К-Софт
Инжиниринг

KSE Platform 3.4

Руководство разработчика: Описание API Lua мнемосхем

Документ описывает специализированные функции и свойства, доступные из кода мнемосхем.

ООО «К-СОФТ ИНЖИНИРИНГ»
+7 (812)383-56-14
support@k-soft-spb.ru

4-10-2021

Содержание

Глава 1. О документе

1. Настоящий документ предназначен для квалифицированных специалистов, обладающих базовыми знаниями в области программирования.
2. Документ описывает методы и свойства API Lua, доступные из кода мнемосхемы с примерами их использования. API Lua мнемосхемы позволяет получить доступ к серверным объектам и элементам самой мнемосхемы. При необходимости, с основами Lua можно ознакомиться здесь *http://www.lua.org/ ( )*
3. ООО "К-СОФТ Инжиниринг" оставляет за собой право на внесение изменений в настоящий документ в любое время. Если изменения будут носить масштабный характер, то они будут зафиксированы в очередном ReleaseNotes.
4. Вопросы по документу, а также запросы на техническую поддержку ПО можно отправить по адресу: support@k-soft-spb.ru.

ВАЖНО!

Внутренние пользователи ПО оформляют запросы в bitrix. Внешние - любым доступным способом (мессенджеры, электронная почта и т.д.)

Глава 2. Соглашения и условные обозначения, принятые в документе

В настоящем документе используются:

- Соглашения:

Меню, названия диалоговых окон и их свойства, названия документов, ключевые слова.	**Жирный шрифт**
Команды, примеры программ.	`Runtime.exe`
Имена файлов и пути.	*Курсив*
Ссылка на раздел настоящего документа (в скобках указан номер страницы).	*ссылка (2)*

- Условные обозначения:

	Информация обязательная для прочтения/выполнения.
	Отсылка к документу, который может содержать более полное описание изучаемой темы.

Глава 3. Общие рекомендации

ВАЖНО!

1. Рекомендуется избегать прямого взаимодействия с серверными объектами с мнемосхемы.
2. Не рекомендуется использовать глобальные переменные в коде мнемосхемы. Все переменные должны быть объявлены внутри реализуемых функций.
3. Некоторые свойства и методы исполняются только для мнемосхем, реализованных в виде диалоговых окон.
4. Изменения элементов и модели мнемосхемы, вызванные из кода мнемосхемы, исполняемой в виде диалогового окна, сохраняться не будут.

Глава 4. Контекстное меню

Методы:

Имя	Описание
AddButtonItem() (4)	Добавляет опцию контекстного меню в коллекцию объектов Context Menu элемента мнемосхемы или группы и возвращает в переменную добавленный объект
AddHeaderItem() (5)	Добавляет заголовок опций контекстного меню в коллекцию Context Menu элемента мнемосхемы или группы и возвращает добавленный объект в переменную
Clear() (5)	Удаляет все объекты из коллекции контекстного меню у элемента мнемосхемы или группы

AddButtonItem()

Синтаксис:

```
item = node.ContextMenu:AddButtonItem()
```

Результат:

Добавляет опцию контекстного меню в коллекцию объектов `ContextMenu` элемента мнемосхемы или группы в переменной `node` и возвращает в переменную `item` добавленный объект.

Также для доступа к объектам коллекции контекстного меню можно использовать имя объекта, например: `node.ContextMenu.MenuItem1` или `node.ContextMenu["MenuItem1"]`, где `MenuItem1` - имя объекта коллекции.

Пример использования:

```
function luanet.each(o)
   local e = o:GetEnumerator()
   return function()
      if e:MoveNext() then
         return e.Current
      end
   end
end
local login = Nodes.txtLogin
local error = Nodes.txtError
local users, err = Client:GetUsers()
if err ~= nil then
```

```
    error.Text = err
end
function Initializing()
    login.ContextMenu:Clear()
    local header = login.ContextMenu:AddHeaderItem()
    header.Name = 'Users'
    header.Caption = 'Пользователи'
    for u in luanet.each(users) do
        local item = login.ContextMenu:AddButtonItem()
        item.Caption = u.DisplayName
        item.Action.ActionTypeId = 8
        item.Action.Action.MethodToRun = 'SetLogin'
        item.Action.Action.Parameters:AddParameter(u.SymbolicName)
    end
end
function SetLogin(symbolicName)
    login.Text = symbolicName
end
function OnLoad()
    Initializing()
end
```

При вызове метода `Initializing` из метода `OnLoad` удаляются все объекты из коллекции объектов контекстного меню элемента мнемосхемы `txtLogin`, создается заголовок с именем `Users` и отображаемой подписью Пользователи. Затем добавляются опции контекстного меню, для каждой опции устанавливается название (отображаемое имя пользователя) и действие при выборе опции (запуск метода `SetLogin` мнемосхемы с параметром, в котором передается символьное имя пользователя). Метод `luanet.each` реализован для перебора коллекции пользователей, подробнее см. описание luanet.each.

AddHeaderItem()

Синтаксис:

```
header = node.ContextMenu:AddHeaderItem()
```

Результат:

Добавляет заголовок опций контекстного меню в коллекцию `ContextMenu` элемента мнемосхемы или группы в переменной `node` и возвращает добавленный объект в переменную `header`. Пример использования метода смотрите в описании метода *AddButtomItem() (4)*.

Clear()

Синтаксис:

```
node.ContextMenu:Clear()
```

Результат:

Удаляет все объекты из коллекции контекстного меню у элемента мнемосхемы или группы `node`.

Пример использования:

```
function Clear()
   local node = Nodes.Rectangle1
   node.ContextMenu:Clear()
end
```

Вызов метода `Clear` с мнемосхемы удаляет все объекты из коллекции контекстного меню элемента с именем `Rectangle1`.

Глава 5. Свойства и методы мнемосхем

Данный раздел включает в себя следующие свойства и методы мнемосхем:

- *для работы с размером, положением и масштабом мнемосхемы (7)*;
- *для работы с серверными объектами (13)*;
- *для автоматического запуска при наступлении определенного события (22)*.

Размер, положение и масштаб мнемосхемы

Свойства

Имя	Тип	Описание
HasOwnedWindows (7)	Boolean	Возвращает логическое значение с информацией о наличии дочерних окон (Platform3.3.35.xxxx)
Location (8)		Возвращает объект с координатами мнемосхемы
ViewMagnification (8)	Number	Возвращает текущее значение масштаба мнемосхемы
ViewOrigin (9)		Возвращает объект с координатами верхнего левого угла текущего вида мнемосхемы
ScreenBounds (10)		Возвращает объект с набором целых чисел, определяющих размер и положение экрана
Size (11)		Возвращает размер мнемосхемы в виде объекта

HasOwnedWindows

Синтаксис:

```
flag = Diagram.HasOwnedWindows
```

Результат:

Возвращает логическое значение. Если из мнемосхемы открыты диалоговые окна (через действия `OpenDiagram`, `ViewPdfFile`, `ShowLoginDialog`, `WriteTagValues`), вернет **true**, иначе **false**. Свойство доступно только на мнемосхемах,реализованных в виде диалоговых окон.

