/say

Функция воспроизводит речь синтезатором речи.

```
function PlaySpeech(s1);
```

или

```
function PlaySpeech(s1,s2);
```

или

```
function PlaySpeech(s1,s2,s3);
```

или

```
function PlaySpeech(s1,s2,s3,...,sn);
```

или

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

или

```
function PlaySpeech();
```

Функции Say и say являются синонимыми для функции PlaySpeech.

Здесь:

$s1,s2,s3,\ldots s_n$ – строка для в синтезатора речи. Функция выбирает случайную строку из предложенных. В строку доступны вставки:

- эмоций:
  - :) – улыбка
  - :(– грусть.
  - :| – нормальное состояние.
  - <3 – глаза-сердечки.
- переменные:
  - {robotName}
- вопросительная интонация:
  - ?
- случайный вариант:
  - (R:вариант$_1$| вариант$_2$| вариант$_3$|…|вариант$_n$)
- вариант в зависимости от пола робота:
  - (G:мужской|женский)
- понизить тон:
  - <
- повысить тон:
  - >
- восстановить тон:
  - ~

При указании PlaySpeech(false) синтезатор речи замолкает.

При вызове без параметров функция возвращает true, если синтезатор речи активен, в противном случае функция возвращает номер выбранного варианта речи.

Пол робота задается функцией RobotGender.

Речь воспроизводится асинхронно. Функция не ожидает завершения воспроизведения.

При остановке скриптов речь замолкает. Поэтому, если пользователь хочет услышать всю фразу целиком, скрипт должен ожидать окончание

воспроизведения, например, за счет проверки путем вызова функции без параметров.

Пример:

```
robotName = "Макс";
RobotGender(0); // мужской пол

// воспроизвести одну из фраз
Say("Меня зовут {robotName}. А Вас?",
           "Привет. Моё имя {robotName}. А как вас
>>>зовут",
           "Приветствую Вас. Меня зовут {robotName}. А
>>>вас");

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

// сказать в зависимости от пола робота: «Рад видеть вас»
или
// «Рада видеть вас»
PlaySpeech("(G:Рад|Рада) видеть Вас");

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

Функция GetLastSpeech

Функция возвращает текст последней фразы, сказанной роботом.

```
function GetLastSpeech();
```

Функция возвращает строку (string) с последним сказанным роботом текстом.

Функция RobotSilenceTime

Функция возвращает время молчания робота в миллисекундах, т.е. сколько миллисекунд прошло с момента окончания последней речи робота. Если робот вообще ничего не говорил, то 1000000 мс.

```
function RobotSilenceTime();
```

Функция HumanSilenceTime

Функция возвращает время молчания человека в миллисекундах, т.е. сколько миллисекунд прошло с момента распознавания последней фразы человека. Если робот вообще ничего не слышал, то 1000000 мс.

```
function HumanSilenceTime();
```

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 в секции [SPEECH].

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();
```

9. Управление системой распознавания лиц

9.1. События системы распознавания лиц

- «* faceIn» – в кадре появилось новое лицо.
- «* hello» – настало время поздороваться.
- «* acquaintance» – настало время знакомиться.
- «* faceOut» – лицо пропала (или появилась другая персона).

Пример:

```
frameset("events",100)  // фреймсет для ловли событий
{
  frame("* hello")  // фрейм ловли события «* hello»
  {
     var person = GetFacePerson(); // получить персону

     // определить имя персоны
     var name = '';
     if (person!='') name = PersonName(person);

     if (name!='') // поздороваться, если узнали человека
         PlaySpeech("Здравствуйте, "+name);
     else // поздороваться с неизвестным
         PlaySpeech("Здравствуйте");
  }
  ...
}

FaceRecognizer(true); // включить распознавание лиц
while(true) sync();   // главный цикл скрипта
```

9.2. Функции системы распознавания лиц

Функция FaceRecognizer

Функция включает или выключает систему распознвания лиц. По умолчанию она выключена.

```
function FaceRecognizer(active);
```

или

```
function FaceRecognition(active);
```

или

```
function FaceRecognizer();
```

или

```
function FaceRecognition();
```

Функции SpeechRecognizer и SpeechRecognition являются синонимыми.
Здесь:

– active – (bool) включить или выключить систему распознавания лиц.

Функция возраразщает (bool) текущее состояние системы распознавания лиц.

При вызове без параметров просто определяет состояние.

Функция GetFaceCount

Функция возвращает (int) текущее количество людей в кадре.

```
function GetFaceCount();
```

Функция работает асинхронно, поэтому число лиц, возвращаемых функцией GetFaceRects, может не совпадать с результатом, возвращаемым функций GetFaceCount, т.к. запросы были произведены в разное время.

Функция IsFace

Функция возвращает true, если прямо сейчас в кадре есть лицо.

```
function IsFace();
```

Функция эквивалентна конструкции GetFaceCount() > 0.

Функция GetFaceRects

Функция возвращает массив координат прямоугольников, обрамляющих всех людей в кадре.

```
function GetFaceRects();
```

Функция возвращает массив объектов со следующими полями:

- x – (int) координата x центра прямоугольника (пиксели).
- y – (int) координата y центра прямоугольника (пиксели).
- width – (int) ширина прямоугольника (пиксели).
- height – (int) высота прямоугольника (пиксели).
- left – (int) левая координата x прямоугольнка (пиксели).
- top – (int) верхняя координата y прямоугольнка (пиксели).
- right – (int) правая координата x прямоугольнка (пиксели).
- bottom – (int) нижняя координата y прямоугольнка (пиксели).
- imageWidth – (int) ширина изображения, на котором искалось лицо.
- imageHeight – (int) высота изображения, на котором искалось лицо.
- imageFOV – (float) угол обзора камеры по ширине (градусы).

Кол-во элементов массива соответствует количеству выделенных лиц.

Если в кадре нет людей, то возвращается пустой массив.

Функция работает асинхронно, поэтому число лиц, возвращаемых функцией GetFaceRects, может не совпадать с результатом, возвращаемым функций GetFaceCount, т.к. запросы были произведены в разное время.

Функция GetFaceRect

Функция возвращает координаты прямоугольника, обрамляющего главную персону в кадре.

```
function GetFaceRect();
```

Функция объект со следующими полями:

- x – (int) координата x центра прямоугольника (пиксели).
- y – (int) координата y центра прямоугольника (пиксели).
- width – (int) ширина прямоугольника (пиксели).
- height – (int) высота прямоугольника (пиксели).
- left – (int) левая координата x прямоугольнка (пиксели).
- top – (int) верхняя координата y прямоугольнка (пиксели).
- right – (int) правая координата x прямоугольнка (пиксели).
- bottom – (int) нижняя координата y прямоугольнка (пиксели).
- imageWidth – (int) ширина изображения, на котором искалось лицо.
- imageHeight – (int) высота изображения, на котором искалось лицо.
- imageFOV – (float) угол обзора камеры по ширине (градусы).

Если в кадре нет главной персоны, то возвращаются координаты прямоуольника, обрамляющего последнее положение человека в кадре.

Функция GetFacePerson

Функция возвращает идентификатор персоны (string) в кадре, если она была детектирована за сеанс трекинга.

```
function GetFacePerson();
```

или

```
function GetFacePersonId();
```

Идентификатор персоны представляет собой название папки из папки FaceDB, где лежит информация по данной персоне.

Если персона не найдена возвращает пустую строку.

Функция SetSearchPerson или SetSearchPersonId

Функции SetSearchPerson и SetSearchPersonId являются синонимами

Функция задает идентификатор персоны, который должен быть главным при поиске лиц. Робот будет следить именно за данной персоной, если она есть в кадре.

```
function SetSearchPerson(personId);
```

Здесь:

personId – (string) идентификатор персоны.

Функция не возвращает ничего.

Для отмены действия функции ее следует вызвать с пустой сторокой или со значеием null.

Использование данной функции замедляет работу системы распознавания лиц, поэтому не задавайте главную персону дольше, чем это требуется.

При перезапуске скриптов главная персона отменяется.

При работе алгоритма FollowFace главная персона заменяется, а по окончанию работы алгоритма полностью отменяется.

Функция GetFaceAge

Функция возвращает примерный возврат (int) лица в кадре.

```
function GetFaceAge();
```

В случае ошибки или невозможности определения возвраста возвращает (-1).

Функция GetFaceGender

Функция возвращает пол (int) лица в кадре.

```
function GetFaceGender();
```

Возвращает: 0 – мужчина. 1 – женщина.

В случае ошибки или невозможности определения пола возвращает (-1).

Функция GetFaceEmotion

Функция возвращает эмоцию (int) лица в кадре.

```
function GetFaceEmotion();
```

Возвращает:

- 0 – невозможно определить эмоцию.
- 1 – злость.
- 2 – испуг.
- 3 – счастье.
- 4 – нейтральное.
- 5 – грусть.
- 6 – сюрприз (удивление).

Функция GetFacePassedTime

Функция возвращает время (int) в секундах с момента последнего появления данной персоны.

```
function GetFacePassedTime();
```

Для неизвестных персон это время 100000.

Функция PersonName

Функция устанавливает или определяет имя указанной персоны.

```
function PersonName(personId, name);
```

или

```
function PersonName(personId);
```

Здесь:

- personId – (string) идентификатор персоны (название папки из FaceDB).
- name – (string) устанавливаемое имя персоны.

В варианте с двумя параметрами функция задает имя персоны и возвращает true в случае успеха или false в случае ошибки.

В варианте с одним параметром функция возврает имя персоны (string). Если имя персоны ни разу не задавалось, то функция возвращает строку, в которой из идентификатора персоны исключены все цифры.

Функция PersonRights

Функция устанавливает или определяет права персоны.

Задать права:

```
function PersonRights(personId, rights);
```

или определить права

```
function PersonRights(personId);
```

Здесь:

- personId – (string) идентификатор персоны (название папки из FaceDB).
- rights – (int) права: 0 – обычное лицо, 1 – VIP-персона, 2 – хозяин.

В варианте с двумя параметрами функция задает права персоны и возвращает true в случае успеха или false в случае ошибки.

В варианте с одним параметром функция возврает текущие права персоны (int).

Функция PersonInfo

Функция устанавливает или определяет произвольный (пользовательский) атрибут персоны. Атрибут определяется по названию и хранит любой строковый параметр.

```
function PersonInfo(personId, attr, value);
```

или

```
function PersonInfo(personId, attr);
```

Здесь:

- personId – (string) идентификатор персоны (название папки из FaceDB).
- attr – (string) название атрибута.
- value – (string) значение атрибута.

В варианте с тремя параметрами функция задает значение атрибута attr для персоны personId и возвращает true в случае успеха или false в случае ошибки.

В варианте с двумя параметрами функция возврает текущее значение атрибута attr. Если атрибута не существует, возвращает пустую строку.

Функция GetPersonList

Функция получает полный список базы данных лиц.

```
function GetPersonList();
```

Функция возвращает массив строк (string) с идентификаторами персон базы данных лиц.

Пример.

```
// запросить перечень id лиц и вывести их имена
var list = GetPersonList();
for(var i=0; i < count(list); i++)
{
    var name = GetPersonName(list[i]); // получить имя
    print(name,"\n");
}
```

Функция AddNewPerson

Функция добавляет новую персону в базу данных.

```
function AddNewPerson(personId);
```

или

```
function AddNewPerson(personId, name);
```

или

```
function AddNewPerson(personId, name, ignoreExists);
```

Здесь:

- personId – (string) идентификатор новой персоны (название папки из FaceDB) или идентификатор персоны, для которой нужно добавить новый шаблон.
- name – (string) имя новой персоны или null, если не следудет добавлять имя.
- ignoreExists – (bool) признак игнорирования схожести с лицами, которые уже занесены в базу данных лиц. По умолчанию false.

Функция ожидает окончания процесса добавления новой персоны (обычно 4 секунды) и возвращает:

- true – персона успешно добавлена в базу данных.
- false – ошибка добавления в базу данных в основном по причине невозможности найти лицо на изображении.
- (string) идентификатор персоны, на которую похоже лицо, которое пытается добавиться в базу данных лиц (только в случае ignoreExists равно false).

Для добавления новой персоны необходимо в скрипте сгенерировать идентификатор для новой персоны. После чего вызвать данную функцию.

При вызове функции с параметром ignoreExists = false, функция пытается определить, на кого из уже существующих персон пожоже лицо в кадре. Если такое лицо в базе данных обнаруживается и одно не совпадает с personId, то функция завершается с ошибкой и возвращает строку с идентификатором лица, на которого лицо в кадре похоже. По сути, реализуется сценарий поведения «а это, случайно, не Вы?».

При вызове функции с параметром ignoreExists = true, функция игнорирует лиц, на которых персона похожа. По сути, реализуется сценарий «Это не Вы? Нет, не я».

Следует учитывать, что в процессе ожидания процесса завершения данной функции в IScript3 могут происходить другие события.

Функция LockAcquaintance

Функция устанавливает блокировку предложения о знакомстве (событие «*acquaintance») или получает статус этой блокировки.

Установить:

```
function LockAcquaintance(onOff);
```

или для получения статуса:

```
function LockAcquaintance();
```

Где:

onOff – (bool) признак блокировки.

Функция возвращает (bool) текущее состояние признака блокировки.

При реализации скрипта очень неудобно, если событие с предложением знакомства («* acquaintance») приходит в неудачный момент времени (например, пользователь сейчас фотографируется и ему не до знакомства).

Поэтому в ДинРобот-3 можно установить блокировку предложения о знакомстве, чтобы отложить возникновение события «* acquaintance» до возникновения блогоприятных для этого условий.

Установка признака предотвращает появление события «* acquaintance» пока блокировка не бдует снята. Если после снятия блокировки трекинг лица, с которым робот хочет познакомиться, все еще ведётся, то событие «* acquaintance» появляется сразу после снятия блокировки.

Удобно увязать блокировку с событием изменения страницы WEB-контента на экране: если страница «index.html», то блокировку можно снять, иначе наоборот установить.

По умолчанию блокировка не стоит.

При завершении работы скриптов блокировка снимается.

10. Функции работы с QR-кодами

Функция DetectQRCodes

Функция распознает QR-коды на видеоизображении с указанной камеры.

```
function DetectQRCodes(cameraIndex);
```

Где:

cameraIndex – (int) номер камеры робота.

Функция возвращает массив (array), каждый элемент которого является объектом (object) с информацией о распознанных QR-кодах. Каждый такой объект имеет следующие поля:

- text – (string) текст QR-кода. Кодировка ansi, хотя чаще всего в QR-кодах используется кодировка UTF-8. Но разработчик скрипта при необходимости может самостоятельно конвертировать текст в нужную кодировку с помощью соответствующий функций IScript3.
- angle – (float) угол (в градусах) горизонтального положения центра QR-кода относительно центра камеры (направление угла – слева направо).
- pitch – (float) угол места (в градусах) положения центра QR-кода относительно центра камеры (направление угла – снизу вверх).
- size – (float) размер QR-кода относительно ширины экрана. Принимает значения от 0 до 1.
- roll – (float) угол поворота (градусы) QR-кода на изображении (положительное направление – по часовой стрелке).

Если QR-кодов не обнаружено функция позвращает пустой массив.

При вызове без аргументов возращает null (ошибка).

Функцию следует вызывать многократно в течение всего времени, пока скрипту требуется QR-коды, с периодом не реже 3 секунд. Спустя 3 секунды после последнего вызова функции система распознавания QR-кодов «засыпает».

Следует понимать, что распознавание QR-кодов начиется отнюдь не с первого вызова данной функции. Функция лишь «будит» систему распознавания QR-кодов, которой требуется некоторое время на запуск. При этом само распознавание QR-кодов происходит асинхронно вне скрипта. Но если при очередном вызове данной функции в системе распозначания QR-кодов появится набор распознанных QR-кодов, то данная функция вернет массив с информацией об этих QR-кодах.

Система распознавания QR-кодов достаточно ресурсоемка, поэтому не следует ей злоупотреблять.

Пример:

```
var found = false; // признак обнаружения
for(var t=0; t < 6000; t+=100) // в течение 6000 мс...
{
  var qrCodes = DetectQRCodes(0); // получить QR-коды с
камеры 0

  // цикл по обнаруженным QR-кодам
  for(var i=0; i < count(qrCodes); i++)
  {
     if (qrCodes[i].text == "I love QR") // ищем текст
     {
```

```
         found = true; // Успех. Мы нашли, что искали!
         break; // выйти из цикла по i
      }
   }
   if (found) break; // если нашли, выйти из цикла по t
   Sleep(100); // пауза 100 мс
}

if (found)
   PlaySpeech("Вижу Кью Ар код");
else
   PlaySpeech("Кью Ар код не обнаружен");
```

11. Экранные сервисы

Функция SnapPhoto (сделать фото)

Функция делает фото с камеры.

```
function SnapPhoto(camera, photoFile, maskFile,
                                        smallFile, anylize);
```

или

```
function SnapPhoto(camera, photoFile, maskFile, smallFile);
```

или

```
function SnapPhoto(camera, photoFile, maskFile);
```

или

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

Где:

- camera – (int) номер камеры или (-1) – камера по умолчанию из config.txt.
- photoFile – (string) имя файла, куда сохранить результат. Если не указан полный путь, то путь задается относительно папки текущего экранного контента.
- maskFile – (string) имя файла с накладываемой рамкой для фото. Или пуста строка. По умолчанию нет. Если не указан полный путь, то путь задается относительно папки текущего экранного контента.
- smallFile – (string) имя файла, куда сохранить уменьшенную копию фото. Или пуста строка. Если не указан полный путь, то путь задается относительно папки текущего экранного контента.
- analyze – (bool) проводить ли анализ лиц на фото, по умолчанию false.

Функция возвращает (int):

- (-1) – ошибка.
- 0 – нет фото (технический код).
- 1 – есть фото.
- 2 – слишком темное фото (возвращается, только если analyze=true).
- 3 – на фото мужчина (возвращается, только если analyze=true).
- 4 – на фото мужчина улыбается (возвращается, только если analyze=true, улыбка определяется крайне плохо).
- 5 – на фото мужчина грустит (возвращается, только если analyze=true).
- 6 – на фото женщина (возвращается, только если analyze=true).
- 7 – на фото женщинаулыбается (возвращается, только если analyze=true, улыбка определяется крайне плохо).
- 8 – на фото женщина грустит (возвращается, только если analyze=true).
- 9 – на фото не видно лиц (возвращается, только если analyze=true).
- 10 – на фото все лица слева (возвращается, только если analyze=true).
- 11 – на фото все лица справа (возвращается, только если analyze=true).
- 12 – на фото все лица слишком высоко (возвращается, только если analyze=true).
- 13 – на фото все лица слишком низко (возвращается, только если analyze=true).
- 14 – на фото только один мужчины их их несколько (возвращается, только если analyze=true).

- 15 – на фото только один женщины и их несколько (возвращается, только если analyze=true).
- 16 – на фото несколько человек, среди них есть и мужчины, и женщины (возвращается, только если analyze=true).

Функция RoboArt (робо-художник)

Функция управляет функцией «робо-художник».

Для запуска, ожидания и получения результата:

```
function RoboArt(cameraIndex);
```

или

получение готового результата:

```
function RoboArt();
```

Где:

- cameraIndex – (int) номер камеры робота или (-1) – фотокамера по умолчанию (из config.txt).

Функция возвращает (int) код статуса или массив линий (array) в случае успеха.

Если функция завершилась с ошибкой, то возвращается (int) статус этой ошибки:

- (-1) – ошибка.
- 0 – функция ни разу не запускалась.
- 1 – робо-художник еще в процессе обработки (был запущен из экранного контента).

В случае успеха на выходе будет массив из объектов (object) со следующими полями:

- .x1 – (int) координата x1 линии.
- .y1 – (int) координата y1 линии.
- .x2 – (int) координата x2 линии.
- .y2 – (int) координата y2 линии.

Координаты линий расположены слева направо по X и сверху вниз по Y. Размеры изображения, относительно которого заданы координаты можно получить с помощью функций GetRoboArtImageWidth и GetRoboArtImageHeight.

Если функция запущена с параметром cameraIndex, то функция ожидает окончания работы робо-художника, прежде чем вернуть результат.

При вызове без параметра, функция получает статус состояния робо-художника или получает результат в виде массива линий. При этом сам робо-художник может быть запущен через экранный контент.

Пример:

```
var lines = RoboArt(0); // запустить и ожидать результат
if (lines===-1 || lines===0 || lines==1)
{
   print("Ошибка\n");
}
else
{
   X0=-40; Y0=60; Z0=60; // начало координат холста
   SCALE = 0.1; // масштаб пересчета картинки в холст

   for(var i=0; i < count(lines); i++)
   {
      var w = object();
      w.x = X0 + lines[i].x1 * SCALE;
```

```
      w.z = Z0 - lines[i].y1 * SCALE;
      w.ox = 0;
      w.oy = 0;
      w.oz = 0;

      w.y = Y0 - 2;
      ArmGoL(0, w); // передвинуть руку над точкой
      w.y = Y0;
      ArmGoL(0, w); // поставить фломастер на хост

     w.x = X0 + lines[i].x2 * SCALE;
     w.z = Z0 - lines[i].y2 * SCALE;
     ArmGoL(0, w); // нарисовать линию
     w.y = Y0 - 2;
     ArmGoL(0, w); // отодвинуть фломастер от холста
   }

   Gesture("Начало"); // возврат в исходную точку
}
```

Подразумевается следующий сценарий:

1. Экранный контент робота реализует интерфейс предварительного просмотра видеоизображения с камеры.
2. Экранный контент ожидает нажатия определенной кнопки на экране. По ее нажатию запускается обратный отсчет, по истечению которого запрос кадров видеоизображения прекращается и формируется запрос:
   photoservice?action=roboart:start&camera=0
   Ожидается ответ «OK».
3. После получения ответа «OK», экранный контент запрашивает картинку с предварительным результатом по адресу:
   photoservice?action=roboart:getimage
   Полученная картинка отображается пользователю.
4. Экранный контент ожидает нажатия кнопки подтверждения результата. На кнопке подтверждения надпись, вроде: «Мне нравится, рисуй». Рядом кнопка «Еще раз», которая возвращает сценарий экранного контента к пункту 1.
5. По нажатию кнопки подтверждения экранный контент отправляет в iScript3 какое-нибудь событие, например, «* DRAW». Это можно сделать через функцию javascript-функцию sendToIScript из «dynrobot.js».
6. Скрипт на iScript3 ожидает событие «* DRAW». Обработчик события в iScript3 запрашивает массив координат линий для рисования роботом через функцию RoboArt() без параметров.
7. Далее скрипт управления роботом заставляет руку робота вывести на бумаге изображение, состоящее из линий.

Однако, сценарий может быть и такой:

1. По определенной команде снять фото с помощью функции RoboArt(cameraIndex), указав в качестве cameraIndex номер камеры. Получить в результате набор линий для рисования.
2. Управлять роботом так, чтобы робот отрисовал на бумаге все эти линии.

Вариант 2 более прост в реализации, но лишен визульной части.

Функция GetRoboArtImageWidth

Функция возвращает ширину изображения, относительно которого заданы линии робохудожника (см. функцию RoboArt).

```
function GetRoboArtImageWidth();
```

Функция возвращает (int) ширину изображения робохудодника.

Функция GetRoboArtImageHeight

Функция возвращает выхоту изображения, относительно которого заданы линии робохудожника (см. функцию RoboArt).

```
function GetRoboArtImageHeight();
```

Функция возвращает (int) высоту изображения робохудодника.

Функция HasMotion (детектор движений)

Функция определяет, есть ли движения по всему видеоизображению или определенной зоне.

```
function HasMotion(analyzer, zone);
```

или

```
function HasMotion(analyzer);
```

Где:

analyzer – (int) номер видеоанализатора.

zone – (int) номер зоны видеоанализатора (0-10). Зона 0 – всё изображения. По умолчанию 0.

Возвращает (bool) true, если было зафиксировано движение. Признак остается активным в течение времени, заданного в config.txt, обычно 3000 мс.

Функцию следует вызывать с периодом не менее 3 сек, в противом случае видеоанализатор уходит в сон.

С помощью функции SetMotionEvent также можно установить событие в iScript3, которое будет автоматически вызываться при обнаружении движения.

Функция HasContrast (детектор объектов или контраста)

Функция определяет, есть ли контрастный объект на видеоизображении или в определенной зоне на видеоизображении.

```
function HasContrast(analyzer, zone);
```

или

```
function HasContrast(analyzer);
```

Где:

analyzer – (int) номер видеоанализатора.

zone – (int) номер зоны видеоанализатора (0-10). Зона 0 – всё изображения. По умолчанию 0.

Возвращает (bool) true, если было зафиксирован контрастный объект на видеоизображении.

Функцию следует вызывать с периодом не менее 3 сек, в противом случае видеоанализатор уходит в сон.

Функция GetAvgBrightness (детектор освещенности)

Функция определяет среднюю яркость кадра видеоизображения.

```
function GetAvgBrightness(analyzer);
```

Где:

analyzer – (int) номер видеоанализатора.

Возвращает (int) среднюю яркость кадра кадра видеоизображения в диапазоне от 0 до 255. Нормальная яркость кадра 128. Если изображение слишком темное яркость будет менее 60. Для очень яркого изображения (например, при внезапной вспышке) яркость будет значительно выше 180.

Функцию следует вызывать с периодом не менее 3 сек, в противом случае видеоанализатор уходит в сон.

С помощью функций SetMinBrightnessEvent и SetMaxBrightnessEvent можно установить события, которые будут автоматически формироваться при изменении яркости кадра.

Функция SetMotionEvent (задать событие на движения в кадре)

Функция устанавливает событие, которое будет формироваться при возникновении движения в кадре или в определенной зоне видеоанализатора.

```
function SetMotionEvent(analyzer, zone, event);
```

Где:

analyzer – (int) номер видеоанализатора.

zone – (int) номер зоны видеоанализатора (0-10). Зона 0 – весь кадр.

event – (string) Событие, например, «* myEvent». null или пустая строка отменяет событие.

Возвращает (bool) true, если событие успешно установлено (или отменено).

При подключении событий видеоанализатор включает детектор движении на постоянной основе (пока работают скрипты).

Пример:

```
frameset("events2", 100)
{
   // реакция на событие motion1
   frame("* motion1")
   {
       print("Движения в зоне 1\n");
   }

   // реакция на событие motion0
   frame("* motion0")
   {
       print("Движения в кадре\n");
   }
}

// установить событие детектора движения по видеоанализатору 0
// по зоне 0 (все изображение)
SetMotionEvent(0, 0, "* motion0");

// установить событие детектора движения по видеоанализатору 0
// по зоне 1
SetMotionEvent(0, 1, "* motion1");

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

Функция SetMinBrightnessEvent

Функция устанавливает событие, которое будет формироваться при падении яркости кадра ниже заданного в config.txt уровня (уровень также формируется из интерфейса настройки).

```
function SetMinBrightnessEvent(analyzer, event);
```

Где:

analyzer – (int) номер видеоанализатора.

event – (string) Событие, например, «* myEvent». null или пустая строка отменяет событие.

Возвращает (bool) true, если событие успешно установлено (или отменено).

При подключении событий видеоанализатор включает анализатор яркости на постоянной основе (пока работают скрипты).

Пример:

```
frameset("events2", 100)
{
   // реакция на событие lowBrightness
   frame("* lowBrightness")
   {
       PlaySpeech("Эй, кто выключил свет");
   }
}

// установить событие на падение минимальной яркости
SetMinBrightnessEvent(0, "* lowBrightness");

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

Функция SetMaxBrightnessEvent

Функция устанавливает событие, которое будет формироваться при поднятии яркости кадра выше заданного в config.txt уровня (уровень также формируется из интерфейса настройки).

```
function SetMaxBrightnessEvent(analyzer, event);
```

Где:

analyzer – (int) номер видеоанализатора.

event – (string) Событие, например, «* myEvent». null или пустая строка отменяет событие.

Возвращает (bool) true, если событие успешно установлено (или отменено).

При подключении событий видеоанализатор включает анализатор яркости на постоянной основе (пока работают скрипты).

Функция GetVideoAnalyzerGrid

Функция возвращает состояние клеток шахматного поля видеоанализатора.

```
function GetVideoAnalyzerGrid(analyzer);
```

Где:

analyzer – (int) номер видеоанализатора.

Возвращает массив (array) из целых чисел (int) состояния клеток. В массиве все ряды сложены в единый массив (как пиксели в изображении).

В каждом числе:

бит 0 – признак присутствия объекта в ячейке.

бит 1 – цвет (0 – черный, 1 – белый).

Таким образом, значение:

- 0 и 2 – отсутствие объекта в клетке.
- 1 – в клетке черная фишка (шашка).
- 3 – в клетке белая фишка (шашка).

В случае ошибки возвращает null.

12. 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) или в двухзначном формате, например 16 (подразумевается 2016 год).
- 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.

13. Временные переменные

Под временными переменными понимаются глобальные переменные, имеющие ограниченное время хранения.

Такие переменные удобно использовать для хранения сущностей речевого диалога. В процессе диалога с пользователем во фразах могут использоваться некоторые сущности, про которые роботу имеет смысл помнить в течение диалога, но по завершении диалога эти сущности становятся не актуальными.

Например, роботу могут сказать «Подними правую руку». Робот может запомнить, что последний раз речь шла про правую руку. Поэтому при последующей фразе «Опусти руку» робот может вспомнить про какую руку шла речь последний раз и опустить нужную. Однако, если фраза «Опусти руку» будет сказана через достаточно длительное время, роботу имеет смысл уже переспросить, о какой руке шла речь. Для реализации этого процесса удобно использовать временные переменные, в которых будет хранится информацию о том, какую руку роботу следует опускать.

Функция SetTempVar

Функция устанавливает или обновляет значение временной переменной.

```
function SetTempVar(varName, value, liveTime);
```

или

```
function SetTempVar(varName, value);
```

Где:

varName – (string) название переменной.

value – (любой тип данных) значение.

liveTime – (int) время хранения переменной (мс), по умолчанию 30000.

Функция ничего не возвращает.

Если временной переменной не существовало, то создается новая переменная. Если переменная существовала, то переменная переприсваивается, а счетчик времени её хранения сбрасывается.

Функция GetTempVar

Функция устанавливает или обновляет значение временной переменной.

```
function GetTempVar(varName, defValue);
```

или

```
function GetTempVar(varName);
```

Где:

varName – (string) название переменной.

defValue – (любой тип данных) значение по умолчанию. По умолчанию null.

Функция возвращает значение временной переменной или значение по умолчанию, если временная переменная не задана или время ее хранения истекло.

14. 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
  ]
}
```

15. Управление скриптами и системные функции

Функция sync

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

```
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 exit

Функция завершает работу скриптов, программы ДинРобот или системы Windows

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

Где:

type – тип завершения работы:

0 – завершить работу скриптов.

1 – завершить работу ДинРобот.

2 – завершить работу Windows.

Пример:

```
frameset("dialogs",1)
{
   frame("Робот тебе необходимо прямо сейчас завершить работу системы")
   {
      PlaySpeech("Хорошо, выключаюсь");
      while(PlaySpeech()) sync(); // подождать пока робот договорит
      exit(2);                    // завершить работу windows
   }
}
SpeechRecognizer(true);
while(1) sync();
```

Функция SetTimer

Функция запускает таймер.

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

Где:

code – (string) строка на IScript3 кода, выполняющегося по таймеру.

interval – интервал таймера (мс).

Функция возвращает идентификатор таймера (int).

Следует отметить, что вызов таймера возможен только во время выполнения функций, ожидающих результат или в процессе вызова функции sync().

Код, выполняющийся по таймеру, рекомендуется свести к вызову какой-либо функции.

```
n = 0;

// функция, вызываемая по таймеру
function OnTimer()
{
     print("tick\n");
     n++;
     // уничтожить таймер через 10 тиков таймера
     if (n>=10) KillTimer(timerID); }

// установить таймер
timerID = SetTimer("OnTimer()", 100);
while(1) sync();
```

Функция KillTimer/ClearTimer

Функция удаляет таймер.

```
function KillTimer(timerID);
```

Где:

timerID – (int) идентификатор таймера, полученный с помощью функции SetTimer(…).

Возвращает true, если таймер найден и уничтожен.

Функция Restart

Функция перезапускает скрипт.

```
function Restart(scriptFileName);
```

или

```
function Restart();
```

Где:

scriptFileName – название файла скрипта из папки scripts, или с указанием относительного пути, относительно папки scripts, или с указанием полного пути к файлу.

Если параметр не указан, то перезапускается скрипт, указанный в качестве файла скрипта в файле config.txt.

16. Функции работы с файлами

Функция fopen

Функция открывает файл на диске робота.

```
function fopen(fileName, mode);
```

Где:

fileName – (string) имя файла относительно папки ДинРобот.

mode – (string) строка доступа к файлу (см. fopen для C++):

w – открыть на запись (перезапись).

r – открыть на чтение.

a – открыть на запись в конец.

r+ – открыть на чтение и запись.

Дополнительные режимы:

b –флаг определяющий, что файл бинарный, иначе текстовый.

t – признак текстового файла (лучше не использовать).

Функция возвращает (int64) дескриптор открытого файла, используемый другими функциями работы с файлами, или 0, в случае ошибки.

Открытый файл следует закрывать с помощью функции fclose.

Пример:

```
f = fopen("HTTP/my.txt", "wb"); // открыть файл на запись
if (f)  // если файл открылся
{
     fwrite(f, "Привет мир!\r\n");  // записать в файл
текст
     fclose(f);  // закрыть файл
}
```

Функция fwrite или fputs

Функции fwrite и fputs являются синонимами. Функция записывает данные в файл.

```
function fwrite(f, data);
```

Где:

f – (int64) дескриптор файла, открытого с помощью функции fopen.

data – данные для записи. В зависимости от типа данных функции совершают разные действия. Если тип данных:

bool – записать один байт 1 или 0.

int – записать 32-битное слово (LE).

float – записать 64-битное значение FLOAT (LE).

string – записать строку (кодировка Win1251).

binary – записать бинарный блок данных.

В случае успеха функция возвращает (bool) true, иначе false.

Или

```
function fwrite(f, data, type);
```

Где:

f – (int64) дескриптор файла, открытого с помощью функции fopen.

data – данные для записи. В зависимости от типа данных функции совершают разные действия.

type – (string) формат данных, аналогичный функции set_bin_data.

Функция возвращает true, если данные записаны, и false в случае ошибки.

Функция fgets

Функция читает строку из файла.

```
function fgets(f);
```

Где:

f – (int64) дескриптор файла, открытого с помощью функции fopen.

Функция возвращает прочитанную строку (string), или null, в случае ошибки

Функция принудительно очищает строку от всех символов переноса строки.

Длина строки ограничена 32кб.

Функция fread

Функция читает из файла бинарные данные.

```
function fread(f, byteCount);
```

Где:

f – (int64) дескриптор файла, открытого с помощью функции fopen.

byteCount – количество байт для чтения.

Функция возвращает прочитанные данные (binary), или null, в случае ошибки.

Или

```
function fread(f, type);
```

Где:

f – (int64) дескриптор файла, открытого с помощью функции fopen.

type – (string) формат данных, аналогично функции get_bin_data.

Функция возвращает данные, аналогичные функции get_bin_data.

Пример:

```
f = fopen("my.jpg","rb"); // открыть файл
if (f) // если файл открылся
{
   bin = fread(f, 4);  // прочитать 4 байта из файла
   if (bin)
   {
      ... // функции работы с binary
   }
   fclose(f); // закрыть файл
}
```

Или

```
f = fopen("my.wav","rb"); // открыть файл
if (f) // если файл открылся
{
   var s = fread(f, "int32"); // прочитать int32
   ...
   fclose(f); // закрыть файл
```

```
}
```

Функция fseek

Функция перемещает указатель чтения/записи в текущем файле.

```
function fseek(f, pos, relative);
```

Где:

f – (int64) дескриптор файла, открытого с помощью функции fopen.

pos – (int) позиция указателя в байтах, в зависимости от параметра relative.

relative – (int) относительно чего задан pos:

- 0 – относительно начала файла.
- 1 – относительно текущей позиции файле.
- 2 – относительно конца файла (при этом pos должен быть меньше или равен нулю).

Функция возвращает (int) позицию чтения/записи файле после применения функции fseek или -1 в случае ошибки.

Примеры:

```
fileSize = fseek(f, 0, 2);  // определить размер файла
pos = fseek(f, 0, 1); // текущее позиция указателя
fseek(f, 12, 0); // сместиться на 12 байт файла
```

Функция применяется только для файлов, размером менее 2 Гб.

Функция feof

Функция возвращает true, если достингут конец файла.

```
function feof(f);
```

Где:

f – (int64) дескриптор файла, открытого с помощью функции fopen.

Функция возвращает (bool), если достигнут конец файла, или false в любом другом случае (в т.ч. в случае ошибки).