Пример использования:

```
function OnLostFocus()
 if Diagram.HasOwnedWindows then return end
```

```
 Diagram:Close()
end
```

Пример используется для обработки потери фокуса у мнемосхем, реализованных в виде диалоговых окон. Если при потере фокуса у диалогового окна нет дочерних окон, то мнемосхема закрывается.

Location

Синтаксис:

```
location = Diagram.Location
```

Результат:

Возвращает объект с координатами мнемосхемы. Изменение координат доступно только для мнемосхем, реализованных в виде диалогового окна (см. пример использования).

Аргументы:

Имя	Тип	Описание
`IsEmpty`	Boolean	Логическое значение, если значения `X` и `Y` равны 0, возвращает **true**, иначе **false**
`X`	Number	Координата мнемосхемы по оси `X`
`Y`	Number	Координата мнемосхемы по оси `Y`

Пример использования:

```
function MoveLocation()
   local location = Diagram.Location
   if not location.IsEmpty then
      location.X = location.X + 10
      location.Y = location.Y + 10
      <b>Diagram.Location</b> = location
   end
end
```

При каждом вызове метода `MoveLocation` из мнемосхемы, диалоговое окно мнемосхемы будет смещаться в правый нижний угол.

ViewMagnification

Синтаксис:

```
scale = Diagram.ViewMagnification
```

Результат:

Возвращает текущее значение масштаба мнемосхемы.

Прим.: *При установленном значении* ***true*** *свойства модели мнемосхемы* `BindMagnificationToWindowSize`*, изменения параметра* `Diagram.ViewMagnification` *из скрипта игнорируется.*

Пример использования:

```
function RandomMagnification()
   Diagram.ViewMagnification = math.random(10,150)
end
```

При вызове метода `RandomMagnification` мнемосхеме устанавливается масштаб от 10 до 150 в случайном порядке.

ViewOrigin

Синтаксис:

```
view = Diagram.ViewOrigin
```

Результат:

Возвращает объект с координатами верхнего левого угла текущего вида мнемосхемы (отображаемой области). Изменение координат видимой области применяется на мнемосхемах, которые не помещаются полностью на экран.

Аргументы:

Имя	Тип	Описание
`IsEmpty`	Boolean	Логическое значение, если значения `X` и `Y` равны 0, возвращает **true**, иначе **false**
`X`	Number	Координата мнемосхемы по оси `X`
`Y`	Number	Координата мнемосхемы по оси `Y`

Пример использования:

```
function ShowRectangle()
   local node = Nodes.Rectangle1
   local origin = Diagram.ViewOrigin
   origin.X = node.PinPoint.X
   origin.Y = node.PinPoint.Y
   Diagram.ViewOrigin = origin
end
```

При вызове метода `ShowRectangle` произойдет перемещение вида мнемосхемы к элементу с именем `Rectangle1`. Таким образом, элемент мнемосхемы отобразится в левом верхнем

углу (или максимально близко к левому верхнему углу, если положение элемента и масштаб/размер мнемосхемы не позволят выполнить перемещение вида). В данном примере не учитывается значение свойства элемента `PinPointOffset`. В зависимости от значения этого свойства положение элемента будет смещаться от установленных координат в `PinPoint`.

ScreenBounds

Синтаксис:

```
bounds = Diagram.ScreenBounds
```

Результат:

Возвращает объект с набором целых чисел, определяющих размер и положение экрана. Все свойства доступны только для чтения.

Аргументы:

Имя	Тип	Описание
`Bottom`	Number	Координата по оси `Y`, является суммой значений свойств `Y` и `Height`
`Height`	Number	Высота экрана
`IsEmpty`	Boolean	Логическое значение, если значения `X` и `Y` равны 0, возвращает **true**, иначе **false**
`Left`	Number	Координата по оси `X` левого края экрана
`Location`	Number	Координаты верхнего левого угла экрана
`Right`	Number	Координата по оси `X`, является суммой значений свойств `X` и `Width`
`Size`	Number	Размер экрана
`Top`	Number	Координата по оси `Y` верхнего края экрана
`Width`	Number	Ширина экрана
`X`	Number	Координата мнемосхемы по оси `X`
`Y`	Number	Координата мнемосхемы по оси `Y`

Пример использования:

```
function ShowScreenBounds()
   local bounds = Diagram.ScreenBounds
   print(bounds.IsEmpty)
```

```
    print(bounds.Top)
    print(bounds.Bottom)
    print(bounds.X)
    print(bounds.Y)
    print(bounds.Left)
    print(bounds.Right)
    print(bounds.Width)
    print(bounds.Height)
    print(bounds.Location.X)
    print(bounds.Location.Y)
    print(bounds.Size.Width)
    print(bounds.Size.Height)
end
```

При вызове метода `ShowScreenBounds` с мнемосхемы, выводятся значения всех свойств в окно `Output` среды разработки.

Size

Синтаксис:

```
size = Diagram.Size
```

Результат:

Возвращает размер мнемосхемы в виде объекта со свойствами.

Аргументы:

Имя	Тип	Описание
`IsEmpty`	Boolean	Логическое значение, если значения `X` и `Y` равны 0, возвращает **true**, иначе **false**
`Height`	Number	Высота мнемосхемы
`Width`	Number	Ширина мнемосхемы

Пример использования:

```
function ChangeSize()
    local size = Diagram.Size
    if size.IsEmpty then error("Has no size") end
    local temp = size.Height
    size.Height = size.Width
    size.Width = temp
    Diagram.Size = size
end
```

При каждом вызове метода `ChangeSize` на мнемосхеме, реализованной в виде диалогового окна, ширина и высота мнемосхемы будут меняться местами.

Методы

Метод	Описание
Close() (12)	Закрывает мнемосхему, открытую в виде диалогового окна
CloseAllFloatingWindows() (12)	Закрывает все диалоговые окна и процессы, запущенные с мнемосхем

Close()

Синтаксис:

```
Diagram:Close()
```

Результат:

Закрывает мнемосхему, открытую в виде диалогового окна.

Пример использования:

```
function Close()
   Diagram:Close()
end
```

Вызов метода с мнемосхемы закрывает диаграмму, открытую в виде диалогового окна.

CloseAllFloatingWindows()

Синтаксис:

```
Diagram:CloseAllFloatingWindows()
```

Результат:

Закрывает все диалоговые окна и процессы, запущенные с мнемосхем (мнемосхемы, открытые в виде диалоговых окон, внешние приложения, запущенные с мнемосхемы, окна с файлами `pdf`, открытые по действию `ViewPdfFile`)

Пример использования:

```
function CloseAll()
   Diagram:CloseAllFloatingWindows()
end
```

Вызов метода с мнемосхемы закрывает все диалоговые окна, открытые с мнемосхем.

Серверные объекты

Методы:

Метод	Описание
GetExpression() (13)	Возвращает в переменную выражение для свойства элемента мнемосхемы в виде строки
GetFolder() (14)	Возвращает папку в переменную
GetJob() (15)	Возвращает программу в переменную
GetJobFolder() (15)	Возвращает папку из корневого каталога Jobs
GetTag() (16)	Возвращает тег в переменную
GetTagFolder() (17)	Возвращает объект папку в переменную из корневого каталога Tags
GetUsers() (18)	Возвращает массив всех существующих объектов типа пользователь в переменную
SetExpression() (18)	Устанавливает выражение свойству элемента мнемосхемы
SetDisplayName() (19)	Устанавливает новое отображаемое имя для объекта (тег, папка или программа)
SetDescription() (19)	Устанавливает новое описание для объекта (тег, папка или программа)
Refresh() (20)	Обновляет значения свойств объекта (тег, папка или программа)
Log() (20)	Выводит сообщение Message в студии, в окно Output
GetBasePath() (21)	Позволяет получить базовый путь мнемосхемы (значение свойства BasePath у модели Model), результат возвращает в виде строки
GetConnectionState() (21)	Возвращает строку о состоянии соединения с сервером

GetExpression()

Синтаксис:

```
exp = node:GetExpression('propertyName')
```

Результат:

Возвращает в виде строки в переменную exp выражение для свойства `propertyName` элемента `node`. В случае ошибки возвращает **nil**.

Пример использования:

```
function GetSetExpression()
    local node = Nodes["Ellipse1"]
    local expression = node:GetExpression("FillStyle:Color")
    if expression ~= nil then
        node:SetExpression("LineStyle:LineColor", expression)
    end
end
```

При вызове метода `GetSetExpression` получаем объект мнемосхемы типа `Ellipse` в переменную `node`, получаем выражение для свойства `FillStyle:Color` в переменную `expression` и устанавливаем выражение для свойства `LineStyle:LineColor` этого объекта.

GetFolder()

Синтаксис:

```
subfolder, err = folder:GetFolder('SymbolicName')
```

Результат:

Возвращает папку в переменную `subfolder` с именем `SymbolicName` из папки `folder` и **nil** в переменную `err`. В случае ошибки возвращает в `err` сообщение об ошибке.

Пример использования:

```
function SetSubFolderProperties()
    local node = Nodes.TextNode1
    local root, err = Client:GetTagFolder('/')
    if err ~= nil then node.Text = err return end

    local folder, err = root:GetFolder('Project')
    if err ~= nil then node.Text = err return end

    node.Text = folder.SymbolicName.. '\n '
    node.Text = node.Text..folder.DisplayName.. '\n '
    node.Text = node.Text..folder.Description
end
```

При вызове метода `SetSubFolderProperties` получаем элемент мнемосхемы типа `TextNode` с именем `TextNode1`, получаем корневой каталог `Tags` в переменную `root`, получаем из корневого каталога папку с символьным именем `Project` и записываем ее свойства в свойство `Text` элемента мнемосхемы.

GetJob()

Синтаксис:

```
job, err = Client:GetJob('jobPath')
```

или

```
job, err = folder:GetJob('jobSymbolicName')
```

Результат:

Возвращает программу в переменную `job` по пути `jobPath` и **nil** в переменную `err` (или программу с символьным именем `jobSymbolicName` из каталога `folder`). В случае ошибки возвращает в `err` сообщение об ошибке.

Пример использования:

```
function StartJob()
    local txtErr = Nodes.txtError
    local job, err = Client:GetJob('/Test')
    if err ~= nil then txtErr.Text = err return end
        err = job:Start()
    if err ~= nil then txtErr.Text = err return end
end
```

При вызове метода `StartJob` получаем элемент мнемосхемы типа `TextNode` с именем `txtError`, получаем программу с символьным именем `Test` из корневого каталога `Jobs` и запускаем программу без каких-либо параметров, не ждем результат выполнения программы. В случае ошибок получения или запуска программы записываем сообщение об ошибке в свойство объекта мнемосхемы.

Пример использования метода `GetJob` из переменной типа каталог смотрите в описании метода *GetJobFolder() (15)*.

Прим.: *Начиная с версии KSE Platform 3.4.8, метод GetJob() не рекомендуется к использованию. Данное обновление не скажется на работоспособности ранних проектов, созданных с использованием указанного метода. Обратите внимание, что метод не подсвечивается в окне редактора кода.*

GetJobFolder()

Синтаксис:

```
folder, err = Client:GetJobFolder( 'folderPath ')
```

Результат:

Возвращает папку `folder` из корневого каталога `Jobs` по пути `folderPath` и **nil** в `err`. В случае ошибки возвращает в `err` сообщение об ошибке.

Пример использования:

```
function StartJobFromFolder()
    local node = Nodes.TextNode1
    local folder, err = Client:GetJobFolder( '/Project ')
    if err ~= nil then node.Text = err return end

    local job, err = folder:GetJob( 'Test ')
    if err ~= nil then node.Text = err return end

    err = job:Start()
    if err ~= nil then node.Text = err return end
end
```

При вызове метода мнемосхемы `StartJobFromFolder` получаем объект мнемосхемы типа `TextNode` в переменную `node`, получаем папку с символьным именем `Project` из корневого каталога `Jobs` в переменную `folder`. Получаем из папки `Project` программу с символьным именем `Test` в переменную `job` и запускаем программу.

Прим.: *Начиная с версии KSE Platform 3.4.8, метод GetJobFolder() не рекомендуется к использованию. Данное обновление не скажется на работоспособности ранних проектов, созданных с использованием указанного метода. Обратите внимание, что метод не подсвечивается в окне редактора кода.*

GetTag()

Синтаксис:

```
tag, err = Client:GetTag('tagPath')
```

или

```
tag, err = folder:GetTag('symbolicName')
```

Результат:

Возвращает тег в переменную `tag` по пути `tagPath` и **nil** в переменную `err` (или тег с символьным именем `symbolicName` из каталога `folder`). В случае ошибки возвращает в `err` сообщение об ошибке.

Пример использования:

```
function SetTagProperties()
    local node = Nodes.TextNode1
    local tag, err = Client:GetTag('/tag1')
    if err ~= nil then print(err) return end

    local value = tostring(tag.Value)..'\n'
    value = value..tag.StatusCode..'\n'
    value = value..tag.Timestamp:ToString()..'\n'
```

```
    value = value..tag.SymbolicName..'\n'
    value = value..tag.DisplayName..'\n'
    value = value..tag.Description
    node.Text = value
end
```

При вызове метода `SetTagProperties` мы получаем объект типа `TextNode`, получаем тег (в случае ошибки выводим на печать сообщение об ошибке и выходим из метода), собираем в переменную `value` все свойства тега с новой строки и записываем их в свойство `Text` переменной `node`.

Пример использования метода GetTag из переменной типа каталог смотрите в описании метода *GetTagFolder() (17)*.

GetTagFolder()

Синтаксис:

```
folder, err = Client:GetTagFolder('folderPath')
```

Результат:

Возвращает объект папку в переменную `folder` из корневого каталога `Tags` по пути `folderPath` и **nil** в переменную `err`. В случае ошибки возвращает в `err` сообщение об ошибке.