Примеры:

```
f = fopen("my.txt","rb"); // открыть файл на чтение
if (f) // если файл открылся
{
     while(!feof(f))  // пока не достингут конец файла
     {
         line = fgets(f);   // читать строку
         printf(line,"\n");   // выводить строку на экран
     }
     fclose(f);  // закрыть файл
}
```

Функция fclose

Функция закрывает файл, открытый ранее функцией fopen.

```
function fclose(f);
```

Где:

f – (int64) дескриптор файла, открытого с помощью функции fopen.

Функция всегда возвращает null

Функция scandir

Функция получает оглавление указанной директории, и возвращает массив записей.

```
function scandir(path);
```

Где:

path – (string) путь к директории

Функция всегда возвращает массив из объектов, каждый из которых содержит поля:

- .fileName – название файла.
- .isDir – является ли директорий.

Например:

```
var dir = scandir("../xml"); // получить оглавление
for(var i=0; i < count(dir); i++)
{
   if (!dir[i].isDir)  // если это не директория
     print(dir[i].fileName,"\n"); // вывести название файла
}
```

17. Функции работы COM-портами

Функция CommOpen

Функция открывает указанный COM-порт на указанной скорости обмена.

```
function CommOpen(portName, baudrate);
```

или

```
function CommOpen(portName);
```

Где:

portName – (string) название COM-порта (например: «COM3» для Windows). Под Windows порты с номерами 10 и более следует указывать в формате «\\.\COM10». Для Linux следует указывать название TTY-устройства, например: «/dev/ttyS0».

baudrate – (int) скорость обмена данными в бодах. По умолчанию 115200.

Функция возвращает (int) дескриптор COM-порта или (-1) в случае ошибки.

Порт всегда открывается в режиме 8 бит, 1 стоповый бит, без контроля четности или нечетности, что соответствует режимам работы 99.9% всех возможных устройств, работающих по последовательному интерфейсу, включая все USB-устройства.

Следует отметить, что ДинРобот-3 сам производит асинхронное чтение и запись по указанному порту, а также асинхронно переоткрывает порт, если вдруг он закрылся или не был сразу же открыт функцией CommOpen. При необходимости ДинРобот-3 отсчитывает несколько секунд перед повторным открытием порта (чтобы не мучать систему постоянными попытками открыть несуществующий порт).

При этом все даннные, которые были отправлены по данному COM-порту в момент его аварийного закрытия, будут отправлены сразу же после его повторного открытия.

Чтобы закрыть порт используйте функцию CommClose.

Все порты, которые остались открытыми скриптом, закрываются по его завершению.

Пример:

```
var comm = CommOpen("\\\\.\\COM10", 115200); // открыть
COM10
CommWrite(comm,"Hello world"); // отправить «Hello world»
CommClose(comm); // закрыть порт
```

Функция CommClose

Функция закрывает COM-порт.

```
function CommClose(handle);
```

Где:

handle – (int) дескриптор COM-порта, ранее открытого с помощью CommOpen.

Функция ничего не возвращает.

При закрытии порта отправляются все ранее неотправленные данные, если аппаратное оборудование готово принять такой объем данных за один раз, при условии, что сам порт был при этом открыт (а не находился в состоянии аварийного закрытия).

Функция IsCommOpen

Функция проверяет, находится ли указанный COM-порт в открытом состоянии.

```
function IsCommOpen(handle);
```

Где:

handle – (int) дескриптор COM-порта, ранее открытого с помощью CommOpen.

Фунция возвращает (bool) признак отрытого состояния порта. В случае ошибки (например, handle не является дескриптором порта или указанный порт уже был закрыт), функция возвращает null.

Следует пояснить, что ДинРобот-3 обеспечивает самостоятельный контроль за открытием или аварийным закрытием COM-порта и переоткрывает его при необходимости. Поэтому в некоторые моменты времени порт может находиться в закрытом состоянии, что и возвразает данная функция.

Пользователь может отслеживать состояние порта прежде чем, например, записывать в него новые данные. Хотя все данные, которые были записаны в порт в состоянии, когда порт закрыт, будут отправлены сразу же при его повтором открытии.

Функция CommWrite

Функция отправляет данные по COM-порту.

```
function CommWrite(handle, data, type);
```

или

```
function CommWrite(handle, data);
```

Где:

handle – (int) дескриптор COM-порта, ранее открытого с помощью CommOpen.

data – отправляемые данные (см. комментарий).

type – (string) тип данных (значения регистровонезавимы):

- «bool» – data считать типом bool. Отправляется 1 байт со значением 0 или 1.
- «int8» – data считать типом int. Отправляется 8 бит (знаковые).
- «uint8» – data считать типом int. Отправляется 8 бит (беззнаковые).
- «int16» – data считать типом int. Отправляется 16 бит (знаковые, LittleEndian).
- «int16be» – data считать типом int. Отправляется 16 бит (знаковые, BigEndian).
- «uint16» – data считать типом int. Отправляется 16 бит (беззнаковые, LittleEndian).
- «uint16be» – data считать типом int. Отправляется 16 бит (беззнаковые, BigEndian).
- «int32» – data считать типом int. Отправляется 32 бита (знаковые, LittleEndian).
- «int32be» – data считать типом int. Отправляется 32 бита (знаковые, BigEndian).
- «uint32» – data считать типом int. Отправляется 32 бита (беззнаковые, LittleEndian).

«uint32be» – data считать типом int. Отправляется 32 бита (беззнаковые, BigEndian).

«int64» – data считать типом int. Отправляется 64 бита (знаковые, LittleEndian).

«int64be» – data считать типом int. Отправляется 64 бита (знаковые, BigEndian).

«uint64» – data считать типом int. Отправляется 64 бита (беззнаковые, LittleEndian).

«uint64be» – data считать типом int. Отправляется 64 бита (беззнаковые, BigEndian).

«float» или «float32» или «F32» – data считать типом float. Отправляется 32 бита в формате float IEEE. Всегда LittleEndian.

«double» или «float64» или «F64» – data считать типом float. Отправляется 64 бита в формате двойной точности IEEE. Всегда LittleEndian.

«string» или «text» – data считать типом string. Отправляется строка в кодировке ANSI. При этом никаких признаков начала и конца строки не приписываются.

«hex2» – data считать типом int. Отправляется 2 байта, представляющие собой целое число data, переведенное в формат двухзначного hex. Например: если data = 15, будет отправлено «0F».

«hex4» – data считать типом int. Отправляется 4 байта, представляющие собой целое число data, переведенное в формат четырехзначного hex. Например: если data = 257, будет отправлено «0101».

«hex6» – data считать типом int. Отправляется 6 байта, представляющие собой целое число data, переведенное в формат шестизначного hex.

«hex8» – data считать типом int. Отправляется 8 байта, представляющие собой целое число data, переведенное в формат восьмизначного hex.

«strbool» – data считать типом bool, Отправляется 1 байт, значение которого «1» или «0» (в текстовом представлении) в зависимости от значения data.

Фунция ничего не возвращает.

Если не указан type, то значение data интерпретируется в зависимости от типа данных самого значения data. Например, если у data тип данных int, то будут отправлены данные в формате «int32».

Если не указан type, а data имеет тип данных binary, то отправляются данные из блока бинарных данных.

Если не указан type, а data имеет тип данных array, то отправляются каждый элемент массива, интерпретируя данные этих элементов в зависимости от типа их данных.

Если не указан type, а data имеет тип данных object, то отправляются каждое поле этого объекта, интерпретируя данные этих полей в зависимости от типа их данных.

Следует отметить, что функция лишь записывает данные в выходной буфер COM-порта. Отправка данных производится только в периоды синхронизации (в выполнении функции sync, sleep или других функций, реализующих внутренний цикл ожидания скрипта). При этом внутренний выходной буфер может быть любой длины, а если оборудование COM-порта не в состоянии забрать такой большой буфер на отправку, то ДинРобот-3 сам производит асинхронную отправку этих данных.

Записать данные в COM-порт можно даже в случае, если в данный момент порт закрыт, данные будут отправлены, как только порт откроется.

При закрытии порта также производится отправка неотправленных данных (см. CommClose).

Пример:

```
var comm = CommOpen("COM3", 115200); // открыть COM3

CommWrite(comm, 0xCC, "uint8");  // отправить 0xCC как
uint8
CommWrite(comm, 0x01, "uint8");  // отправить 0x01 как
uint8
CommWrite(comm, 2500, "int16");  // отправить 2600 как
int16
sync(); // синхронизация и отправка данных

var arr = array(1,2,3,0.4,"hello"); // данные на отправку
CommWrite(comm, arr); // отправить каждый элемент массива

var obj = object();
obj.x = 0.34;
obj.y = 0.123
CommWrite(comm, obj); // отправить каждое поле объекта

var bin = binary();  // создать бинарные данные и заполнить
set bin size(bin, 4);
set_bin_byte(bin, 0, 0xCC);
set_bin_byte(bin, 1, 0x01);
set_bin_byte(bin, 2, 0x20);
set_bin_byte(bin, 3, 0x3E);

CommWrite(comm, bin); // отправить бинарные данные

Sleep(100); // задержка 100 мс. Точно хватит на отправку

CommClose(comm); // закрыть порт
```

Функция CommWriteBE

Функция отправляет данные по COM-порту в порядке байт BigEndian (старшим байтом вперед).

```
function CommWrite(handle, data);
```

Где:

handle – (int) дескриптор COM-порта, ранее открытого с помощью CommOpen.

data – отправляемые данные (см. комментарий).

Фунция ничего не возвращает.

Значение data интерпретируется в зависимости от типа данных самого значения data, интерпретируя многобайтные целые числа в формате BigEndian. Например, если у data тип данных int, то будут отправлены данные в формате «int32be». Типы данных float и string передаются в прямом порядке байт (LittleEndian).

Если data имеет тип данных binary, то отправляются данные из блока бинарных данных.

Если data имеет тип данных array, то отправляются каждый элемент массива, интерпретируя данные этих элементов в зависимости от типа их данных, интерпретируя многобайтовые структуры в порядке байт BigEndian.

Если data имеет тип данных object, то отправляются каждое поле этого объекта, интерпретируя данные этих полей в зависимости от типа их данных, интерпретируя многобайтовые структуры в порядке байт BigEndian.

Следует отметить, что функция лишь записывает данные в выходной буфер COM-порта. Отправка данных производится только в периоды синхронизации (в выполнении функции sync, sleep или других функций, реализующих внутренний цикл ожидания скрипта). При этом внутренний выходной буфер может быть любой длины, а если оборудование COM-порта не в состоянии забрать такой большой буфер на отправку, то ДинРобот-3 сам производит асинхронную отправку этих данных.

Записать данные в COM-порт можно даже в случае, если в данный момент порт закрыт, данные будут отправлены, как только порт откроется.

При закрытии порта также производится отправка неотправленных данных (см. CommClose).

Пример:

```
var comm = CommOpen("COM3", 115200); // открыть COM3

CommWrite(comm, 0xCC, "uint8");  // отправить 0xCC как
uint8
CommWrite(comm, 0x01, "uint8");  // отправить 0x01 как
uint8
CommWriteBE(comm, 2500, "int16");  // отправить 2600 как
int16
sync(); // синхронизация и отправка данных
CommClose(comm); // закрыть порт
```

Функция CommRead

Функция читает данные из COM-порта.

```
function CommRead(handle, type);
```

или

```
function CommRead(handle, len);
```

или

```
function CommRead(handle);
```

Где:

handle – (int) дескриптор COM-порта, ранее открытого с помощью CommOpen.

len – (int) количество считываемых байт (по умолчанию все имеющиеся на момент вызова функции данные).

type – (string) тип данных (значения регистровонезавимы):

«bool» – Читатет 1 байт со значением 0 или 1. Любое значение, отличное от 0, считается за 1. Функция возвращает его значение в виде типа данных bool

«int8» – считать 1 знаковый байт. Функция возвращает его значение в виде типа данных int.

«uint8» – считать 1 беззнаковый байт. Функция возвращает его значение в виде типа данных int.

«int16» – считать 2 байта, интепретируя их, как знаковое двухбайтное число (LittleEndian). Функция возвращает его значение в виде типа данных int.

«int16be» – считать 2 байта, интепретируя их, как знаковое двухбайтное число (BigEndian). Функция возвращает его значение в виде типа данных int.

«uint16» – считать 2 байта, интепретируя их, как беззнаковое двухбайтное число (LittleEndian). Функция возвращает его значение в виде типа данных int.

«uint16be»– считать 2 байта, интепретируя их, как беззнаковое двухбайтное число (BigEndian). Функция возвращает его значение в виде типа данных int.

«int32» – считать 4 байта, интепретируя их, как знаковое 32-битное число (LittleEndian). Функция возвращает его значение в виде типа данных int.

«int32be» – считать 4 байта, интепретируя их, как знаковое 32-битное число (BigEndian). Функция возвращает его значение в виде типа данных int.

«uint32» – считать 4 байта, интепретируя их, как беззнаковое 32-битное число (LittleEndian). Функция возвращает его значение в виде типа данных int64.

«uint32be»– считать 4 байта, интепретируя их, как беззнаковое 32-битное число (BigEndian). Функция возвращает его значение в виде типа данных int64.

«int64» или «uint64» – считать 8 байта, интепретируя их, как знаковое 64-битное число (LittleEndian). Функция возвращает его значение в виде типа данных int64.

«int64be» – считать 8 байта, интепретируя их, как знаковое 64-битное число (BigEndian). Функция возвращает его значение в виде типа данных int64.

«float» или «float32» или «F32» – считать 4 байта, интепретируя их, как число в формате float (32-бита IEEE, LittleEndian). Функция возвращает его значение в виде типа данных float.

«double» или «float64» или «F64» – считать 4 байта, интепретируя их, как число в формате double (64-бита IEEE, LittleEndian). Функция возвращает его значение в виде типа данных float.

«string*XXX*» или «text*XXX*» – тип даных string. При этом XXX – должна указывать на считываемое количество символов. Например: «string28». Кодировке ANSI.

«hex2» – считать 2 байта, интерпретируя их, как строка, в которой записано число в формате HEX (беззнаковое). Функция возвращает его значение в виде типа данных int.

«hex4» – считать 4 байта, интерпретируя их, как строка, в которой записано число в формате HEX (беззнаковое). Функция возвращает его значение в виде типа данных int.

«hex6» – считать 6 байта, интерпретируя их, как строка, в которой записано число в формате HEX (беззнаковое). Функция возвращает его значение в виде типа данных int.

«hex8» – считать 8 байта, интерпретируя их, как строка, в которой записано число в формате HEX (беззнаковое). Функция возвращает его значение в виде типа данных int.

«strbool» – считать 1 байт, интерпретируя его, как строка, в которой записан «0» или «1» (тестовое представление). Функция возвращает его значение в виде типа данных bool.

Если type не указан, то функция читает из входого буфера COM-порта len байт (а если не указано len, то все имеющиеся данные на момент вызова функции), и возвращает их в виде типа данных binary.

Если на момент вызова функции во входном буфере COM-порта не оказалось тремуемого числа байт, то функция возвращает null.

Если функция вызывается вне блока CommBeginRead/CommEndRead, то функция изымает из входного буфера прочитанные байты, иначе данные считаются непрочитанными, и остаются во входном буфере COM-порта.

Удобно использовать функцию в блоке CommBeginRead/CommEndRead, это позволяет реализовать чтение пакетов данных.

Например, пусть устройство на устройство нужно отправить PING с помощью пакета «0xCC 0x01», а устройство шлет пакет данных:

Смещение от начала пакета	Тип данных	Описание
0	uint8	START_BYTE. Стартовый байт, всегда 0xCC
1	uint8	ans. Код ответа устройства: 1 – PONG, 2 – показания
2	uint16	sensor. Показания датчика (только если ans=2).

```
START_BYTE = 0xCC; // стартовый байт
// коды команд
CMD PING = 1;


// коды ответов устройства
ANS_PONG = 1;
```

```
ANS SENSOR = 2;

function ReadData()
{
    // цикл по всем принятым пакетам
    while(true)
    {
      CommBeginRead(comm); // начать чтение пакета COM-
порта
      var byte;

      // читаем COM-порт, пока есть данные. Останавливаемся
      // если считаем START_BYTE
      do
      {
         byte = CommRead(comm,"uint8");
      } while(byte!==null && byte!= START_BYTE);

      if (byte===null) return; // Пока нет стартового байта


      // нашли START_BYTE.
      // читаем код ответа устройства
      var ans = CommRead(comm, "uint8");
      if (ans===null) return; // пока не все данные пришли

      if (ans == ANS_PONG) // ответ PONG
      {
          print("PONG от устройства\n");
          CommEndRead();  // конец чтения пакета
      }
      else
      if (ans == ANS_SENSOR) // данные сенсора
      {
          var sensor = CommRead(comm, "uint16");
          if (sensor === null) // пока не все данные пришли
          print("Sensor = ", sensor, "\n");
          CommEndRead();// конец чтения пакета
      }
      else
      {
         // неизвестный ответ устройства - удаляем его
         CommEndRead();
      }
   }
}

comm = CommOpen("COM3", 115200); // открыть COM3

// отправить устройству PING
CommWrite(comm, START BYTE);
CommWrite(comm, CMD_PING);
```

```
// главный цикл
while(true)
{
   ReadData(); // чтение данных
   sync(); // синхронизация и отправка данных
}
```

Функция CommBeginRead

Функция начинает чтение пакета данных из COM-порта.

```
function CommRead(handle);
```

где:

handle – (int) дескриптор COM-порта, ранее открытого с помощью CommOpen.

Функция ничего не возвращает.

Функция переводит COM-порт в режим чтения пакета данных. Указатель чтения при этом переводится на начало непрочитанных данных. В этом режиме все почитанные данные из COM-порта с помощью функции CommRead не удаляются из входного буфера, а остаются до момента вызова функции CommEndRead.

Таким образом, пользователь может начать читать данные из COM-порта функций CommBeginRead. Далее начать читать данные функцией CommRead, и, обнаружив, что требуемые входные данные еще не полностью пришли, остановится до следующей итерации чтения. А после того, как был читан пакет целиком, может вызвать функцию CommEndRead, чтобы удалить из входного буфера COM-порта прочитанный пакет данных.

Пример см. функцию CommRead.

Функция CommEndRead

Функция завершает чтение пакета данных из COM-порта.

```
function CommRead(handle);
```

где:

handle – (int) дескриптор COM-порта, ранее открытого с помощью CommOpen.

Функция ничего не возвращает.

Функция проверяет, была ли ранее вызвана фунция CommBeginRead для указанного COM-порта. Если была, то удаляет из входного буфера COM-порта все прочитанные байты. После чего выключает режим пакетного четния.