Пример использования:

```
function SetValueFromFolder()
    local node = Nodes.TextNode1
    local folder, err = Client:GetTagFolder('/')
    if err ~= nil then node.Text = err return end
    node.Text = folder.SymbolicName..'\n'
    node.Text = node.Text..folder.DisplayName..'\n'
    node.Text = node.Text..folder.Description..'\n'

    local tag, err = folder:GetTag('tag1')
    if err ~= nil then node.Text = err return end
    node.Text = node.Text..tag.SymbolicName..'\n'
    node.Text = node.Text..tag.DisplayName..'\n'
    node.Text = node.Text..tag.Description..'\n'
    node.Text = node.Text..tostring(tag.Value)
end
```

При вызове метода мнемосхемы `SetValueFromFolder` получаем объект мнемосхемы типа `TextNode` в переменную `node`, получаем корневой каталог `Tags` в переменную `folder` и устанавливаем в значение свойства `node.Text` символьное имя, отображаемое имя и описание каталога с новой строки.

Затем получаем тег с символьным именем `tag1` из переменной `folder` и добавляем значения свойств тега в свойство объекта мнемосхемы `node.Text`.

GetUsers()

Синтаксис:

```
users, err = Client:GetUsers()
```

Результат:

В случае успеха возвращает массив всех существующих объектов типа пользователь в переменную `users` и **nil** в переменную `err`. В случае ошибки возвращает в `err` сообщение об ошибке.

Пример использования:

```
local txtError = Nodes.txtError
local txtLogin = Nodes.txtLogin
local txtPassword = Nodes.txtPassword

function Login()
    local users, err = Client:GetUsers()
    if err ~= nil then txtError.Text = err return end
    for i = 0,users.Length-1 do
        if users[i].SymbolicName == txtLogin.Text then
            err = Runtime:Relogin(users[i], txtPassword.Text)
            if err == nil then
                txtError.Text = ' '
                Diagram:Close()
            else
                txtError.Text = err
            end
        end
    end
end
```

Пример предназначен для вызова на мнемосхеме, реализованной в виде диалогового окна. Перед вызовом метода получаем элементы мнемосхемы типа `TextNode` с именами `txtLogin`, `txtPassword`, `txtError`. При вызове метода `Login` получаем список всех существующих пользователей, перебираем в цикле полученных пользователей и проверяем символьное имя и значение свойства переменной `txtLogin.Text`. В случае совпадения вызываем метод перелогинивания и закрываем диалоговое окно при отсутствии ошибок.

SetExpression()

Синтаксис:

```
node:SetExpression('propertyName', 'exp')
```

Результат:

Устанавливает выражение `exp` свойству `propertyName` элемента `node`. Пример использования метода см. в описании метода *GetExpression() (13)* для элемента мнемосхемы.

SetDisplayName()

Синтаксис:

```
err = object:SetDisplayName(newName)
```

Результат:

Устанавливает новое отображаемое имя `newName` для `object`, где `object` - тег, папка или программа (job). В случае успеха возвращает **nil** в переменную `err`, в случае ошибки возвращает сообщение об ошибке.

Пример использования:

```
function SetPropertyValue()
    local node = Nodes.TextNode1
    local folder, err = Client:GetTagFolder('/Project')
    if err ~= nil then node.Text = err return end

    err = folder:SetDisplayName('Тестовый проект')
    if err ~= nil then node.Text = err return end
end
```

При вызове метода `SetPropertyValue` получаем элемент мнемосхемы типа `TextNode` с именем `TextNode1`, получаем папку с символьным именем `Project` из корневого каталога `Tags` и устанавливаем новое отображаемое имя Тестовый проект.

SetDescription()

Синтаксис:

```
err = object:SetDescription(newName)
```

Результат:

Устанавливает новое описание для `newName` для `object`, где `object` - тег, папка или программа (job). В случае ошибки возвращает в `err` сообщение об ошибке.

Пример использования:

```
function SetPropertyValue()
    local node = Nodes.TextNode1
    local folder, err = Client:GetTagFolder('/Project')
    if err ~= nil then node.Text = err return end
```

```
    err = folder:SetDescription('Тестовое описание проекта')
    if err ~= nil then node.Text = err return end
end
```

При вызове метода `SetPropertyValue` получаем элемент мнемосхемы типа `TextNode` с именем `TextNode1`, получаем папку с символьным именем `Project` из корневого каталога `Tags` и устанавливаем новое описание 'Тестовое описание проекта'.

Refresh()

Синтаксис:

```
err = object:Refresh('dDv')
```

Результат:

Обновляет значения свойств объекта `object`, где `object` - тег, папка или программа (job). Сокращения свойств:

- `d` - DisplayName,
- `D` - Description,
- `v` - Value (доступно только для тега).

В случае ошибки возвращает в `err` сообщение об ошибке. При получении серверного объекта все свойства уже обновлены, нет необходимости их обновлять.

Пример использования:

```
local tag, err = Client:GetTag('/tag1')
if err ~= nil then error(err) end

function RefreshValue()
    local node = Nodes.TextNode1
    local err = tag:Refresh('v')
    if err ~= nil then node.Text = err return end

    node.Text = tostring(tag.Value)
end
```

При вызове метода `RefreshValue` получаем объект мнемосхемы типа `TextNode` и обновляем значение тега в переменной `tag`. Обновленное значение записываем в свойство объекта мнемосхемы.

Log()

Синтаксис:

```
Log('Message')
```

Результат:

Метод аналогичен методу `print`. Выводит сообщение `Message` в студии, в окно `Output`. В среде исполнения результат выполнения метода недоступен.

Пример использования:

```
Log('Hello, world!')
```

В результате выполнения метода в студии, в окно `Output`, будет напечатано сообщение 'Hello, world!'. Аналогично вызову print('Hello, world!').

GetBasePath()

Синтаксис:

```
result = GetBasePath()
```

Результат:

Позволяет получить базовый путь мнемосхемы (значение свойства `BasePath` у модели `Model`), результат возвращает в виде строки: для значения свойства `Tags/` результат вызова /.

Пример использования:

```
local tagPath = GetBasePath()
local tag, err = Client:GetTag(tagPath.. 'tag1 ')
if err ~= nil then error(err) end
```

В примере получаем базовый путь и подставляем полученное значение в вызов метода `GetTag` плюс символьное имя тега. Таким образом, для мнемосхем с разными базовыми путями но одной структурой тегов реализация кода мнемосхемы будет неизменной.

GetConnectionState()

Результат:

Возвращает в переменную `state` строку с текущим состоянием соединения с сервером.Возможные варианты полученных значений:

- `Closed` - соединение с сервером закрыто;
- `Open` - соединение с сервером установлено;
- `Connecting` - устанавливается соединение с сервером;
- `Reconnecting` - устанавливается повторное соединение с сервером;
- `Fetching` - зарезервировано;
- `Broken` - ошибка соединения с сервером.