Пример см. функцию CommRead.

18. Функции работы с внутренней базой данных параметров

Внутренняя база данных параметров является таблицу параметров, созданную по принципу:

параметр1 = значение1
параметр2 = значение2
параметр3 = значение3
...

Внутренняя база данных позволяет сохранять какие-либо параметры между сеансами запуска программы.

Функция SetInnerDB

Функция записывает данные во внутреннюю базу данных.

```
function SetInnerDB(param, value);
```

Где:
param – (string) название параметра.
value – значение. Значение всегда интерпретируется как строка.
Функция возвращает true, если запись удалась.

Функция GetInnerDB

Функция запрашивает строку из внутренней базы данных.

```
function GetInnerDB(param);
```

или

```
function GetInnerDB(param, def);
```

Где:
param – (string) название параметра.
def – (string) значение по умолчанию.

Функция возвращает строку (string) значения параметра. Если параметр в базе данных отсутствует, то возвращает значение параметра def. Если параметр def не указан, то возвращает пустую строку.

Пример:

```
coffeeName = GetInnerDB("coffeeName", "Нескафе");
...
// после изменения coffeeName
SetInnerDB("coffeeName", coffeeName);
```

Функция GetInnerDBInt

Функция запрашивает целое число из внутренней базы данных.

```
function GetInnerDBInt(param);
```

или

```
function GetInnerDBInt(param, def);
```

Где:
param – (string) название параметра.
def – (int) значение по умолчанию.

Функция возвращает целое число из указанного параметра. Если параметр в базе данных отсутствует, то возвращает значение параметра def. Если параметр def не указан, то возвращает 0.

Функция GetInnerDBInt64

Функция запрашивает целое число в формате int64 из внутренней базы данных.

```
function GetInnerDBInt64(param);
```

или

```
function GetInnerDBInt64(param, def);
```

Где:
param – (string) название параметра.
def – (int) значение по умолчанию.

Функция возвращает целое число из указанного параметра. Если параметр в базе данных отсутствует, то возвращает значение параметра def. Если параметр def не указан, то возвращает 0.

Функция GetInnerDBBool

Функция запрашивает параметр типа bool из внутренней базы данных.

```
function GetInnerDBBool(param);
```

или

```
function GetInnerDBBool(param, def);
```

Где:
param – (string) название параметра.
def – (bool) значение по умолчанию.

Функция возвращает параметр типа bool из указанного параметра. Если параметр в базе данных отсутствует, то возвращает значение параметра def. Если параметр def не указан, то возвращает false.

Функция GetInnerDBFloat

Функция запрашивает вещественное число из внутренней базы данных.

```
function GetInnerDBFloat(param);
```

или

```
function GetInnerDBFloat(param, def);
```

Где:
param – (string) название параметра.
def – (float) значение по умолчанию.

Функция возвращает вещественное число из указанного параметра. Если параметр в базе данных отсутствует, то возвращает значение параметра def. Если параметр def не указан, то возвращает 0.0.

Функция DeleteInnerDB

Функция удаляет параметр из внутренней базы данных.

```
function DeleteInnerDB(param);
```

Где:
param – (string) название параметра.

Возвращает true, если удаление удалось.

Функция DeleteAllInnerDB

Функция удаляет все параметры из внутренней базы данных, начинающихся с указанной строки

```
function DeleteAllInnerDB(startWith);
```

или

```
function DeleteAllInnerDB();
```

Где:
startWith– (string) начало названия параметра. Если параметр не указан или представляет собой пустую строку, то производится удаление всех параметров из внутренней базы данных.

Возвращает true, если удаление удалось.

Пример:

```
SetInnerDB("kitchen_milk", true);
SetInnerDB("kitchen coffee", true);
SetInnerDB("kitchen_tea", true);

SetInnerDB("room_box", true);
...


// удалить все параметры, начинающиеся на «kitchen_»
DeleteAllInnerDB("kitchen_");

// в базе останется только параметр «room_box»
```

19. Функции работы с базой данных объектов

База данных объектов представляет собой одну или несколько таблиц любых поименованных данных. Названия каждой такой записи данных представляет собой фразу на естественном языке, автоматически транслируемую во фремы фреймообразной структуры базы знаний. Название фреймсета этих фреймов совпадает с названием таблицы такой базы данных.

Например, требуется сделать базу данных объектов, с которыми может взаимодействовать робот. Для примера пусть это будут «Яблоко», «Груша», «Персик». С помощью функции «CreateSubjDB("Fructs")» создается таблица «Fructs» и автоматически появляется фреймсет «Fructs». При добавлении, изменении или удалении из таблицы «Fructs» записей, производится автоматическое переформирование фреймсета «Fructs» так, что в нем будут появляться фреймы с названиями существующих объектов во всех возможных падедах. При активации этих фреймов будет возвращаться значения, соответствующие записям в базе данных.

Такой подход избавляет пользователя от необходимости самостоятельного формирования фреймов, соответствующих полям базы данных. Зато он может писать фреймы, типа «ПРИНЕСИ <что:Fructs>». Функция ValueOf("что") в этом случае будет возвращать значение, хранимое в базе данных, соответствующее данному фрукту.

Таблицы базы данных храняться в отдельных файлах и при создании таблицы система их загружает из указанных файлов. Путь к таким таблицам задается с помощью функции SetSubjDBPath.

По аналогии с базой данных объектов с помощью функции «CreatePersonFrameset("Persons")» создается фреймсет, транслирующий в себя базу лиц, а точнее имен соответствующих персон. При добавлении или изменении персон производится автоматическое перестроение этого фреймсета. Таким образом, пользователь может писать фреймы типа «Подойди к <кому:Persons>. Функция ValueOf("кому") в этом случае будет возвращать идентификатор персоны из базы данных лиц.

Функция CreateSubjDB или CreateSubjectDB

Функции CreateSubjDB и CreateSubjectDB являются синонимами.

Функция создает таблицу в базе данных объектов.

```
function CreateSubjDB(table);
```

или

```
function CreateSubjectDB(table);
```

Здесь:

table – (string) название таблицы. Желательно латинскими буквами.

Функция возвращает true в случае успеха.

Созданная таблица автоматически создает одноименный фреймсет, в который будут транслироваться названия данных, помещаемых в таблицу базы данных объектов. Кроме этого таблица ассоциируется с одноименным файлом в папке, указанной через SetSubjDBPath.

Если ассоциированный файл существует на момент вызова CreateSubDB, то даные из ассоциированного файла автоматически загружаются в таблицу. При изменении данных в таблице данные будут записаны в ассоциированный файл (спустя 4 секунды после последнего изменения).

При внесении ручных изменений в ассоциированный файл (например, с помощью программы «Блокнот»), файл перечитывается (не позднее 5 секунд после изменений).

Пример:

```
SetSubjDBPath("myDB"); // указать папку для хранения данных

CreateSubjDB("Fructs");  // создать таблицу (и считать ее)
SetSubjDB("Fructs", "Яблоко", 1); // Добавить «яблоко» id:1
SetSubjDB("Fructs", "Груша", 2); // Добавить «яблоко» id:2
SetSubjDB("Fructs", "Персик", 3); // Добавить «яблоко» id:3

// пробуем использовать
frameset("myCommands",1)
{
    frame("Принеси <что:Fructs>")
    {
        var objID = ValueOf("что"); // получить ID объекта
        ...
    }
}
SpeechRecognizer(true); // включить распознавание речи
while(true) sync();
```

Функция SetSubjDB

Функция устанавливает значение в таблице базы данных объектов (см. выше).

```
function SetSubjDB(table, name, value, updateFrameset);
```

или

```
function SetSubjDB(table, name, value);
```

Где:

table – (string) название таблицы, ранее созданной в базе данных объектов с помощью функции CreateSubjDB.

name – (string) название объекта на естественном человеческом языке. На основе склонения этого имени по падежам будет создан фрейм во фреймсете, связанным с таблицей базы данных.

value – любые данные (включая массивы или объекты) связанные с объектом базы данных.

updateFrameset – (bool) признак необходимости обновления фреймов во фреймсете. При групповом добалении данных имеет смысл ставить false. По умолчанию true.

Функция ничего не возвращает.

При установке значения, если одноименный объект уже был в таблице (без учета регистра букв), то функция заменяет его значение. Иначе создает новое поле.

При внесении изменений таблица сохраняется в файл не позднее 4 секунд после последнего внесения изменений.

Функция GetSubjDB

Функция получает значение из таблицы базы данных объектов (см. выше).

```
function GetSubjDB(table, name);
```

Где:

table – (string) название таблицы, ранее созданной в базе данных объектов с помощью функции CreateSubjDB.

name – (string) название объекта на естественном человеческом языке. На основе склонения этого имени по падежам будет создан фрейм во фреймсете, связанным с таблицей базы данных.

Функция возвращает данные, хранимые в указанном объекте или null в случае ошибки или отсутствия этих данных.

Функция DelSubjDB

Функция удаляет значение из таблицы базы данных объектов или всю таблицу.

Для удаления записи из таблицы:

```
function DelSubjDB(table, name, updateFrameset);
```

или

```
function DelSubjDB(table, name);
```

Для удаления таблицы:

```
function DelSubjDB(table);
```

Где:

table – (string) название таблицы, ранее созданной в базе данных объектов с помощью функции CreateSubjDB.

name – (string) название объекта на естественном человеческом языке. На основе склонения этого имени по падежам будет создан фрейм во фреймсете, связанным с таблицей базы данных.

updateFrameset – (bool) признак необходимости обновления фреймов во фреймсете. При групповом удалении данных имеет смысл ставить false. По умолчанию true.

Функция ничего не возвращает.

При удалении записи из таблицы (вызов с двумя параметрами) производится удаление соответствующей записи, и при updateFrameet=true (знач.по умолчанию) производится обновление соответствующего фреймсеса.

При удалении таблицы (вызов с одним параметром) производится удаление таблицы, удаление фреймсета, а также удаление ассоциированного файла.

Функция UpdateSubjDBFrameset

Функция принудительно обновлят фреймсет соответствующий указанной таблицы базы данных объектов.

```
function UpdateSubjDBFrameset(table);
```

Где:

table – (string) название таблицы, ранее созданной в базе данных объектов с помощью функции CreateSubjDB.

Функция ничего не возвращает.

Вызов функции имеет смысл поизводить только при групповом добавлении или удалении данных из соответствующей таблицы базы данных объектов. В этом случае параметр updateFrameset у операций SetSubjDB или DelSubjDB имеет смысл делать равным false, а после проведения всех операций вызвать UpdateSubjDBFrameset. Такой механизм более эффективен в плане производительности.

Пример:

```
SetSubjDBPath("myDB"); // указать папку для хранения данных

CreateSubjDB("Fructs");  // создать таблицу (и считать ее)

// внести групповое изменение данных без обновл.фреймсета
SetSubjDB("Fructs", "Яблоко", 1, false);
SetSubjDB("Fructs", "Груша", 2, false);
SetSubjDB("Fructs", "Персик", 3, false);
UpdateSubjDBFrameset("Fructs"); // обновить фреймсет
```

Функция SetSubjDBPath

Функция указывает папку на диске (относительно папки DynRobot3), где нужно хранить базу данных объектов.

```
function SetSubjDBPath(path);
```

Где:

path – (string) путь относительно папки DynRobot3 или абсолютный путь.

Функция ничего не возвращает.

По умолчанию путь – пустая строка, т.е. файлы с таблицами база данных объектов создаются прямо в папке DynRobot3.

Пример:

```
SetSubjDBPath("c:\\Data\\myDB");
```

Функция CreatePersonFrameset

Функция создает фреймсет, ассоциированный с базой данных лиц, а точнее с именами персон из базы данных лиц.

```
function CreatePersonFrameset(name);
```

Где:

name – (string) название frameset-а.

Функция не возвращает ничего.

До вызова этой функции система не ассоциирует персон базы данных лиц с фреймом. Первый вызов этой функции задает название такого фреймсета.

После вызова отказаться от такого фреймсета больше нельзя.

При этом если в записи персон вносятся изменения (например, добавляются персоны), то обновление фреймсета производится автоматически.

Во фреймсет записываются фреймы, соответствующие именам персон в базе данных лиц, подверженные склонениям во всех падежах единственного числа. Эти фреймы возвращают идентификатор персоны.

Следует отметить, что при изменении имени персоны через прямую правку файла «info.txt» в соответствующей папке с персоной, система не обнаруживает этих изменений, пока не будет обращений к имени персоны через функцию PersonName или иным атрибутам этой персоны через функции PersonRights или PersonInfo. Поэтому данный фреймсет при ручном изменении файла «info.txt» в ряде случаев перестраиваться не будет.

Пример:

```
// создать фреймсет persons, ассоциированный с базой данных
лиц
CreatePersonFrameset("persons");

// пробуем
frameset("commands2",1)
{
   frame("Подойди к <кому:persons>")
   {
      var personID = ValueOf("кому"); // получить id-
персоны
      var personName = GetPersonName(personID); // имя
персоны

      // запустить алгоритм подхода к персоне
      FollowFace(personID, true);
      while(FollowFace()>0) sync();
      if (FollowFace()==0)
         say("Здравствуйте, "+personName);
      else
         say("Не удалось подойти к "+TextOf("кому"));
   }
}

SpeechRecognizer(true);
FaceRecognizer(true);
while(true) sync();
```

Функция CreateSpecialPersonFrameset

Функция создает фреймсет, ассоциированный с базой данных лиц, а точнее с именами персон из базы данных лиц, только для лиц с правами, отличными от нуля.

```
function CreateSpecialPersonFrameset(name);
```

Где:

name – (string) название frameset-а.

Функция не возвращает ничего.

Действия и поведение функции полностью идентичны функциии CreatePersonFrameset, за исключением того, что список составляется не по всем персонам, а только по VIP-персонам и хозяевам робота.

20. Функции работы с экранным контентом и мультимедиа

Функция SetBrowserURL

Функция отправляет команду в экраннвый контент.

```
function SetBrowserURL(command);
```

Здесь:

command – команда для экранного контента, аналогичная событиям в XML-файле, и может состоять из строк следующего формата:

«**javascript://***скрипт*» – выполнить javascrpt

«*текст*» – трактуется как переход на указанных URL.

Например:

```
SetBrowserURL("main.html"); // открыть на экране main.html

Sleep(2000);

// выполнить javascript, в частности вызвать функцию
myJSFunc()
SetBrowserURL("javsscript://myJSFunc()");
```

Функция PlayWave

Функция управляет воспроизведением аудиофайлов (MP3/WAV/WMA).

```
function PlayWave();    // определить состояние
```

или

```
function PlayWave(false);  // остановить воспроизведение
```

или

```
function PlayWave(mp3File); // воспроизвести файл
```

Здесь:

mp3File – аудиофайл (MP3/WAV/WMA).

Функция возвращает true, если воспроизведение идет в данный момент времени.

Функция асинхронная, т.е. она лишь аапускает (или останавливает) воспроизведение и выходит. При этом воспроизведение идет фоноым процссом.

Например:

```
PlayWave("../my.mp3"); // запустить MP3
while(PlayWave()) sync();  // ждать, пока воспроизводится.
```

Функция AudioVolume

Функция устанавливает или получает текущую громкость аудиосистемы.

```
function AudioVolume();    // определить громкость
```

или

```
function AudioVolume(volume); // установить громкость
```

Здесь:

volume – (int) громкость в процентах (0..100).

Функция возвращает (int) текущую громкость в процентах.

Функция VideoRecorder

Функция управляет видеорекордером робота.

```
function VideoRecorder(camera, fileName, duration, fps);
```

или

```
function VideoRecorder(camera, fileName, duration);
```

или

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

или

Остановить запись:

```
function VideoRecorder(false);
```

или

Опросить состояние:

```
function VideoRecorder();
```

Здесь:

camera – (int) номер камеры, с которой нужно производить запись. Если (-1), то используется камера по умолчанию заданная в «config.txt» параметром «VIDEO_RECORDER/camera».

fileName – (string) имя файла. Формат MP4 (h264,AAC) Если не указан полный путь, то используется путь относительно папки текущего Web-контента. В режиме непрерывной записи серии видеороликов, имя файла желательно составлять в формате «xxxx0000.mp4» (здесь xxxx – произвольные буквы, «0000» – цифры, количество которых определяет формат номера файла). Только в режиме записи серии видеороликов система определяет номер аналогичного файла в данной папке, после чего цифры заменяются на номер, на единицу больше, чем номер файла, который уже есть в папке.

duration – (float) время записи в секундах, по умолчанию 60. Если указан 0 или меньше, то включается режим непрерывной записи серии видеороликов. Продолжительнось каждого такого ролика определяется в «config.txt» параметром «VIDEO_RECORDER/ seq_part_duration». При этом имя файла желательно составить согласно рекомендациям выше.

fps – (float) FPS записи, если указать 0, отрицательное число или не указывать параметр, то FPS определяется через «config.txt» параметром «VIDEO_RECORDER/fps».

Функция возвращает (bool) текущее состояние видеоплеера: true – идет запись,
false – запись остановлена.

При вызове с параметром false видеозапись останавливается.

21. Работа с внешними интернет-соединениями

21.1. Отправка почты (работа с SMTP)

Функция SendMailTo

Функция отправляет e-mail.

```
function SendMailTo(to, subject, body,
                        attach, mime, attach2, mime2);
```

или

```
function SendMailTo(to, subject, body, attach, mime);
```

или

```
function SendMailTo(to, subject, body);
```

Здесь:

- to – e-mail, куда отправлять письмо.
- subject – тема письма.
- body – тело письма. Если в теле содержится тег <html>, то письмо формируется в формате HTML, иначе в формате простого текста.
- attach – название прикрепляемого к письму файла.
- mime – MIME-type прикрепленного файла (например, «image/jpeg»).
- attach2 – название второго прикрепляемого к письму файла.
- mime2 – MIME-type второго прикрепленного файла (например, «image/jpeg»).

Функция возвращает false в случае ошибки.

Следует учесть, что отправка почты осуществляется асинхронно, поэтому даже если функция вернула true, нет гарантий того, что письмо отправилось.

21.2. Управление HTTP-запросами

Система IScript3 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, method,
ignoreHttpStatus);
```

или

```
function HttpRequestPlain(URL,
                        postData, cookie, headers, method);
```

или

```
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://».
  Знак «@» перед адресом запрещает использовать прокси-сервер для указанного адреса. Например. «@http://192.168.0.100:81/index.html».
- postData – данные, передаваемые методом POST. Если данные не заданы или равны null, то формируется запрос методом GET (при условии, что не задан иной метод параметром method). В случае отправки данных «Content-Type» задается в виде «`application/x-www-form-urlencoded`», при условии, что в параметре headers не содержится иного значения параметра «Content-Type».
- cookie – строка COOKIE (в формате HTTP-запроса) или null (см. также GetLastCookie()).
- headers – дополнительные заголовки HTTP-запроса. Заголовки должны представлять собой набор строк, разделенных символами«\r\n». Строка заголовков должна также заканчиваться символами «\r\n».
- method – HTTP-метод запроса. По умолчанию GET или POST в зависимости от присутствия postData.
- ignoreHTTPStatus – игнорировать статус ответа HTTP, вернуть текст, даже если HTTP-статус не равен 200. По умолчанию false.

Функция возвращает планарный текст документа, который вернул сервер, или null в случае ошибки.

Параметры запросов отправляются в оригинальной кодировке.

Кодировка ответа будет зависеть от кодировки, заданной с помощью HTTPEncoding().

Если в системных настройках Интернета среды Windows указан прокси-сервер, функция использует указанный прокси-сервер, если его использование не запрещено значком «@» перез адресом.

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 HttpRequestJSON

Функция формирует запрос к WEB-серверу по протоколу HTTP/HTTPS или file, возвращающему ответ в виде JSON.

```
function HttpRequestJSON(URL,
         postData, cookie, headers, method,
ignoreHttpStatus);
```

или

```
function HttpRequestJSON(URL,
                        postData, cookie, headers, method);
```

или

```
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://», «https://» или «file://».
  Знак «@» перед адресом запрещает использовать прокси-сервер для указанного адреса. Например. «@http://192.168.0.100:81/index.html».
- postData – данные, передаваемые методом POST. Если данные не заданы или равны null, то формируется запрос методом GET (при условии, что не задан иной метод параметром method). В случае отправки данных «Content-Type» задается в виде «`application/x-www-form-urlencoded`», при условии, что в параметре headers не содержится иного значения параметра «Content-Type».
- cookie – строка COOKIE (в формате HTTP-запроса) или null (см. также GetLastCookie()).

- headers – дополнительные заголовки HTTP-запроса. Заголовки должны представлять собой набор строк, разделенных символами«\r\n». Строка заголовков должна также заканчиваться символами «\r\n».
- method – HTTP-метод запроса. По умолчанию GET или POST в зависимости от присутствия postData.
- ignoreHTTPStatus – игнорировать статус ответа HTTP, вернуть текст, даже если HTTP-статус не равен 200. По умолчанию false.

Функция возвращает объект, представляющий собой разобранный json-ответ сервера, или null в случае ошибки.

Запросы, формируемые данной функцией, аналогичны запросам функции HttpRequestPlain.

Если в системных настройках Интернета среды Windows указан прокси-сервер, функция использует указанный прокси-сервер, если его использование не запрещено значком «@» перез адресом.

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://», «https://» или «file://».
  Знак «@» перед адресом запрещает использовать прокси-сервер для указанного адреса. Например. «@http://192.168.0.100:81/index.html».
- postData – данные, передаваемые методом POST. Если данные не заданы или равны null, то формируется запрос методом GET (при условии, что не задан иной метод параметром method). В случае отправки данных «Content-Type» задается в виде «`application/x-www-form-urlencoded`», при условии, что в параметре headers не содержится иного значения параметра «Content-Type».
- cookie – строка COOKIE (в формате HTTP-запроса) или null (см. также GetLastCookie()).
- headers – дополнительные заголовки HTTP-запроса. Заголовки должны представлять собой набор строк, разделенных символами«\r\n». Строка заголовков должна также заканчиваться символами «\r\n».
- method – HTTP-метод запроса. По умолчанию GET или POST в зависимости от присутствия postData.
- ignoreHTTPStatus – игнорировать статус ответа HTTP, вернуть текст, даже если HTTP-статус не равен 200. По умолчанию false.

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.
.text – (строка) текст простого тэга (например: <config>My Test</config>)

Все атрибуты XML-тэга также формируются в виде полей объекта.

Например, пусть сервер возвращает следующий 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.text, "\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 кодировке заданный с помощью HTTPEncoding().

Если в системных настройках Интернета среды Windows указан прокси-сервер, функция использует указанный прокси-сервер, если его использование не запрещено значком «@» перез адресом.

Параметры запроса передаются в оригинальной кодировке.

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();
      ...
}
```

Функция HTTPEncoding

Функция задает кодировку HTTP-ответов.

```
function HTTPEncoding(encoding);
```

или

```
function HTTPEncoding();
```

Где:

encoding – (string) кодировка. Поддержиается только «UTF-8» и «ANSI».

Функция возвращает текущую кодировку.

По умолчанию при запуске скрипта установлена кодировка «UTF-8».

Например:

```
HTTPEncoding("ansi");
html =
HttpRequestPlain("http://ipusn.dynsoft.ru/forum.php");
if (html)
{
     print html;
}
```

Функция GetIPAddressList

Функция получает список IP-адресов робота.

```
function GetIPAddressList();
```

Функция возвращает массив строк, в каждом из которых записан IP-адрес (через точки).

Например:

```
var list = GetIPAddressList();
for(var i=0; i < count(list); i++)
{
    print(list[i],"\n");  // выводит, что-то типа:
«11.0.0.2»
}
```

Функция IsInternet

Функция сообщает о наличие или отсутствие интернета.

```
function IsInternet();
```

Функция возвращает (bool) признак наличия интернета. Проверка происходит асинхронно, но нужно с помощью данной функции «будить» процесс опроса интернета с интервалом, не менее 8 секунд (лучше меньше). В противном случае функция будет возвращать последнее состояние опроса.

Функция HasUpdates

Функция определяет наличие обновлений для робота.

```
function HasUpdates();
```

Функция возвращает (bool) признак наличия обновлений для робота.

Функция InstallUpdates

Функция устанавливает обновлений для робота.

Установить обновления или продолжить установку в случае ошибки:

```
function InstallUpdates(true);
```

или

Прервать установку обновлений:

```
function InstallUpdates(false);
```

или

Проверить, устаналиваются ли сейчас обновления:

```
function InstallUpdates();
```

Функция возвращает (int):

- 0 – обновлений не устаналиваются или нет обвлений, требующих установки (в случае запуска с параметром true).
- 1 – обновления установлены.
- (-1) – ошибка установки. Требует повторного вызова функции с параметром true, либо false.

22. Управление роботом

22.1. Управление шасси

Функция SetWheelSpeed или SetWheelsSpeed

Функция управляет скоростью шасси:

```
function SetWheelSpeed();
```

или

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

или

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

или

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

или

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

Функция SetWheelsSpeed является синонимум SetWheelSpeed.

Здесь:

- v – (float) скорость линейного движения в диапазоне от -1.0 до 1.0.
- q – (float) скорость поворота в диапазоне от -1.0 до 1.0. По умолчанию 0.0.
- s – (float) скорость стрейфа в диапазоне от -1.0 до 1.0 (если поддерживается). По умолчанию 0.0.
- wheelMode – режим шасси (если трехколесное шасси с поворотными колесами):
  - 0 – режим движения.
  - 1 – режим поворота на месте.
  - 2 – режим стрейфа на месте.

  По умолчанию используется текущий режим шасси.

При вызове без параметров лишь возвращает значение, не управляя скоростью.

Функция возвращает текущий режим шасси 0, 1 или 2, если он поддерживается.

Функция GetZoneDanger

Функция возвращает опасность указанной зоны локальной карты местности.

```
function GetZoneDanger(n);
```

или

```
function GetZoneDanger();
```

Где:

n – (int) номер зоны локальной карты:

(-1) – левая зона.
0 – передняя зона
1 – правая зона.
2 – задняя зона.

Если зона не указана, или *n* вне указанного перечня, то возвращает опасность передней зоны.

Опасность зоны определяется в виде (float) от 0 до 1.

22.2. Управление интеллектуальным движением и картой

Функция Go

Функция отправляет робота к указанной точке на карте или проверяет статус движения.

Отправить к месту на карте:

```
function Go(placeNumber);
```

или

Отправить к точке, заданной координатами

```
function Go(coord);
```

или

Остановить движение:

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

или

Проверить статус движения:

```
function Go();
```

Где:

placeNumber – номер места на карте.

coord – объект (object) имеющий следующие поля:

.x – координата X (см).

.y – координата Y (см).

.a – угол азимута (радианы).

Функция возвращает:

- 0 – движение не активно или закончилось успехом.
- 1 – движение все еще в процессе.
- (-1) – движение завершилось ошибкой (робот не пришел к точке).

Функция выполняется асинхронно и не задерживает выполнение скрипта.

Вызов Go() без параметров проверяет текущий стутус движения. При этом если движение завершилось с ошибкой, то все последующие вызовы Go() без параметров будет возвращать статус ошибки, пока робота не отправят в новую точку функциями Go(…) или GoToPlace().

Вызов Go(false) останавливает движение. При этом, если движение осуществлялось, функция возвращает (-1) – ошибка, что символизирует, что движение было прервано, и робот до точки не дошёл. Если робот не двигался, то функция не меняет статус движения.

Пример:

```
Go(5); // отправить в точку 5
while(Go()==1) // пока робот двигается...
{
    // здесь можно совершить какие-нибудь полезные действия
    // во время движения
    ...
```

```
    sync(); // синхронизация
}
if (Go()==-1)  // Ошибка!
    PlaySpeech("Не удалось приехать в точку 5");
else  // Успех
    PlaySpeech("Я на точке 5");
```

Функция GoToPlace

Функция отправляет робота к указанной точке на карте и дожидается, пока робот не приедет.

```
function GoToPlace(placeNumber);
```

или

```
function GoToPlace(coord);
```

Где:

placeNumber – номер места на карте.

coord – объект (object) имеющий следующие поля:

.x – координата X (см).

.y – координата Y (см).

.a – угол азимута (радианы).

Функция возвращает:

- true – робот пришел в точку.
- false – движение завершилось ошибкой (робот не пришел к точке).

Скрипт не выходит из указанной функции, пока движение активно. Однако паралельно движению возникают события и работают таймеры.

Пример:

```
if (GoToPlace(5)) // отправить в точку 5
{
    // Успех
    PlaySpeech("Я на точке 5");
}
else
{
    // Ошибка!
    PlaySpeech("Не удалось приехать в точку 5");
}
```

Функция WheelMovingBy

Функция отправляет робота на указанное перемещение относительно текущей точки.

Отправить на перемещение:

```
function WheelMovingBy(da, dy, dx, obstacles);
```

или

```
function WheelMovingBy(da, dy, dx);
```

или

```
function WheelMovingBy(da, dy, obstacles);
```

или

```
function WheelMovingBy(da, dy);
```

или

```
function WheelMovingBy(da, obstacles);
```

или

```
function WheelMovingBy(da);
```

или определить статус:

```
function WheelMovingBy();
```

или отменить движение:

```
function WheelMovingBy(false);
```

Где:

da – (float) угол поворота (градусы), положительное направление вправо.

dy – (float) величина перемещения вперед/назад (см), положительное направление вперед.

dx – (float) величина стрейфа вправо/влево (см), положительное правление вправо.

obstacles – (bool) признак реакции на препятствия. По умолчанию false.

Функция возвращает:

1 – движение запущено и всё ещё продолжается.

0 – движение удачно завершилось или не начиналось.

(-1) – движение закончилось неудачей (было прервано или робот наткнулся на препятствия).

Функция осуществляется асинхронно. Т.е. она лишь запускает сам процесс движения, скрипт при этом продолжает работать. Пользователь может определить статус этого движения, выполняя функцию без параметров.

Чтобы остановить движение функцию нужно вызывать с параметром false.

Следует понимать, что функции в качестве параметров передаются не координаты точки, куда роботу нужно попасть, а величины перемещений по осям, которые робот выполняет в порядке ПОВОРОТ (da), СТРЕЙФ (dx), ДВИЖЕНИЕ (dy). Стрейф осуществляется, только если стрейф поддерживается текущим шасси робота.

См. также WheelMovingByAndWait.

Пример:

```
WheelMovingBy(30); // задать роботу поворот на 30° право
while(WheelMovingBy()==1) // пока робот двигается...
{
     ...  // сделать что-то полезное
     sync(); // синхронизация
}

if (WheelMovingBy()==-1)
    PlaySpeech("Не удалось повернуться");
else
    PlaySpeech("Поворот удачно завершён");
```

Функция WheelMovingByAndWait

Функция отправляет робота на указанное перемещение относительно текущей точки и ожидает его завершение.

Отправить на перемещение:

```
function WheelMovingByAndWait(da, dy, dx, obstacles);
```

или

```
function WheelMovingByAndWait(da, dy, dx);
```

или

```
function WheelMovingByAndWait(da, dy, obstacles);
```

или

```
function WheelMovingByAndWait(da, dy);
```

или

```
function WheelMovingByAndWait(da, obstacles);
```

или

```
function WheelMovingByAndWait(da);
```

Где:

da – (float) угол поворота (градусы), положительное направление вправо.

dy – (float) величина пробега вперед/назад (см), положительное направление вперед.

dx – (float) величина стрейфа вправо/влево (см), положительное правление вправо.

obstacles – (bool) признак реакции на препятствия. По умолчанию false.

Функция возвращает:

true – движение удачно завершено.

false – движение закончилось неудачей (было прервано или робот наткнулся на препятствия).

Функция дожидается завершения движения, при этом все события и таймеры в скрипте продолжают работать.

Чтобы остановить движение нужно вызывать функцию WheelMovingBy с параметром false.

Следует понимать, что функции в качестве параметров передаются не координаты точки, куда роботу нужно попасть, а величины перемещений по осям, которые робот выполняет в порядке ПОВОРОТ (da), СТРЕЙФ (dx), ДВИЖЕНИЕ (dy). Стрейф осуществляется, только если стрейф поддерживается текущим шасси робота.

См. также WheelMovingBy.

Пример:

```
if (WheelMovingByAndWait(30)) // задать роботу поворот на
30°
    PlaySpeech("Поворот удачно завершён");
else
     PlaySpeech("Не удалось повернуться");
```

Функция KickNavigation

Функция будит навигацию робота на следующие 3 секунды.

```
function KickNavigation();
```

Вспомогательная функция, использование которой требуется лишь в некоторых специфических случаях, т.к. навигация робота сама автоматически «просыпается» и «засыпает» по мере необходимости. Поэтому у разработчика скрипта нет необходимости использовать данную функцию, кроме исключительных случаев.

Но в некоторых особых случаях ее можно принудительно заранее «разбудить» вызовом данной функции. Если после этого навигация останется невостребованной, то через 3 секунды навигация опять уйдет в режим сна. Поэтому данную функцию следует вызывать переодически, пока навигация нужна.

Функция GetCurrentZone

Функция сообщает номер зоны, в которой в данный момент находится робот.

```
function GetCurrentZone();
```

Возвращает номер зоны, по которой в данный момент движется робот или возвращает (-1).

Функцию удобно использовать в циклах совместно с функцией Go().

Пример:

```
Go(5); // отправить в точку 5
var zone = -1; // переменная для хранения последней зоны
while(Go()==1) // пока робот двигается...
{
   // получить номер зоны, по которой едет робот или -1.
   var n = GetCurrentZone();
   if (n != zone) // если не совпадает с предыдущей, то
сообщить
   {
      if (zone>=0)
          PlaySpeech("Я в зоне "+n);
      else
          PlaySpeech("Я покинул зону "+zone);
      zone = n;
   }
   sync(); // синхронизация
}
if (Go()==-1)
    PlaySpeech("Не удалось приехать в точку 5");
else
    PlaySpeech("Я на точке 5");
```

Функция GetCurrentPlace

Функция сообщает номер текщего места, в котором в данный момент находится робот.

```
function GetCurrentPlace();
```

Возвращает номер места, в которое в последний раз отправляли робота (с помощью скриптов или иным способом). Этот номер хранится даже после перезапуска скриптов, но не хранится после перезапуска «ДинРобот-3».

Как только робот отправляется к новой точке, номер текущего места становится равным (-1). Если робот приходит в указанную точку номер текущего места становится равным номеру этого места.

Функция SetCurrentPlace

Функция устанавливает виртуальные координаты робота на карте в координаты указанного места.

```
function GetCurrentPlace(number);
```

Где:

number – номер места.

Функция возвращает true в случае успеха. Если место не найдно, то возращает false.

После успешного вызова данной функции, функция GetCurrentPlace() будет возращать указанный номер.

Если number имеет отрицательное значение, например, (-1), то виртуальные координаты робота на карте не меняются, но функция GetCurrentPlace() будет возвращать значение (-1).

Функция IsPlaceExists

Функция возвращает true, если место существует на карте.

```
function IsPlaceExists(number);
```

или

```
function IsPlaceExists();
```

Где:

number – номер места.

При вызове без параметров или при number равным (-1), функция проверяет существование текущего места, если оно задано (см. GetCurrentPlace()).

Функция возвращает bool.

Функция GetPlaceDesc

Функция возвращает текст описания указанного места, заданное на карте.

```
function GetPlaceDesc(number);
```

или

```
function GetPlaceDesc();
```

Где:

number – номер места.

При вызове без параметров или при number равным (-1), функция возвращает описание текущего места (см. GetCurrentPlace()).

Функция возвращает строку (string) с описанием текущего места. Если место не найдено, возвращает null.

Функция GetZoneDesc

Функция возвращает текст описания указанной зоны, заданной на карте.

```
function GetZoneDesc(number);
```

или

```
function GetZoneDesc();
```

Где:

number – номер зоны.

При вызове без параметров или при number равным (-1), функция возвращает описание текущей зоны (см. GetCurrentZone()).

Функция возвращает строку (string) с описанием текущей зоны. Если зона не найдена, возвращает null.

Функция GetPlaceCoord

Функция возвращает координаты указанного места.

```
function GetPlaceCoord(number);
```

или

```
function GetPlaceCoord();
```

Где:

number – номер места.

При вызове без параметров или при number равным (-1), функция возвращает описание текущего места (см. GetCurrentPlace()).

Функция возвращает объект со следующими полями:

- .x – координата X (см).
- .y – координата Y (см).
- .a – угол азимута (радианы).

Если место не найдено, возвращает null.

Функция GetRobotCoord

Функция возвращает текущие координаты робота.

```
function GetRobotCoord();
```

Функция возвращает объект со следующими полями:

- .x – координата X (см).
- .y – координата Y (см).
- .a – угол азимута (радианы).