Пример использования:

```
local state = Client:GetConnectionState()
 if state == 'Open' then
  Nodes.TextNode1.Text = 'Подключено'
  Nodes.TextNode1.BackgroundStyle.Color = ColorFromArgb(255,0,255,0)
```

```
 end
end
```

В примере, приведенном выше, в переменную `state` получили строку со сведениями о текущем состоянии соединения с сервером, далее в зависимости от полученного значения присвоили свойству `Text` элемента мнемосхемы `TextNode1` значение `Подключено` и поменяли фоновый цвет в свойстве `BackgroundStyle`.

Методы мнемосхемы, вызываемые при наступлении какого-либо действия

Метод	Описание
OnClose() (22)	Вызывается при попытке закрыть мнемосхему, открытую в виде диалогового окна
OnGetFocus() (23)	Вызывается при активации мнемосхемы, реализованной в виде диалогового окна
OnLoad() (23)	Вызывается при запуске мнемосхемы, после того, как: все теги привязаны к элементам, их значения получены и все выражения выполнены
OnLostFocus() (24)	Вызывается при потере фокуса с окна мнемосхемы
OnMoving() (25)	Вызывается при перемещении окна мнемосхемы
OnSizing() (26)	Вызывается при изменении размера окна мнемосхемы
OnConnectionStateChanged() (27)	Вызывается при смене состояния подключения к серверу

OnClose()

Синтаксис:

```
function OnClose(e)
    e.Cancel = true
end
```

Результат:

Вызывается при попытке закрыть мнемосхему, открытую в виде диалогового окна. Функция содержит единственный параметр, в котором можно выставить флаг `Cancel = true` и отменить закрытие мнемосхемы.

Пример использования:

```
function OnClose(e)
    local node = Nodes['TextNode1']
    local value = tonumber(node.Text)
    if value ~= nil then
        if tonumber(node.Text) > 100 then
            e.Cancel = true
        end
    end
end
```

При попытке закрыть диалоговое окно мнемосхемы автоматически вызывается метод `OnClose`, в котором получаем объект мнемосхемы типа `TextNode`, приводим значение свойства к числовому типу и проверяем его. Если значение больше 100, то отменяем закрытие окна.

OnGetFocus()

Синтаксис:

```
function OnGetFocus()
    func1()
    func2()
    ...
end
```

Результат:

Вызывается при активации мнемосхемы, реализованной в виде диалогового окна. Таким образом, первый вызов метода осуществляется при открытии диалогового окна.

Пример использования:

```
function OnGetFocus()
    print(Diagram.Location:ToString())
    print(Diagram.Size:ToString())
end
```

В примере при активации диалогового окна выводим на печать положение и размер мнемосхемы в окно `Output` среды разработки.

OnLoad()

Синтаксис:

```
function OnLoad()
    func1()
    func2()
```

```
    ...
end
```

Результат:

Вызывается при запуске мнемосхемы, после того, как все теги привязаны к элементам, их значения получены и все выражения выполнены (при условии, что метод `OnLoad` объявлен).

Пример использования:

```
function OnLoad()
    for i = 1,10 do
        local node = Nodes["TextNode"..i]
        if node ~= nil then
            node.Text = "---"
        end
    end
end
```

Метод `OnLoad` вызывается при запуске мнемосхемы в виде вкладки в среде исполнения, либо при открытии мнемосхемы в виде диалогового окна (при условии, что метод объявлен в коде мнемосхемы). При вызове метода в цикле получаем элементы мнемосхемы с именами `TextNode1..10` и устанавливаем начальное значение "---" для свойства `Text`.

OnLostFocus()

Синтаксис:

```
function OnLostFocus()
    func1()
    func2()
    ...
end
```

Результат:

Метод доступен только для мнемосхем, реализованных в виде диалоговых окон (не модальных). Вызывается при потере фокуса с окна этой мнемосхемы (при условии, что метод `OnLostFocus` объявлен).

Прим.: *запрещается использовать внутри метода взаимодействие с сервером, т.к. это приведет к потере отзывчивости среды исполнения. Реализация метода должна быть простой, без выполнения долгих операций.*

Пример использования:

```
function OnLostFocus()
    Diagram:Close()
end
```

В примере закрываем окно мнемосхемы при потере фокуса.

OnMoving()

Синтаксис:

```
function OnMoving(bounds)
    func1()
    func2()
    ...
end
```

Результат:

Метод доступен только для мнемосхем, реализованных в виде диалоговых окон. Вызывается при перемещении окна этой мнемосхемы (при условии, что метод `OnMoving` объявлен).

В переменную `bounds` передаются координаты и размер окна в соответствующие свойства:

- `bounds.X`: координата по оси X;
- `bounds.Y`: координата по оси У;
- `bounds.Height`: высота окна;
- `bounds.Width`: ширина окна.

Пример использования:

```
local left = 100
local right = 700
local top = 200
local bottom = 600

function Limit(value, min, max)
    if value < min then return min end
    if value > max then return max end
    return value
end

function OnMoving(bounds)
    local x = Limit(bounds.X, left, right - bounds.Width)
    local y = Limit(bounds.Y, top, bottom - bounds.Height)
    return CreateBounds(x, y, bounds.Width, bounds.Height)
end
```

В примере ограничиваем возможность перемещения окна в пределах прямоугольника (100x200 и 700x600).

Методы `OnMoving` и `OnSizing` используются совместно для корректного поведения окна мнемосхемы, иначе при изменении размера или перетаскивании окна возможны «скачки» (резкие перемещения) окна мнемосхемы в выделенную прямоугольником область.

OnSizing()

Синтаксис:

```
function OnSizing(bounds)
    func1()
    func2()
    ...
end
```

Результат:

Метод доступен только для мнемосхем, реализованных в виде диалоговых окон. Вызывается при изменении размера окна этой мнемосхемы (при условии, что метод `OnSizing` объявлен).

В переменную `bounds` передаются координаты и размер окна в соответствующие свойства:

- `bounds.Left`: левая граница окна;
- `bounds.Right`: правая граница окна;
- `bounds.Top`: верхняя граница окна;
- `bounds.Bottom`: нижняя граница окна.

Пример использования:

```
local left = 100
local right = 700
local top = 200
local bottom = 600

function Limit(value, min, max)
    if value < min then return min end
    if value > max then return max end
    return value
end

function OnSizing(bounds)
    local l = Limit(bounds.Left, left, right)
    local r = Limit(bounds.Right, left, right)
    local t = Limit(bounds.Top, top, bottom)
    local b = Limit(bounds.Bottom, top, bottom)
    return CreateBounds(l, t, r - l, b - t)
end
```

В примере ограничиваем возможность изменения размера окна в пределах прямоугольника (100x200 и 700x600).

Методы `OnSizing` и `OnMoving` используются совместно для корректного поведения окна мнемосхемы, иначе при изменении размера или перетаскивании окна возможны «скачки» (резкие перемещения) окна мнемосхемы в выделенную прямоугольником область.

OnConnectionStateChanged()

Синтаксис:

```
OnConnectionStateChanged(originalstate, currentstate)
```

Результат:

Вызывается при смене состояния подключения к серверу.