Функция SetRobotCoord

Функция устанавливает координаты виртуальной отметки робота на карте.

```
function SetRobotCoord(coord);
```

или

```
function SetRobotCoord(placeNumber);
```

или

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

или

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

Где:

- coord – объект (object) со следующими полями:
  - .x – координата X (см).
  - .y – координата Y (см).
  - .a – угол азимута (радианы).
- x, y – координаты робота в сантиметрах.
- a – угол азимута робота в радианах.
- placeNumber – (int) номер места. Должно быть исключительно типом int. Аналогичен вызову SetCurrentPlace().

Функция возвращает true (bool) в случае успеха или false в случае ошибки.

Функция GetPlaceList

Функция получает список всех мест (точек) на карте в виде массива.

```
function GetPlaceList();
```

Возвращает массив (array) с номерами мест карты (int), или null в случае ошибки.

Функция GetZoneList

Функция получает список всех зон на карте в виде массива.

```
function GetZoneList();
```

или для определенного типа зон:

```
function GetZoneList(zoneType);
```

Где:

zoneType – (int) тип интересующих зон:

- (-1) – все зоны;
- 0 – пользовательские зоны;
- 1 – закрытые зоны;
- 2 – узкие зоны;
- 3 – светофоры.

Возвращает массив (array) с номерами зон карты (int), или null в случае ошибки.

Функция GetZoneType

Функция получает тип указанной зоны.

```
function GetZoneType(number);
```

или для текущей зоны, по которой двигается робот:

```
function GetZoneType();
```

Где:

number – (int) номер зоны.

Возвращает (int):

- (-1) – ошибка;
- 0 – пользовательская зона;
- 1 – закрытая зона;
- 2 – узкая зона;
- 3 – светофор.

Функция IsNarrowZone

Функция получает, едет ли сейчас робот по узкой зоны

```
function IsNarrowZone();
```

Возвращает (bool) true, если зона узкая, иначе false.

Функция GetLighttraffic

Функция получает, является ли указанный светофор красным.

```
function GetLighttraffic(number);
```

или для текущего светофора, на котором сейчас находится робот:

```
function GetLighttraffic();
```

Где:

number – (int) Номер зоны-светофора.

Возвращает (bool) true, если светофор красный; false – светофор зеленый; null – ошибка, например, светофора не существует.

Функция Park

Функция управляет парковкой на зарядку.

```
function Park(onOff);
```

или

```
function Park();
```

Где:

onOff – (bool) запустить или выключить систему парковки на зарядку.

Функция возвращает (int):

- 0 – парковка успешно завершена или не запущена.
- 1 – парковка в процессе.
- (-1) – парковка завершена неудачно (робот не припарковался) или парковка была прервана.

Парковка на зарядку осуществляется асинхронно. Функция Park лишь запускает или останавливает систему парковки. При вызове функции без параметров производится лишь проверка состояния процесса парковки.

Пример:

```
Park(true);         // запустить процесс парковки
while(Park()==1)    // пока парковка в процессе
{
   sync();      // синхронизация
}
if (Park()==-1)  // проверить состояние: -1 или 0
   PlaySpeech("Не удалось подключиться к зарядке");
else
   PlaySpeech("Как же приятно подключиться к любимой
зарядке");
```

Функция FollowFace

Функция включает или выключает режим «Следуй за лицом».

```
function FollowFace(personId, comeOnly);
```

или

```
function FollowFace(personId);
```

или

```
function FollowFace(false);
```

или

```
function FollowFace();
```

Где:

personId – (string) идентификатор персоны, за которой нужно следовать.

comeOnly – (bool) режим, при котором робот только подходит к персоне. По умолчанию false.

При вызове функции FollowFace(…) с единственным параметром false, производится выключение режима «Следуй за лицом».

Вызов функции без параметров осуществляется лишь для определения текущего состояния системы движения за лицом.

Независимо от параметров функция возвращает (int) текущее состояние системы следования за лицом:

- -1 – режим отменен путем вызова с функции FollowFace с параметром false.
- 0 – режим отключился сам по таймауту или по приходу к персоне (в режиме comeOnly).
- 1 – ожидание появления персоны.
- 2 – персона обнаружена.

Чтобы работала функция необходимо наличие системы распознавания лиц.

Пример:

```
FollowFace("Дима222", false); // включить режим следуй за лицом

// ждать появления персоны или выключения системы
while(FollowFace()==1) sync();

// обнаружено ли лицо?
if (FollowFace() == 2)
{
     PlaySpeech("Следую за Димой");

     // ждать пропадания персоны
     while(FollowFace()==2) sync();

     PlaySpeech("Бужу ждать здесь");
}
else
{
     PlaySpeech("Эх. Дима так и не подошёл");
}
```

Функция GetEmg

Функция возвращает true, если хотя бы на одном аппаратном устройстве имеется ошибка.

```
function GetEmg();
```

Возвращает (bool) true, признак наличия аппаратной ошибки на одном из устройств робота. Например, нажатие аварийной кнопки на роботах «Маша» или аварийный режим для манипулятором xArm.

Функция SetLighttraffics

Функция устанавливает перечень закрытых светофоров.

```
function SetLighttraffics(redLighttraffics);
```

Где:

redLighttraffics – (string) перечень (через любой разделитель) закрытых светофоров.

Функция не возврает ничего.

Напомним, что «Светофор» – это зона на карте, имеющая тип «Светофор». Светофоры могут быть открытом состоянии (по умолчанию все светофоры открыты), и в закрытом состоянии. Перечень закрытых светофоров устанавливается данной функцией. Кстати, при остановке скриптов все светофоры в памяти робота открываются.

«ДинРобот-3» использует светофоры при работе нескольких роботов в одном помещении. В основном светофоры предназначены для блокировки въезда двух роботов в узкие тупиковые ответвления карты. Вся зона тупикового ответвления должна быть помечена зоной «Светофор» так, чтобы оставались пути выезда из этой зоны.

По средствам IScript3 между роботами следует организовать обмен данными. Рекомендуется использовать для этого отдельный WEB-сервер управления виртуальными светофорами, но можно организовать такой сервер и на одном из роботов.

Каждый робот, въезжая в зону светофора, получает событие в IScript3 «* LIGHTTRAFFIC *n*», где *n* – номер светофора (номер зоны с типом «Светофор»). Этот номер он может передать на сервер управления светофорами, как признак его блокировки. Следует отметить, что выезжая из зоны светофора, в роботе формируется событие «* LIGHTTRAFFIC 0». Передавая этот «0» на сервер управления светофорами, робот информирует сервер, что он покинул зону действия светофора.

Сервер управления светофорами должен разослать всем остальным роботам сообщение о том, какие из светофоров заблокированы. Светофоры, заблокированные роботом, для которого предназначено сообщение, должы быть исключены из этого списка. Рассылка сообщения о заблокированных светофорах может быть реализована за счет того, что роботы сами периодически запрашивают у сервера состояние этих заблокированных светофоров.

Робот, получая от сервера управления светофорами информацию о заблокированных светофорах, с помощью функции SetLighttraffics может установить блокировку виртуальных светофоров в своей памяти.

Система управления движением робота по карте местности следит за заблокированными светофорами и останавливает робота при въезде в зону заблокированного светофора. Движение не возобновляется до тех пор, пока светофор заблокирован.

Реализации сервера управления светофорами в «ДинРобот-3» нет, этот сервер должен быть разработан самим пользователем робота. Это связано с тем, что полный функционал такого сервера достаточно сильно зависит от задач, решаемых роботами.

22.3. Управление головой и подъемником

Функция Head

Функция управляет поворотами головы.

```
function Head(angle, pitch, roll);
```

или

```
function Head(angle, pitch);
```

или

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

или

```
function Head();
```

Здесь:

angle – (float) угол поворота по горизонтали (градусы) (положительно – вправо).

pitch – (float) угол налона по горизонтали (градусы) (положительно – вверх). По умолчанию используется текущий угол наклона головы.

roll – (float) угол крена головы (градусы) (положительно – вправо). По умолчанию используется текущий угол крена головы.

Функция возвращает горизонтальный угол поворота головы.

Функция HeadRel

Функция управляет поворотами головы относительно ее текущих уголов поворота.

```
function HeadRel(dAngle, dPitch, dRoll);
```

или

```
function HeadRel(dAngle, aPitch);
```

или

```
function HeadRel(dAngle);
```

Здесь:

dAngle – (float) относительный угол поворота по горизонтали (градусы) (положительно – вправо).

dPitch – (float) относительный угол налона по горизонтали (градусы) (положительно – вверх). По умолчанию 0.0.

dRoll – (float) относительный угол крена головы (градусы) (положительно – вправо). По умолчанию 0.0.

Функция не возвращает никаких значений.

Функция GetHeadAngle

Функция возвращает трекущий горизонтальный угол поворота головы (градусы).

```
function GetHeadAngle();
```

Функция не возвращает (float) угол поворота головы (градусы). Аналогична вызову функции Head без параметров.

Функция GetHeadPitch

Функция возвращает трекущий вертикальный угол наклона головы (градусы).

```
function GetHeadAngle();
```

Функция не возвращает (float) угол наклона головы (градусы).

Функция GetHeadRoll

Функция возвращает трекущий угол крена головы (градусы).

```
function GetHeadRoll();
```

Функция не возвращает (float) угол крена головы (градусы).

Функция Lift

Функция управляет подъемником (если он есть).

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

или

```
function Lift();
```

Здесь:

height – (float) высота подъема (см).

Функция возвращает текущее значение высоты подъемника.

Функция SetLiftSpeed

Функция управляет скорость движения подъемника (если он есть).

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

Здесь:

speed – (float) задаваемая скорость от -1 до 1.

Функция ничего не возвращает.

После применения функции необходимо останавливать движение подъемника путем установки нулевой скорости.

Функция LockFaceTracking или LockFaceTracker

Функции LockFaceTracking и LockFaceTracker являются синонимами.

Функция блокирует работу трекера лиц с помощью головы или определяет статус этой блокировки.

Для блокировки или разблокировки:

```
function LockFaceTracking(lock);
```

или для определения статуса

```
function LockFaceTracking();
```

Здесь:

lock – (bool) признак блокировки.

Функция возвращает текущее состояние блокировки.

Блокировка головы актуальна при работе фотосервисов. Постоянный поворот головой мешает людям адекватно встать перед роботом.

22.4. Управление отдельными моторами

Функция SetMotorPos или SetMotorPosition

Функция устанавливает заданное положение вала логического мотора.

```
function SetMotorPos(motor, pos);
```

или

```
function SetMotorPos(motor, pos, time);
```

Функция SetMotorPosition является синонимум SetMotorPos.

Здесь:

motor – (int) номер логического мотора.

pos – (float) положение вала мотора в сантиметрах или градусах (согласно формуле пересчета показания датчика в значение, заданная в hw_config.txt).

time – (int) кол-во миллисекунд, за которое мотор должен придти в заданное положение. При time=0, время прихода (скорость движения) определяется автоматически драйвером мотора. По умолчанию time=0.

Функция в случае успеха возвращает true. В случае отсутствия соответствующего логического мотора возвращает false.

Драйвер указанного логического мотор должен поддерживать команды установки положения.

Функция GetMotorPos или GetMotorPosition

Функция устанавливает заданное положение вала логического мотора.

```
function GetMotorPos(motor);
```

или

```
function GetMotorPosition(motor);
```

Функция GetMotorPos является синонимум GetMotorPosition.

Здесь:

motor – (int) номер логического мотора

Функция возвращает положение (float) указанного логического мотора в сантиметрах или градусах, согласно формуле пересчета, заданной в hw_config.txt.

Драйвер указанного логического мотор должен поддерживать команды запроса положения.

Функция SetMotorSpeed

Функция устанавливает скорость указанного логического мотора.

```
function SetMotorSpeed(motor, speed);
```

Здесь:

motor – (int) номер логического мотора

speed – (int) требуемая скорость -127…127.

Функция возвращает true вслучае успеха.

Драйвер указанного мотора должен поддерживать либо команды установки скорости, либо установки значения ШИМ. Если мотор не поддерживает команды установки скорости, то функция устанавливает ШИМ.

Функция SetMotorPWM

Функция устанавливает ШИМ для указанного логического мотора.

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

Здесь:

motor – (int) номер логического мотора

pwm – (int) требуемое значение ШИМ (-127…127).

Функция возвращает true вслучае успеха.

Драйвер указанного мотора должен поддерживать либо команды установки ШИМ, либо команды установки скорости. Если мотор не поддерживает команды установки ШИМ, то используется функция установки скорости мотора.

Функция BeginGroupCtrl

Функция отбирает логический мотор для участия в командах групповой установки положения, скорости или ШИМ моторов. Например, если требуется синхронно запустить сразу несколько моторов. Это особенно актуально для манипуляторов, которых единовременно запускаются все звенья.

```
function BeginGroupCtrl(motor);
```

Здесь:

motor – (int) номер логического мотора

Функция возвращает true вслучае успеха.

Драйвер указанного мотора должен поддерживать команды группового управления.

Пример:

```
...
BeginGroupCtrl(1); // отобрать мотор 1 для группового
управления
BeginGroupCtrl(2); // отобрать мотор 2 для группового
управления
BeginGroupCtrl(3); // отобрать мотор 3 для группового
управления

// задать заданное положение валов моторов, но
// пока не запускать их
SetMotorPos(1, 10.0, 2000);
SetMotorPos(2, 0.15, 2000);
SetMotorPos(3, 42.21, 2000);

// запустить моторы
EndGroupCtrl(1); // отпустить групповое управление мотора 1
EndGroupCtrl(2); // отпустить групповое управление мотора 2
EndGroupCtrl(3); // отпустить групповое управление мотора 3
```

Следует отметить, что если отобранные логические моторы относятся к одному драйверу, то отбор первого же мотора драйвера уже приводит к захвату всех остальных моторов данного драйвера, и отбор последующих моторов данного драйвера уже игнорируется. Но если моторы принадлежат к разным драйверам, то драйвер одной группы моторов «не знает», что производится групповой отобор моторов другого драйвера, поэтому отбор следует осуществить по всем моторам, используемым в групповом управлении.

Отпускание группового управления одного из моторов драйвера функцией EndGroupCtrl приводит к отпусканию всех моторов данного драйвера. Но, опять-таки, если моторы принадлежат к разным драйверам, то другой драйвер «не будет знать», что отпускают моторы соседнего драйвера. Поэтому отпускать следует все моторы, захваченные функцией BeginGroupCtrl.

Следует отметить, что при остановке скриптов производится отпускание всех захваченных моторов.

Функция EndGroupCtrl

Функция отпускает логический мотор, захваченный для группового управления функцией BeginGroupCtrl.

```
function EndGroupCtrl(motor);
```

или

```
function EndGroupCtrl(motor, linearInterpolation);
```

Здесь:

motor – (int) номер логического мотора.

linearInterpolation – (bool) признак линейного движения в указанную точку.

Функция возвращает true в случае успеха.

Драйвер указанного мотора должен поддерживать команды группового управления. Линейная интерполяция возможна только в том случае, если захватываемые моторы принадлежат одному драйверу, а сам драйвер поддерживает команды линейной интерполяции.

Пример:

```
...
BeginGroupCtrl(1); // отобрать мотор 1 для группового
управления
BeginGroupCtrl(2); // отобрать мотор 2 для группового
управления
BeginGroupCtrl(3); // отобрать мотор 3 для группового
управления

// задать заданное положение валов моторов, но
// пока не запускать их
SetMotorPos(1, 10.0, 2000);
SetMotorPos(2, 0.15, 2000);
SetMotorPos(3, 42.21, 2000);

// запустить моторы
EndGroupCtrl(1, true); // отпустить мотор 1, в
лин.интерполяции
EndGroupCtrl(2, true); // отпустить мотор 1, в
лин.интерполяции
EndGroupCtrl(3, true); // отпустить мотор 1, в
лин.интерполяции
```

Следует отметить, что если отобранные логические моторы относятся к одному драйверу, то отбор первого же мотора драйвера уже приводит к захвату всех остальных моторов данного драйвера, и отбор последующих моторов данного драйвера уже игнорируется. Но если моторы принадлежат к разным драйверам, то драйвер одной группы моторов «не знает», что производится групповой отобор моторов другого драйвера, поэтому отбор следует осуществить по всем моторам, используемым в групповом управлении.

Отпускание группового управления одного из моторов драйвера функцией EndGroupCtrl приводит к отпусканию всех моторов данного драйвера. Но, опять-таки, если моторы принадлежат к разным драйверам, то другой драйвер «не будет знать», что отпускают моторы соседнего драйвера. Поэтому отпускать следует все моторы, захваченные функцией BeginGroupCtrl.

Следует отметить, что при остановке скриптов производится отпускание всех захваченных моторов.

Функция CalibMotor

Функция калибрует указанный мотор.

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

или

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

Здесь:

motor – (int) номер логического мотора.

forced – (bool) признак принудительной калибровки, по умолчанию true.

Функция возвращает true в случае успеха.

Функция работает асинхронно, и нет способа определить окончания процесса калибровки.

Драйвера некоторые моторов имеют мягкую и приудительную калибровку. Например, подъемник роботов «Masha» при мягкой калибровке просто считывает последнее установленное значение подъемника из файла, вместо проведения самой калибровки. И лишь при принудительной калибровке происходит непосредственно механический процесс калибовки.

Драйвер указанного мотора должен поддерживать команды калибровки.

Функция IsMotorMoving

Функция запрашивает, осуществляется ли процесс движения указанного мотора.

```
function IsMotorMoving(motor);
```

Здесь:

motor – (int) номер логического мотора.

Функция возвращает true в случае, если мотор находится в процессе движения.

Возможность определения движения мотора зависит от его драйвера. Чаще всего, драйвер не поддерживает эту функцию.

22.5. Управление системой комманд и входных событий

ДинРобот3 поддерживает управление отправкой команд и приема событий от драйверов устройств.

Под командой подразумевается действие, типа «совершить что-либо», например «налить чашку кофе на кофе-машине».

Событие – это аналогичное действие со стороны внешнего по отношению к роботу устройства. Например, «кофе налилось».

Через конфигурационный файл «hwConfig.txt» назначается взаимосвязь между логическим командным каналом и каналом драйвера устройства.

Для событий через «hwConfig.txt» можно назначить формирование фреймовых событий для IScript3, типа «* MyEvent cmd param1 param2 param3».

Функция SendCommand

Функция отправляет команду по указанному логическому каналу.

```
function SendCommand(channel, cmd, param1, param2, param3);
```

или

```
function SendCommand(channel, cmd, param1, param2);
```

или

```
function SendCommand(channel, cmd, param1);
```

или

```
function SendCommand(channel, cmd);
```

Где:

channel – (int) логический номер канала (номер COMMAND_CHANNEL*x,* заданного через hwConfig.txt).

cmd – (int) номер команды (определяется устройством).

param1…param3 – (float) параметры (определяется устройством).

Функция ничего возвращает.

Функция GetEvents

Функция читает события из указанного логического канала событий.

```
function GetEvents(channel);
```

Где:

channel – (int) логический номер канала (номер COMMAND_CHANNEL*x,* заданного через hwConfig.txt).

Функция возвращает массив объектов, с информацией о событиях, возникших с момента последнего чтения этих событий с этого канала. Количество элементов массива соответствует числу событий, а каждый объект описывает параметры самого события. Этот объект содержит следующий поля:

- .cmd – (int) номер команды.
- param1 – (float) первый параметр.
- param2 – (float) второй параметр.
- param3 – (float) третий параметр.

Использовать данную функцию для каналов, которые сами формируют события в IScript3, бессмысленно.

Пример:

```
var events = GetEvents(1); // получить события по каналу 1
```

```
for(var i=0; i < count(events); i++) // цикл по каждому
событию…
{
   // вывести в консоль
   print("-----------------\n");
   print("cmd=", events[i].cmd ,"\n");
   print("param1=", events[i].param1 ,"\n");
   print("param2=", events[i].param2 ,"\n");
   print("param3=", events[i].param3 ,"\n");
}
```

22.6. Управление системой питания

Функция GetBattery

Функция возращает показание уровня заряда батареи в процентах.

```
function GetBattery();
```

Функция возвращает (float) текущее значение заряда батареи. Показывает 100% в случае ошибки.

Функция MinPowerLevel

Функция возращает значение критически значимого минимально уровень заряда батареи в процентах, заданного в hwconfig.txt

```
function MinPowerLevel();
```

Функция возвращает (float) параметр, заданный в виде константы в hwconfig.txt параметром POWER/MIN_POWER_LEVEL.

Функция IsChargeConnected

Функция возращает, подключена ли зарядка к роботу.

```
function IsChargeConnected();
```

Функция возвращает (bool) текущее состояние подключения зарядки. Следует отметить, что в случае любой ошибки функция возвращает false.

22.7. Управление сенсорами

Под сенсорами понимаются любые другие датчики, отличные от датчиков обратной связи и дальномером. Например, датчики заряда батареи или датчики температуры, датчики касания и прочие датчики, которыми может быть оснащен робот.

Функция GetSensor

Функция возвращает показание указанного логического сенсора.

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

Где:

sensor – (int) номер логического сенсора.

Функция возвращает показание (float) укаданного логического сенсора.

В случае ошибки, связанной с указанием несуществующего логического сенсора, функция возвращает 0. Если указан существующий логический сенсор, но возникает ошибка, то возвращает нулевое значение этого логического сенсора, заданное в hwconfig.txt.

Функция SetSensorTrigger или CreateSensorTrigger

Функция устанавливает триггер на указанный логический сенсор, который будет формировать событие в IScript3 при возникновении заданных условий.

Функции SetSensorTrigger и CreateSensorTrigger являются синонимами.

```
function SetSensorTrigger(sensor,
                          op,value1, value2,
                          event,minPeriod,repeatPeriod);
```

или

```
function SetSensorTrigger(sensor,
                          op,value1, value2,
                          event,minPeriod);
```

или

```
function SetSensorTrigger(sensor,
                          op,value1, value2,
                          event);
```

или

```
function SetSensorTrigger(sensor,
                          op,value1,
                          event,minPeriod,repeatPeriod);
```

или

```
function SetSensorTrigger(sensor, op,value1,
event,minPeriod);
```

или

```
function SetSensorTrigger(sensor,op,value1, event);
```

Где:

sensor – (int) номер логического сенсора.

op – (string) операция проверки условия срабатывания триггера:

- "EQ" или "=" – равенство показаний сенсора и value1.
- "NE" или "!=" – неравенство показаний сенсора и value1.
- "LT" или "<" – показаний сенсора строго меньше value1.
- "LE" или "<=" – показаний сенсора меньше или равно value1.
- "GT" или ">" – показаний сенсора строго больше value1.
- "GE" или ">=" – показаний сенсора больше или равно value1.

- "IN" – показаний сенсора в дипазоне от value1 по value2 включительно.
- "OUT" – показаний сенсора вне дипазона от value1 по value2.

value1 – (float) значение, с которым сравнивается показание датчика.

value2 – (float) значение конца диапазона, с которым сравнивается показание датчика (используется только для операций «IN» и «OUT»).

event – (string) строка с тектом события, которое формируется. Должно нажинаться со знаков «*» и «ПРОБЕЛ». Например: «* MYEVENT». Название события может быть любым, лишь бы скрипт был бы готов ловить это событие с помощью фреймовых структр.

minPeriod – (int) минимальный период (мс) срабатывания события. По умолчанию 0. Если этот период задать больше нуля, то, не смотря на то, что условия возникновения события соблюдаются, событие чаще указанного интервала не формируется. Особенно актуально для сломанных дребезжащих сенсоров.

repeatPeriod – (int) период (мс) повтора события. По умолчанию 0. Если этот период больше нуля, то пока условие триггера верно, событие формируется многократно с указанным периодом.

Функция возвращает (int) дескипор созданного сенсорного триггера или 0 в случае ошибки. Дескриптор может потребоваться для функции удаления данного сенсорного триггера.

Все созданные сенсорные триггеры уничтожаются с завершением работы скрипта.

Если minPeriod=0 и repeatPeriod=0, то событие триггера формируется лишь при первом соблюдении условия срабатывания триггера. Повторно событие не формируется, пока условие верно. Но если условие будет нарушено, триггер будет вновь взведен для формирования события, как только условие вновь станет верным.

Пример, пусть имеется логический сенсор 2, который показывает некую температуру:

```
frameset(event, 100) // фреймсет, которым будем ловить
события
{
  // ловим событие, которое будет формировать сенсорный
триггер
  frame("* HITEMP")
  {
     PlaySpeech("Температура слишком высока");
  }
}

// установим триггер на лог.сенсор 2, который в случае
показания
// датчика выше значения 60 будет формировать
// событие «* HITEMP» (название события мы придумали сами)
```

```
SetSensorTrigger(2, ">", 60, "* HITEMP");

// главный цикл скрипта
while(true)
{
   sync(); // синхронизация (чтобы события возникали)
}
```

Функция DeleteSensorTrigger, ClearSensorTrigger или RemoveSensorTriggger

Функция удаляет ранее установленный сенсорный триггер.

Функции DeleteSensorTrigger, ClearSensorTrigger и RemoveSensorTrigger являются синонимами.

```
function DeleteSensorTrigger(handle);
```

Где:

handle – дескриптор сенсорного триггера, полученный ранее с помощью функции SetSensorTrigger.

Функция ничего не возвращает.

22.8. Управление дальномерами

Функция GetRangefinder

Функция возвращает показание указанного логического дальномера.

```
function GetRangefinder(rfIndex);
```

Где:

rfIndex – номер логического дальномера.

Функция возвращает (float) показание дальномера в сантиметрах. В случае ошибки возвращает null.

Вызов функции «будит» дальномер, если он был в режиме ожидания.

Функция GetRangeFinder является синонимом функции GetRangefinder.

Функция GetSideRFDistance

Функция возвращает минимальное показание логического дальномера, расположенного с указанной стороны.

```
function GetSideRFDistance(side);
```

Где:

side – номер стороны:

(-1) – слева;
0 – спереди;
1 – справа;
2 – сзади.

Функция возвращает (float) минимальное показание дальномеров указаной стороны в сантиметрах. В случае ошибки возвращает null.

Вызов функции «будит» дальномеры указанной стороны, если они были в режиме ожидания.

Функция GetStairDetectorStopSignal

Функция возвращает стоп-сигнал дальномеров, используемых для обнаружения лестниц.

```
function GetStairDetectorStopSignal(dir);
```

или

```
function GetStairDetectorStopSignal();
```

Где:

dir – номер стороны:

1 – спереди (по умолчанию);

(-1) – сзади.

Функция возвращает (bool) признак срабатывания одного из дальномеров-датчиков обнаружения лестниц.

Вызов фунции «будит» дальномеры-датчики обнаружения лестиниц с указанной стороны, если они были в режиме ожидания.

22.9. Управление манипуляторами и жестами

Функция ArmGo

Функция управляет движением манипулятора arm с интерполяцией в обобщенной системе координат.

запрос, двигается ли рука arm:

```
function ArmGo(arm);
```

или

запустить жест с названием gesture на всех руках:

```
function ArmGo(gesture);
```

или

запустить жест с названием gesture на всех руках за указанное время

```
function ArmGo(gesture, time);
```

или

запустить жест с знаванием gesture только на указанной руке arm:

```
function ArmGo(arm, gesture);
```

или

запустить жест с знаванием gesture только на указанной руке arm за время time:

```
function ArmGo(arm, gesture, time);
```

или

запустить руку arm в положение, указанное массивом обобщенных координат jointArray, за время time:

```
function ArmGo(arm, jointArray, time);
```

или

```
function ArmGo(arm, jointArray);
```

или

запустить руку arm в положение, указанное точкой с мировыми координатами worldPoint, за время time:

```
function ArmGo(arm, worldPoint, time);
```

или

```
function ArmGo(arm, worldPoint);
```

Здесь:

motor – (int) номер логического мотора.

arm – (int) номер манипулятора.

gesture – (string) название жеста.

time – (int) время в мс, при значении 0 время (скорость движения) определяется драйвером по умолчанию.

jointArray – (array) массив обобщенных координат, соответствующих положению звеньев манипулятора.

worldPoint – (object) объект с координатами точки в мировой СК:

.x – координата X (см).

.y – координата Y (см).

.z – координата Z (см).

.yaw или .oz – угол рыскания (градусы).

.pitch или .oy – угол места (градусы).

.roll или .ox – угол крена (градусы).

Функция возвращает true в случае, если мотор находится в процессе движения.

Пример:

```
...
// подготовить точку p с мировыми координатами
p = object();
p.x = 60.0;
p.y = 40.0;
p.z = 5.0;
p.yaw = 0.0;
p.pitch = 0.0;
p.roll = 0.0;

// двигаться в точку p рукой 0 за 4000 мс
ArmGo(0, p, 4000);

// ждать, пока рука 0 двигается
while(ArmGo(0)) sync();

// идти рукой 0 в положение, определенное жестом «Супер
жест»
ArmGo(0, "Супер жест");

// ждать, пока рука 0 двигается
```

```
while(ArmGo(0)) sync();
```

Функция ArmGoL

Функция управляет движением манипулятора arm с интерполяцией в линейной мировой системе координат.

запрос, двигается ли рука arm:

```
function ArmGoL(arm);
```

или

запустить жест с названием gesture на всех руках:

```
function ArmGoL(gesture);
```

или

запустить жест с названием gesture на всех руках за указанное время

```
function ArmGoL(gesture, time);
```

или

запустить жест с знаванием gesture только на указанной руке arm:

```
function ArmGoL(arm, gesture);
```

или

запустить жест с знаванием gesture только на указанной руке arm за время time:

```
function ArmGoL(arm, gesture, time);
```

или

запустить руку arm в положение, указанное массивом обобщенных координат jointArray, за время time:

```
function ArmGoL(arm, jointArray, time);
```

или

```
function ArmGoL(arm, jointArray);
```

или

запустить руку arm в положение, указанное точкой с мировыми координатами worldPoint, за время time:

```
function ArmGoL(arm, worldPoint, time);
```

или

```
function ArmGoL(arm, worldPoint);
```

Здесь:

motor – (int) номер логического мотора.

arm – (int) номер манипулятора.

gesture – (string) название жеста.

time – (int) время в мс, при значении 0 время (скорость движения) определяется драйвером по умолчанию.

jointArray – (array) массив обобщенных координат, соответствующих положению звеньев манипулятора.

worldPoint – (object) объект с координатами точки в мировой СК:

.x – координата X (см).

.y – координата Y (см).

.z – координата Z (см).

.yaw или .oz – угол рыскания (градусы).

.pitch или .oy – угол места (градусы).
.roll или .ox – угол крена (градусы).

Функция возвращает true в случае, если мотор находится в процессе движения.

Пример:

```
...
// подготовить точку p с мировыми координатами
p = object();
p.x = 60.0;
p.y = 40.0;
p.z = 5.0;
p.yaw = 0.0;
p.pitch = 0.0;
p.roll = 0.0;

// двигаться в точку p рукой 0 за 4000 мс
ArmGoL(0, p, 4000);

// ждать, пока рука 0 двигается
while(ArmGo(0)) sync();
```

Следует отметить, что возможность линейной интерполяции поддерживается только, если сам драйвер (т.е. аппаратная реализация манипулятора) поддерживает процедуру линейной интерполяции.

Функция Gesture

Функция запускает жест с интерполяцией, указанной в самом жесте. При этом функция дожидается окончания движения.

Установить жест с названием gesture на всех руках со скоростью, установленной по умолчанию:

```
function Gesture(gesture);
```

или

Установить жест с названием gesture на всех руках за время time:

```
function Gesture(gesture, time);
```

или

Установить жест с названием gesture на руке всех руках за указанное время

```
function Gesture(arm, gesture);
```

или

Установить жест на руке arm с знаванием gesture, со скоростью, установленной по умолчанию:

```
function Gesture(arm, gesture);
```

или

Установить жест на руке arm с знаванием gesture за время time:

```
function Gesture(arm, gesture, time);
```

Здесь:

motor – (int) номер логического мотора.

arm – (int) номер манипулятора.
gesture – (string) название жеста.
time – (int) время в мс, при значении 0 время (скорость движения).

Функция возвращает true в случае успеха.

Функция дожидается, пока манипулятор придет в указанное положение, за это время работают все события и таймеры в IScript3.

Пример:

```
...
Gesture("Танец 1");  // установить жест
Gesture("Танец 2");  // установить жест
Gesture("Танец 1");  // установить жест
Gesture("Танец 2");  // установить жест
Gesture("Танец 3");  // установить жест
Gesture("Танец 4");  // установить жест
```

Следует отметить, что тип интерполяции при движении определяется типом интерполяции, указанной в настройке жестов.

Функция GestureJ

Функция запускает жест с интерполяцией в обобщенной системе координат, независимо от того, какая система координат была указана в самом жесте. При этом функция дожидается окончания движения.

Установить жест с названием gesture на всех руках со скоростью, установленной по умолчанию:

```
function GestureJ(gesture);
```

или

Установить жест с названием gesture на всех руках за время time:

```
function GestureJ(gesture, time);
```

или

Установить жест с названием gesture на руке всех руках за указанное время

```
function GestureJ(arm, gesture);
```

или

Установить жест на руке arm с знаванием gesture, со скоростью, установленной по умолчанию:

```
function GestureJ(arm, gesture);
```

или

Установить жест на руке arm с знаванием gesture за время time:

```
function GestureJ(arm, gesture, time);
```

Здесь:

motor – (int) номер логического мотора.
arm – (int) номер манипулятора.
gesture – (string) название жеста.
time – (int) время в мс, при значении 0 время (скорость движения).

Функция возвращает true в случае успеха.

Функция дожидается, пока манипулятор придет в указанное положение, за это время работают все события и таймеры в IScript3.

Пример:

```
...
```

```
GestureJ("Танец 1");  // установить жест
GestureJ("Танец 2");  // установить жест
GestureJ("Танец 1");  // установить жест
GestureJ("Танец 2");  // установить жест
GestureJ("Танец 3");  // установить жест
GestureJ("Танец 4");  // установить жест
```

Следует отметить, что тип интерполяции при движении всегда осуществляется в обобщенной системе координат, независимо от того, что было задано в параметрах жеста.

Функция GestureL

Функция запускает жест с интерполяцией в мировой системе координат, независимо от того, какая система координат была указана в самом жесте. При этом функция дожидается окончания движения.

Установить жест с названием gesture на всех руках со скоростью, установленной по умолчанию:

```
function GestureL(gesture);
```

или

Установить жест с названием gesture на всех руках за время time:

```
function GestureL(gesture, time);
```

или

Установить жест с названием gesture на руке всех руках за указанное время

```
function GestureL(arm, gesture);
```

или

Установить жест на руке arm с знаванием gesture, со скоростью, установленной по умолчанию:

```
function GestureL(arm, gesture);
```

или

Установить жест на руке arm с знаванием gesture за время time:

```
function GestureL(arm, gesture, time);
```

Здесь:

motor – (int) номер логического мотора.
arm – (int) номер манипулятора.
gesture – (string) название жеста.
time – (int) время в мс, при значении 0 время (скорость движения).

Функция возвращает true в случае успеха.

Функция дожидается, пока манипулятор придет в указанное положение, за это время работают все события и таймеры в IScript3.

Пример:

```
...
GestureL(0, "точка 1");  // установить жест
GestureL(0, "точка 2");  // установить жест
```

Следует отметить, что тип интерполяции при движении осуществляется линейная интерполяция, независимо от того, что было задано в параметрах жеста.

Однако линейная интерполяция должна поддерживается исключительно самим драйвером манипулятора, в противном случае интерполяция все равно осуществляется в обобщенной системе координат.

Функция RunGesture

Функция запускает жест с интерполяцией, указанной в самом жесте. При этом функция осуществляется асинхронно и не дожидается окончания движения.

Запустить жест с названием gesture на всех руках со скоростью, установленной по умолчанию:

```
function RunGesture(gesture);
```

или

Запустить жест с названием gesture на всех руках за время time:

```
function RunGesture(gesture, time);
```

или

Запустить жест с названием gesture на руке всех руках за указанное время

```
function RunGesture(arm, gesture);
```

или

Запустить жест на руке arm с знаванием gesture, со скоростью, установленной по умолчанию:

```
function RunGesture(arm, gesture);
```

или

Запутстить жест на руке arm с знаванием gesture за время time:

```
function RunGesture(arm, gesture, time);
```

Здесь:

motor – (int) номер логического мотора.
arm – (int) номер манипулятора.
gesture – (string) название жеста.
time – (int) время в мс, при значении 0 время (скорость движения).

Функция возвращает true в случае успеха.

Функция НЕ дожидается, пока манипулятор придет в указанное положение.

Пример:

```
...
RunGesture(0, "Танец 1");  // запустить жест
while(ArmGo(0))
{
   ... // делать что-то полезное, пока манипулятор
двигается
   sync();
}
PlaySpeech("Я пришёл");
```

Следует отметить, что тип интерполяции при движении определяется типом интерполяции, указанной в настройке жестов.