Аргументы метода:

Имя	Тип	Описание
`originalstate`	String	Исходное состояние соединения с сервером
`currentstate`	String	Текущее состояние соединения с сервером

Возможные варианты значений `originalstate` и `currentstate`:

- `Closed` - соединение с сервером закрыто;
- `Open` - соединение с сервером установлено;
- `Connecting` - устанавливается соединение с сервером;
- `Reconnecting` - устанавливается повторное соединение с сервером;
- `Fetching` - зарезервировано;
- `Broken` - ошибка соединения с сервером.

Пример использования:

```
function OnConnectionStateChanged(originalstate, currentstate)
if currentstate == 'Open' then
  Nodes.TextNode1.Text = 'подключено'
  Nodes.TextNode1.BackgroundStyle.Color = ColorFromArgb(255,0,255,0)
  return
elseif currentstate == 'Reconnecting' then
  Nodes.TextNode1.Text = 'повторное подключение'
  Nodes.TextNode1.BackgroundStyle.Color = ColorFromArgb(100,255,180,0)
  return
end
  Nodes.TextNode1.Text = 'неизвестно'
```

```
  Nodes.TextNode1.BackgroundStyle.Color = ColorFromArgb(255,220,220,220)
end
```

В примере к зависимости от значения текущего состояния соединения с сервером `currentstate` меняем свойства текст (`Text`) и фон (`BackgroundStyle`) элемента мнемосхемы `TextNode`.

Глава 6. Свойства и методы для работы с событиями

Данный раздел включает в себя следующие свойства и методы для:

- *элемента мнемосхемы типа EventListControlNode (29)*;
- *элемента мнемосхемы типа EventListHistoryControlNode (29)*.

EventListControlNode

Методы:

Метод	Описание
EventListControl-Node:AckAllEvents() (29)	Квитирует все события от имени текущего пользователя в таблице элемента мнемосхемы типа EventListControlNode
EventListControl-Node:Clear() (30)	Удаляет все квитированные события из таблицы элемента мнемосхемы типа EventListControlNode
EventListControl-Node:ClearEvents() (30)	Удаляет все события из таблицы элемента мнемосхемы типа EventListControlNode

EventListControlNode:AckAllEvents()

Синтаксис:

```
EventListControlNode:AckAllEvents()
```

Результат:

Квитирует все события от имени текущего пользователя в таблице элемента мнемосхемы типа `EventListControlNode`.

Пример использования:

```
function AckAllEvents()
local eventList = Nodes['EventListControlNode']
   if eventList ~= nil then
      eventList:AckAllEvents()
   end
end
```

При вызове метода `AckAllEvents` с мнемосхемы, квитируются все события, отображенные в таблице элемента мнемосхемы `EventListControlNode`.

EventListControlNode:Clear()

Синтаксис:

```
EventListControlNode:Clear()
```

Результат:

Удаляет все квитированные события из таблицы элемента мнемосхемы типа `EventListControlNode`.

Пример использования:

```
function Clear()
   local eventList = Nodes['EventListControlNode']
   if eventList ~= nil then
      eventList:Clear()
   end
end
```

При вызове метода `Clear` из мнемосхемы, все квитированные события, отображенные в таблице элемента мнемосхемы `EventListControlNode`, будут удалены из таблицы.

EventListControlNode:ClearEvents()

Синтаксис:

```
EventListControlNode:ClearEvents()
```

Результат:

Удаляет все события из таблицы элемента мнемосхемы типа `EventListControlNode`.

Пример использования:

```
function ClearEvents()
   local eventList = Nodes['EventListControlNode']
   if eventList ~= nil then
      eventList:ClearEvents()
   end
end
```

При вызове метода `ClearEvents` из мнемосхемы, все события, отображенные в таблице элемента мнемосхемы `EventListControlNode`, будут удалены из таблицы.

EventListHistoryControlNode

Свойства

Имя	Тип	Описание
EventListHistory-ControlNode.Events-Count (31)	Number	Возвращает количество событий в таблице элемента мнемосхемы типа EventListHistoryControlNode

EventListHistoryControlNode.EventsCount

Синтаксис:

```
node.EventsCount
```

Результат:

Возвращает количество событий в таблице элемента мнемосхемы типа `EventListHistoryControlNode`. Переменная `node` - элемент мнемосхемы типа `EventListHistoryControlNode`.

Пример использования:

```
local list = Nodes.EventListHistoryControlNode1
local eventsCount = Nodes.TextNode1

function EventsCountChanged()
   eventsCount.Text = tostring(list.EventsCount)
end
```

При вызове метода `EventsCountChanged` в свойство `Text` элемента мнемосхемы `TextNode1` будет записано количество событий в таблице элемента мнемосхемы `EventListHistoryControlNode1`. Метод `EventsCountChanged` необходимо вызывать при изменении свойства `EventsCount` (EventsCount:Changed).

Методы

Метод	Описание
EventListControl-Node:AckAllEvents() (32)	Квитирует все события в таблице элемента мнемосхемы типа EventListHistoryControlNode
EventListHistory-ControlNode:Clear-Acked() (32)	Удаляет все квитированные события из таблицы элемента мнемосхемы типа EventListHistoryControlNode

Метод	Описание
EventListHistory-ControlNode:Clear-All() (33)	Удаляет все события из таблицы элемента мнемосхемы типа EventListHistoryControlNode
EventListControl-Node:Print() (33)	Отправляет на печать содержимое таблицы элемента мнемосхемы типа EventListHistoryControlNode
EventListControl-Node:Refresh() (33)	Обновляет список событий в таблице элемента мнемосхемы типа EventListHistoryControlNode
EventListControl-Node:ExportToExel() (34)	Сохраняет содержимое таблицы элемента мнемосхемы типа EventListHistoryControlNode в файл формата xlsx по пути filePath

EventListControlNode:AckAllEvents()

Синтаксис:

```
node:AckAllEvents()
```

Результат:

Квитирует все события в таблице элемента мнемосхемы типа `EventListHistoryControlNode`.

Пример использования:

```
local list = Nodes.EventListHistoryControlNode1
function AckAllEvents()
    list:AckAllEvents()
end
```

При вызове метода `AckAllEvents` с мнемосхемы, в таблице элемента типа `EventListHistoryControlNode` будут квитированны все события.

EventListHistoryControlNode:ClearAcked()

Синтаксис:

```
node:ClearAcked()
```

Результат:

Удаляет все квитированные события из таблицы элемента мнемосхемы типа `EventListHistoryControlNode`.

Пример использования:

```
local list = Nodes.EventListHistoryControlNode1
function ClearAcked()
```

```
    list:ClearAcked()
end
```

При вызове метода `ClearAcked` с мнемосхемы, из таблицы элемента мнемосхемы будут удалены все квитированные события.

EventListHistoryControlNode:ClearAll()

Синтаксис:

```
node:ClearAll()
```

Результат:

Удаляет все события из таблицы элемента мнемосхемы типа `EventListHistoryControlNode`.

Пример использования:

```
local list = Nodes.EventListHistoryControlNode1
function ClearAll()
    list:ClearAll()
end
```

При вызове метода `ClearAll` с мнемосхемы, из таблицы элемента мнемосхемы будут удалены все события.

EventListControlNode:Print()

Синтаксис:

```
node:Print(printerName)
```

Результат:

Отправляет на печать содержимое таблицы элемента мнемосхемы типа `EventListHistoryControlNode`.

Пример использования:

```
local list = Nodes.EventListHistoryControlNode1
function Print()
   list:Print( 'HP LaserJet 1020 ')
end
```

При вызове метода `Print` содержимое таблицы элемента мнемосхемы в переменной `list` отравляется для печати на принтер с именем HP LaserJet 1020.

EventListControlNode:Refresh()

Синтаксис:

```
node:Refresh(fromTicks, toTicks, source)
```

Результат:

Обновляет список событий в таблице элемента мнемосхемы типа `EventListHistoryControlNode`. Переменная `node` - элемент мнемосхемы типа `EventListHistoryControlNode`, аргумент `fromTicks` - начальная дата и время в тиках, `toTicks` - конечная дата и время в тиках, `source` - пусть к источнику событий в виде строки (например, для корневого каталога значение будет «Tags»).

Пример использования:

```
local list = Nodes.EventListHistoryControlNode1
local dateFrom = Nodes.DatePickerControlNode1
local dateTo = Nodes.DatePickerControlNode2
local timeFrom = Nodes.TimePickerControlNode1
local timeTo = Nodes.TimePickerControlNode2

function RefreshList()
   local startTicks = dateFrom.DateTicks + timeFrom.TimeOfDayTicks
   local endTicks = dateTo.DateTicks + timeTo.TimeOfDayTicks
   local source = 'Tags '
   list:Refresh(startTicks, endTicks, source)
end
```

При вызове метода `RefreshList` с мнемосхемы, в таблицу элемента в переменной `list` будут загруженны все события за период, указанный в элементах мнемосхемы из переменных `dateFrom + timeFrom` и `dateTo + timeTo`.

EventListControlNode:ExportToExel()

Синтаксис:

```
node:ExportToExel(filePath)
```

Результат:

Сохраняет содержимое таблицы элемента мнемосхемы типа `EventListHistoryControlNode` в файл формата `xlsx` по пути `filePath`.

Пример использования:

```
function Export()
local fileName = 'C:/Desktop/ELHC.xlsx'
   list:ExportToExel(fileName)
end
```

При вызове метода `ExportToExel` с мнемосхемы, содержимое таблицы элемента мнемосхемы в переменной `list` сохраняется в файл с расширением `xlsx` в каталоге `C:/Desktop`.

Глава 7. Свойства и методы для работы с алармами

Метод	Описание
AlarmListControl-Node:AckAllAlarms()	Квитирует все алармы в таблице элемента мнемосхемы типа AlarmListControlNode

AlarmListControlNode:AckAllAlarms()

Синтаксис:

```
AlarmtListControlNode:AckAllAlarms()
```

Результат:

Квитирует все алармы от имени текущего пользователя в таблице элемента мнемосхемы типа `AlarmListControlNode`.

Пример использования:

```
function AckAllAlarms()
  local node = Nodes['AlarmListControlNode1']
  node:AckAllAlarms('Ack')
end
```

При вызове метода `AckAllAlarms` с мнемосхемы, квитируются все алармы, отображенные в таблице элемента мнемосхемы `AlarmListControlNode`.

> **ВАЖНО!**
>
> Аргумент передаваемый в скобках не может быть пустым, скобки обязательно должны быть заполнены

Глава 8. Элементы мнемосхемы

Свойства

Имя	Описание
Nodes[] (36)	Возвращает элемент мнемосхемы с именем NodeName
ExpressionBag (37)	Возвращает выражение для свойства Property элемента node
Properties[] (37)	Возвращает пользовательское свойство модели мнемосхемы с именем CustomPropertyName

Nodes[]

Синтаксис:

```
node = Nodes.NodeName
```

или

```
node = Nodes[NodeName]
```

Результат:

Возвращает элемент мнемосхемы с именем `NodeName`. В случае ошибки возвращает **nil**.

Пример:

```
node = Nodes.Group1.Nodes.SubGroup2.Nodes.TextNode2
```

В примере показано, что ключевое слово `Nodes` указывается после каждой группы для доступа к элементам внутри группы.

Пример использования:

```
local txt = Nodes.TextNode1
local rectangle = Nodes["Rectangle1"]
function Change()
    txt.Text = 'Test '
    rectangle.FillStyle.Color = ColorFromArgb(255, 255, 0, 0)
end
```

В примере получаем текстовый элемент с именем `TextNode1` и элемент прямоугольник с именем `Rectangle1`. В методе `Change` меняем свойства полученных элементов. Для представления цвета в нужном формате используется метод `ColorFromArgb()`.

ExpressionBag

Синтаксис:

```
node.ExpressionBag['Property']
```

Результат:

Возвращает выражение для свойства `Property` элемента `node`.

Пример:

```
node.ExpressionBag[’FontColorStyle:Color’].
```

Пример использования:

```
local txt = Nodes.TextNode1
function SetNodeExpression()
    print(txt.ExpressionBag['Text'])
    txt.ExpressionBag['Text'] = "Iif([/Tags/flag] == 1,'On','Off')"
end
```

В примере получаем текстовый элемент с именем `TextNode1` и в методе `SetNodeExpression` сначала выводим текущее выражение для свойства `Text` элемента `txt`, затем устанавливаем новое выражение для этого свойства. Пример установки выражения для свойства аналогичен вызову метода `SetExpression()` у объекта `txt` (подробнее см. в описании метода `SetExpression()`).

Properties[]

Синтаксис:

```
property = Properties.CustomPropertyName
```

или

```
property = Properties[CustomPropertyName]
```

Результат:

Возвращает пользовательское свойство модели мнемосхемы с именем `CustomPropertyName`. В случае ошибки возвращает **nil**.

Пример использования:

```
function SetProperties()
    local p = Properties.Color
    p.Value = 'Pink '
    p.DefaultValue = 'Red '
    p.Handler = 'CustomHandler '
    Properties["Color"] = p
end
```

При вызове метода `SetProperties` с мнемосхемы мы получаем свойство `Color` в переменную `p`. Затем в переменной `p` меняем доступные параметры объекта:

- `Value` - значение свойства;
- `DefaultValue` - значение свойства по умолчанию;
- `Handler` - метод-обработчик, запускаемый при изменении свойства (lua-скрипт в мнемосхеме);

В конце метода присваиваем свойству `Color` изменный объект переменной `p`.

Методы

Метод	Описание
BringToFront() (38)	Перемещает элемент мнемосхемы на передний план
SendToBack() (38)	Перемещает элемент мнемосхемы на задний план
ColorFromArgb() (39)	Возвращает цвет в формате, применимом к свойствам объекта мнемосхемы
Model.Properties:AddNew() (39)	Добавляет новое свойство в коллекцию пользовательских свойств модели мнемосхемы Properties и возвращает добавленный объект

BringToFront()

Синтаксис:

```
BringToFront(Nodes.TextNode1)
```

Результат:

Перемещает элемент мнемосхемы на передний план.