Функция RunGestureJ

Функция запускает жест с интерполяцией в обобщенной системе координат, независимо от того, какая система координат была указана в самом жесте. При этом функция НЕ дожидается окончания движения.

Запустить жест с названием gesture на всех руках со скоростью, установленной по умолчанию:

```
function RunGestureJ(gesture);
```

или

Запустить жест с названием gesture на всех руках за время time:

```
function RunGestureJ(gesture, time);
```

или

Запустить жест с названием gesture на руке всех руках за указанное время

```
function RunGestureJ(arm, gesture);
```

или

Запустить жест на руке arm с знаванием gesture, со скоростью, установленной по умолчанию:

```
function RunGestureJ(arm, gesture);
```

или

Запустить жест на руке arm с знаванием gesture за время time:

```
function RunGestureJ(arm, gesture, time);
```

Здесь:

motor – (int) номер логического мотора.
arm – (int) номер манипулятора.
gesture – (string) название жеста.
time – (int) время в мс, при значении 0 время (скорость движения).

Функция возвращает true в случае успеха.

Функция НЕ дожидается, пока манипулятор придет в указанное положение.

Пример:

```
...
RunGestureJ(0, "Танец 1");  // запустить жест
while(ArmGo(0))
{
   ... // делать что-то полезное, пока манипулятор
двигается
   sync();
}
PlaySpeech("Я пришёл");
```

Следует отметить, что тип интерполяции при движении всегда осуществляется в обобщенной системе координат, независимо от того, что было задано в параметрах жеста.

Функция RunGestureL

Функция запускает жест с интерполяцией в мировой системе координат, независимо от того, какая система координат была указана в самом жесте. При этом функция НЕ дожидается окончания движения.

Установить жест с названием gesture на всех руках со скоростью, установленной по умолчанию:

```
function RunGestureL(gesture);
```

или

Установить жест с названием gesture на всех руках за время time:

```
function RunGestureL(gesture, time);
```

или

Установить жест с названием gesture на руке всех руках за указанное время

```
function RunGestureL(arm, gesture);
```

или

Установить жест на руке arm с знаванием gesture, со скоростью, установленной по умолчанию:

```
function RunGestureL(arm, gesture);
```

или

Установить жест на руке arm с знаванием gesture за время time:

```
function RunGestureL(arm, gesture, time);
```

Здесь:

motor – (int) номер логического мотора.
arm – (int) номер манипулятора.
gesture – (string) название жеста.
time – (int) время в мс, при значении 0 время (скорость движения).

Функция возвращает true в случае успеха.

Функция НЕ дожидается, пока манипулятор придет в указанное положение

Пример:

```
...
RunGestureL(0, "Танец 1");  // запустить жест
while(ArmGo(0))
{
   ... // делать что-то полезное, пока манипулятор
двигается
   sync();
}
PlaySpeech("Я пришёл");
```

Следует отметить, что тип интерполяции при движении осуществляется линейная интерполяция, независимо от того, что было задано в параметрах жеста.

Однако линейная интерполяция должна поддерживается исключительно самим драйвером манипулятора, в противном случае интерполяция все равно осуществляется в обобщенной системе координат.

Функция GetGestureCoord

Получает координаты жеста.

```
function GetGestureCoord(arm, gesture);
```

или

```
function GetGestureCoord(gesture);
```

Здесь:

arm – (int) номер манипулятора, по умолчанию -1.

gesture – (string) название жеста.

Если жест в обобщенной системе координат, то функция возвращает массив (array) с обощенными координатами жеста.

Если жест в мировой системе координат, то функция возвращает объект (object) с полями {x,y,z,ox,oy,oz}. x,y,z – в сантиметрах, ox,oy,oz – в градусах.

В случае ошибки, например, жест не найден, возвращает null.

При вызове функции с одним параметром, или если arm < 0, то функция ищет жест по всем манипуляторам и работает с первым из них, в котором найден указанный жест.

Функция AddGesture

Функция добавляет жест.

```
function AddGesture(gesture);
```

или

```
function AddGesture(gesture, worldCoord);
```

или

```
function AddGesture(gesture, worldCoord, showOnMainScreen);
```

или

```
function AddGesture(arm, gesture);
```

или

```
function AddGesture(arm, gesture, worldCoord);
```

или

```
function AddGesture(arm, gesture, worldCoord,
showOnMainScreen);
```

Где:

gesture – (string) название жеста.

worldCoord – (bool) признак, что жест будет записан в мировых координатах (true). При значении false жест записывается в обобщенных координатах. По умолчанию false.

showOnMainScreen – (bool) признак, что данный жест будет отображаться на главном экране управления. По умолчанию false.

arm – (int) номер манипулятора.

Фунция возвращает true в случае успеха.

Функция записывает в качестве жеста текущее положение манипулятора arm. Если arm не указывается, то жест записывается по всем манипуляторам.

Если жест с указанным названием уже существует, то он перезаписывается.

Название жеста не должно содержать специальных символов @,#,=, табуляция, пененос строки.

Функция UpdateGesture

Функция обновляет положение жеста.

```
function UpdateGesture(gesture);
```

или

```
function UpdateGesture(gesture, worldCoord);
```

или

```
function UpdateGesture(arm, gesture);
```

или

```
function UpdateGesture(arm, gesture, worldCoord);
```

Где:

gesture – (string) название жеста.

worldCoord – (bool) признак, что жест будет записан в мировых координатах (true). При значении false жест записывается в обобщенных координатах. По умолчанию false.

arm – (int) номер манипулятора.

Фунция возвращает true в случае успеха.

Функция обновляет координаты в указанном жесте, но если жест не существует, то функция не добавляет его.

Если arm не указывается, то жест обновляется по всем манипуляторам.

Функция DeleteGesture

Функция удаляет жест.

```
function DeleteGesture(gesture);
```

или

```
function DeleteGesture(arm, gesture);
```

Где:

gesture – (string) название жеста.

arm – (int) номер манипулятора.

Фунция возвращает true в случае успеха.

Функция удаляет жест. Если arm не указывается, то жест удаляется по всем манипуляторам.

Функция GetGestureList

Функция получает список существующих жестов:

По всем рукам:

```
function GetGestureList();
```

или только тех, что на главном экране управления:

```
function GetGestureList(mainScreenOnly);
```

или только по указанной руке:

```
function GetGestureList(arm);
```

или только по указанной руке и тех, что есть на главном экране управления:

```
function GetGestureList(arm, mainScreenOnly);
```

Где:

mainScreenOnly – (bool) признак того, что требуется список только тех жестов, что отображаются на главном экране управления роботом, по умолчанию false.

arm – (int) Логический номер руки. По умолчанию для всех рук.

Функция вовзращает массив строк (array) с названием запрошенных жестов.

Функция RenameGesture

Функция переименовает жест и изменяет его свойства.

```
function RenameGesture(oldGesture,newGesture);
```

или

```
function RenameGesture(oldGesture,newGesture, showOnMain);
```

или

```
function RenameGesture(arm, oldGesture, newGesture);
```

или

```
function RenameGesture(arm, oldGesture, newGesture,
showOnMain);
```

Где:

oldGesture – (string) старое название жеста.
newGesture – (string) новое название жеста.

arm – (int) номер манипулятора.
showOnMain – (int или bool).
1 или true – отображать на главном экране.
0 или false – не отображать на главном экране.
-1 – не изменять.
По умолчанию (-1).

Фунция возвращает true в случае успеха.

Функция переименовывает жест, но не изменяет его координат. Если arm не указывается, то жест переименовывается по всем манипуляторам.

Функция SetJointSpeed

Функция задает скорость перемещения звеньев руки (манипулятора). Работает только с теми манипуляторами, которые управляются кинематическим устройством.

```
function SetJointSpeed(arm, speed);
```

или для скорости по умолчанию:

```
function SetJointSpeed(arm);
```

Где:

arm – (int) логический номер руки (манипулятора).
speed – (float) значение скорости или 0 для скорости по умолчанию.

Не возвращает ничего.

Функция SetJointAcc

Функция задает ускорение перемещения звеньев руки (манипулятора). Работает только с теми манипуляторами, которые управляются кинематическим устройством.

```
function SetJointAcc(arm,acc);
```

или для ускорения по умолчанию:

```
function SetJointAcc(arm);
```

Где:

arm – (int) логический номер руки (манипулятора).

acc – (float) значение ускорения или 0 для ускорения по умолчанию.

Не возвращает ничего.

Функция SetLinearSpeed

Функция задает скорость линейного перемещения руки (манипулятора). Работает только с теми манипуляторами, которые управляются кинематическим устройством.

```
function SetLinearSpeed(arm, speed);
```

или для скорости по умолчанию:

```
function SetLinearSpeed(arm);
```

Где:

arm – (int) логический номер руки (манипулятора).

speed – (float) значение скорости или 0 для скорости по умолчанию.

Не возвращает ничего.

Функция SetLinearAcc

Функция задает ускорение линейного перемещения руки (манипулятора). Работает только с теми манипуляторами, которые управляются кинематическим устройством.

```
function SetLinearAcc(arm,acc);
```

или для ускорения по умолчанию:

```
function SetLinearAcc(arm);
```

Где:

arm – (int) логический номер руки (манипулятора).

acc – (float) значение ускорения или 0 для ускорения по умолчанию.

Не возвращает ничего.

Функция IKAngularCritery

Функция задает или получает значение качества решения обратной задачи кинематики по углам ориентации (см. комментарий).

```
function IKAngularCritery(x, y, z);
```

или

```
function IKAngularCritery(xy, z);
```

или

```
function IKAngularCritery(xyz);
```

или для определения текущего значения критерия:

```
function IKAngularCritery();
```

Где:

x, y, z – (float) значение критерия по осям X, Y, Z соответственно.

xy, z – (float) значение критерия по осям X и Y.

xyz – (float) значение критерия единого по всем осям.

При вызове функции с параметрами функция задает значение критерия, при этом функция не возвращает ничего. При вызове без параметров функция возвращает массив, в котором элемент 0 соответствует значению критерия по оси X, элемент 1 – по оси Y, элемент 2 – по оси Z.

Программный комплекс «ДинРобот-3» решает обратную задачу кинематики модифицированным методом покоординатного спуска. Т.е. алгоритмы

программного комплекса многократно решают прямую задачу кинематики для манипулятора, подбирая такое положения моторов, которое позволяет достичь требуемого положения и ориентации рабочего органа манипулятора.

Качество решения ОЗК определяется критерием, выраженным в сантиметрах погрешности (углы ориентации из радиан также приводятся к сантиметрам через коэф. 40*AngularCritery).

В некоторых случаях пользователю может не так уж и важна ориентация рабочего органа по тем или иным осям, учитывая, что целевая точка может быть вообще недостижима в заданной ориентации. Например, чтобы взять со стола стаканчик руке робота не обязательно соблюдать ориентацию вокруг вертикальной оси – стаканчик круглый, его можно взять с любой стороны. Тем более что рука робота вообще может не иметь возможности достигнуть этого стаканчика в заданной ориентации.

Для этих целей служит данная функция, которая задает важность критерия соблюдения ориентации по отдельным осям мировой системы координат (связанной с центром робота). Значение 1.0 соответствует нормальной важности (1 градус погрешности по осям соответствует примерно 0.7 см линейного промаха). В примере со стаканчиком важность по осям X, Y можно было быть поставить равным 0, а по оси Z равной 1. Т.е. вызвать функцию с параметрами IKAngularCritery(0, 1.0);

Если соблюдение ориентации достаточно важно, можно задавать значения по осям больше 1, при условии, что при такой важности решение будет найдено.

При завершении работы скриптов востанавливается значение важности по умолчанию (задано в config.txt в параметрами IK/IK_ANGULAR_CRITERY_X, IK/IK_ANGULAR_CRITERY_Y, IK/IK_ANGULAR_CRITERY_Z).

Функция IKMinCritery и MinIKCritery

Функции IKMinCritery и MinIKCritery являются синонимами.

Функция задает или получает минимальное значение критерия качества решения обратной задачи кинематики.

```
function IKMinCritery(value);
```

или

```
function IKMinCritery();
```

Где:

value – (float) устаналиваемое значение (см).

Функция всегда возвращает текущее минимального значение качества решения обратной задачи кинематики.

Программный комплекс «ДинРобот-3» решает обратную задачу кинематики (ОЗК) модифицированным методом покоординатного спуска. Т.е. алгоритмы программного комплекса многократно решают прямую задачу кинематики для манипулятора, подбирая такое положения моторов, которое позволяет достичь требуемого положения и ориентации рабочего органа манипулятора.

Качество решения ОЗК определяется критерием, выраженным в сантиметрах погрешности (углы ориентации из радиан также приводятся к сантиметрам через коэф. 40*AngularCritery).

В процессе поиска значение критерия может получиться ниже некоторого порогового значения, заданного данной функцией. Такое решение можно считать уже приемлимым, и прекратить на этом дальнейший поиск лучшего решения, что сократит время расчета.

Обычно минимальное значение критерия поиска следует задавать равным 1 см. Однако пользователь может самостоятельно задать данное значение для кокретного случая.

Функция IKMaxCritery и MaxIKCritery

Функции IKMaxCritery и MaxIKCritery являются синонимами.

Функция задает или получает максимальное значение критерия качества решения обратной задачи кинематики.

```
function IKMaxCritery(value);
```

или

```
function IKMaxCritery();
```

Где:

value – (float) устаналиваемое значение (см).

Функция всегда возвращает текущее максимальное значение качества решения обратной задачи кинематики.

Программный комплекс «ДинРобот-3» решает обратную задачу кинематики (ОЗК) модифицированным методом покоординатного спуска. Т.е. алгоритмы программного комплекса многократно решают прямую задачу кинематики для манипулятора, подбирая такое положения моторов, которое позволяет достичь требуемого положения и ориентации рабочего органа манипулятора.

Качество решения ОЗК определяется критерием, выраженным в сантиметрах погрешности (углы ориентации из радиан также приводятся к сантиметрам через коэф. 40*AngularCritery).

Если по окончанию расчета значение критеря получается выше максимального критерия, то точка считается недостяжимой.

Обычно максимальное значение критерия поиска следует задавать равным 4 см. Однако пользователь может самостоятельно задать данное значение для кокретного случая.

Функция GetArmPose и ArmGetPose

Функции GetArmPose и ArmGetPose являются синонимами.

Функция возвращает текущие обобщенные координаты звеньев манипулятора в виде массива.

```
function GetArmPose(arm);
```

Где:

arm – (int) номер манипулятора.

Функция вовзращает массив (array), элементами которого являются обобщенные координаты (float) звеньев манипулятора. В массиве столько элементов, сколько моторов в указанной руке.

В случае ошибки возвращает null.

Результат функции совместим с требованиями второго аргумента функции ArmGo(arm, joints, time).

Функция ArmKinematic

Функция решает прямую задачу кинематики для руки манипулятора.

```
function ArmKinematic(arm, joints);
```

или

```
function ArmKinematic(arm);
```

Где:

arm – (int) номер манипулятора.

joints – (array) массив обобщенных координат, для которого нужно решить прямую задачу кинематики.

Функция вовзращает объект (object) со следующими полями:

- .x – (float) координата X конца манипулятора (см).
- .y – (float) координата Y конца манипулятора (см).
- .z – (float) координата Z конца манипулятора (см).
- .ox – (float) угол поворота (roll) вокруг оси X (градусы).
- .oy – (float) угол поворота (pitch) вокруг оси Y (градусы).
- .oz – (float) угол поворота (yaw) вокруг оси Z (градусы).

В случае ошибки возвращает null.

В случае вызова функции без указания параметра joints функция возвращает решение прямой задачи кинематики для текущего положения робота. В случае указания параметра joints решается прямая задача кинематики для конкретной точки.

Функция ArmIK

Функция решает обратную задачу кинематики для руки манипулятора.

```
function ArmIK(arm, point);
```

Где:

arm – (int) номер манипулятора.

point – (object) объект со следующими полями:

- .x – (float) координата X конца манипулятора (см).
- .y – (float) координата Y конца манипулятора (см).
- .z – (float) координата Z конца манипулятора (см).
- .ox или .roll – (float) угол поворота (roll) вокруг оси X (градусы).
- .oy или .pitch – (float) угол поворота (pitch) вокруг оси Y (градусы).
- .oz или .yaw – (float) угол поворота (yaw) вокруг оси Z (градусы).

В случае ошибки вызова возвращает null.

В случае невозможности решить ОЗК вовзрашает false.

В случае успеха функция вовзращает массив (array), элементами которого являются обобщенные координаты (float), соответствующие указанной точки. В массиве столько элементов, сколько моторов в указанной руке.

22.10. Управление камерами глубины

Функция PickDepth

Функция определяет координаты точки по камере глубины.

```
function PickDepth(camera,normX, normY);
```

где:

camera – (int) логический номер кармеры глубины.

normX – (float) нормированная шириной экрана экранная координата X (от 0 до 1, слева направо).

normY – (float) нормированная высотой экрана экранная координата Y (от 0 до 1, сверху вниз).

Возвращает объект со следующими полями:

.point2D – array(2) координаты X,Y указанной точки, совпадают с nornX, normY.

.local3D – array(3) координата X,Y,Z точки в локальной системе координат камеры, где ось X направлена вправо из центра камеры. Ось Y направлена вверх из цетра камеры. Ось Z направлена прямо вдоль оси визирования камеры.

.point3D – array(3) координаты X,Y,Z точки в системе координат робота, где ось X направлена вправо от центра робота, ось Y направлена вперед из центра робота по направлению движения, ось Z направлена вверх от пола.

В случае ошибки функция возвращает null. Ошибкой может стать, как отсутствие нужной камеры, так и то, что камера не является камерой глубины.

Вызов функции на время запускает указанную камеру, если она не была запущена.

Функция ожидает ответа от камеры. При этом в процессе ожидания параллельно работают таймеры и возникают события в IScript3.

Следует также отметить, что качество расчета координат точки point3D зависит от точности, с которой задано поле MATRIX в «hwconfig.txt» в секции [CAMERAx] данной камеры, а также поля AZIMUTH_PIVOT, PITCH_PIVOT и ROLL_PIVOT в секции [HEAD], при условии, что камера находится на голове.

Также важно, чтобы углы поворота головы определялись правильно.

22.11. Управление подсветкой

Функция SetLED

Функция устанавливает свечение указанной подсветки.

```
function SetLED(led,value, blink);
```

или

```
function SetLED(led,value);
```

где:

led – (int) логический номер подсветки.

value – (int) значение подсветки (определяется драйвером).

blink – (int) период мерцания (мс) или 0.

Возвращает true в случае успеха.