Пример использования:

```
function ToFront()
    local node = Nodes['Ellipse1']
    BringToFront(node)
end
```

При вызове метода `ToFront` получаем объект мнемосхемы с именем `Ellipse1` и перемещаем его на передний план.

SendToBack()

Синтаксис:

```
SendToBack(Nodes.TextNode1)
```

Результат:

Перемещает элемент мнемосхемы на задний план.

Пример использования:

```
function ToBack()
    local node = Nodes['Ellipse1']
    SendToBack(node)
end
```

При вызове метода `ToBack` получаем объект мнемосхемы с именем `Ellipse1` и перемещаем его на задний план.

ColorFromArgb()

Синтаксис:

```
color = ColorFromArgb(A, R, G, B)
```

Результат:

Возвращает цвет в формате, применимом к свойствам объекта мнемосхемы. Аргументы метода принимают значения от 0 до 255:

- `A` (alpha) - прозрачность;
- `R` (red) - красный;
- `G` (green) - зеленый;
- `B` (blue) - синий.

Пример использования:

```
function Paint()
    Nodes["Ellipse1"].FillStyle.Color = ColorFromArgb(255, 0, 255, 255)
    Nodes.RoundRect1.LineStyle.LineColor = ColorFromArgb(255, 255, 255, 0)
end
```

При вызове метода `Paint` устанавливаем цвет морской волны для элемента мнемосхемы с именем `Ellipse1` и желтый цвет для линий элемента с именем `RoundRect1`.

Model.Properties:AddNew()

Синтаксис:

```
Model.Properties:AddNew( 'propertyName ')
```

Результат:

Добавляет новое свойство в коллекцию пользовательских свойств модели мнемосхемы `Properties` и возвращает добавленный объект.

Пример использования:

```
function AddProperty()
    local p = Model.Properties:AddNew('Color')
    p.Value = 'Blue'
    p.DefaultValue = 'Red'
end
```

Вызов метода `AddProperty` добавляет новое свойство с именем `Color` в коллекцию пользовательский свойств модели и устанавливает значение в `Value` и значение по умолчанию в `DefaultValue`.

Глава 9. Среда исполнения Runtime

Методы:

Метод	Описание
GetUsers() (41)	Возвращает массив объектов типа пользователь, добавленных в среду исполнения
GoDiagram() (42)	Активирует вкладку с мнемосхемой, хранимой по абсолютному пути
Logout() (42)	Завершает работу среды исполнения
Relogin() (43)	Позволяет авторизоваться в среде исполнения
ReloginBySymbolicName() (43)	Позволяет авторизоваться в среде исполнения под пользователем с символьным именем и паролем

GetUsers()

Синтаксис:

```
users, err = Runtime:GetUsers()
```

Результат:

В случае успеха возвращает массив объектов типа пользователь, добавленных в среду исполнения, в переменную `users` и **nil** в переменную `err`. В случае ошибки возвращает в `err` сообщение об ошибке.

Пример использования:

```
local txtError = Nodes.txtError
local txtLogin = Nodes.txtLogin
local txtPassword = Nodes.txtPassword

function Login()
  local users, err = Runtime:GetUsers()
  if err ~= nil then txtError.Text = err return end
  for i = 0,users.Length-1 do
    if users[i].SymbolicName == txtLogin.Text then
      err = Runtime:Relogin(users[i], txtPassword.Text)
      if err == nil then
        txtError.Text = ' '
        Diagram:Close()
      else
        txtError.Text = err
```

```
      end
    end
  end
end
```

Пример предназначен для вызова на мнемосхеме, реализованной в виде диалогового окна. Перед вызовом метода получаем элементы мнемосхемы типа `TextNode` с именами `txtLogin`, `txtPassword`, `txtError`. При вызове метода `Login` получаем список всех пользователей среды исполнения, перебираем в цикле полученных пользователей и проверяем символьное имя и значение свойства переменной `txtLogin.Text`. В случае совпадения вызываем метод перелогинивания и закрываем диалоговое окно при отсутствии ошибок.

GoDiagram()

Синтаксис:

```
Runtime:GoDiagram('path')
```

Результат:

Активирует вкладку с мнемосхемой, хранимой по абсолютному пути `path`, к примеру: `Runtime:GoDiagram('/Diagrams/Diagram1')` активирует вкладку с мнемосхемой `Diagram1`, если мнемосхема отсутствует в списке или путь указан неверно, откроется окно с сообщением об ошибке.

Пример использования:

```
function GoToDiagram()
  Runtime:GoDiagram('/Diagrams/Project/Test')
end
```

При вызове метода `GoToDiagram` в среде исполнения происходит переход к вкладке с мнемосхемой по пути `/Diagrams/Project/Test` (при наличии мнемосхемы в среде исполнения).

Logout()

Синтаксис:

```
err = Runtime:Logout(showConfirmationDialog)
```

Результат:

Вызов метода завершает работу среды исполнения. Параметр showConfirmationDialog - логическая переменная, если указать true - при вызове метода появится диалоговое окно для подтверждения операции, если указать false - выход пользователя из системы осуществится без запроса подтверждения.

Пример использования:

```
local txtError = Nodes.TextNode1
function CloseRuntime()
  local err = Runtime:Logout(true)
  if err ~= nil then
    txtError.Text = err
  end
end
```

При вызове метода `CloseRuntime` в среде исполнения всплывает диалоговое окно для подтверждения действия. После подтверждения операции происходит выход пользователя из среды исполнения.

Relogin()

Синтаксис:

```
err = Runtime:Relogin(user,'password')
```

Результат:

Вызов метода позволяет авторизоваться в среде исполнения. Параметр `user` - элемент массива, полученного методами `Client:GetUsers()` или `Runtime:GetUsers()`, строковый параметр `password` - пароль для пользователя `user`. В случае успеха возвращает **nil** в переменную `err` и авторизует пользователя в среде исполнения, иначе возвращает сообщение об ошибке в `err`.

Пример использования:

```
local err = Runtime:Relogin(user,pass))
if err == nil then
txtError.Text = ''
else
txtError.Text = err
end
```

> **ВАЖНО!**
>
> При использовании мнемосхемного метода `Runtime:Relogin(user,pass)`, не рекомендуется в этой же части кода использовать `Diagram:Close()`, т.к. последнее - избыточно (при перелогине закрываются плавающие мнемосхемы и загружаются новые для пользователя, если такие есть).

ReloginBySymbolicName()

Синтаксис:

```
err = Runtime:ReloginBySymbolicName('userSymbolicName','password')
```

Результат:

Вызов метода позволяет авторизоваться в среде исполнения под пользователем с символьным именем `userSymbolicName` и паролем `password`. В случае успеха возвращает **nil** в переменную `err` и авторизует пользователя в среде исполнения, иначе возвращает сообщение об ошибке в `err`.

Пример использования:

```
function ReloginByName()
  local err = Runtime:ReloginBySymbolicName(Nodes.TextNode1.Text, "")
  if err ~= nil then Nodes.TextNode1.Text = err end
end
```

При вызове метода `ReloginByName` произойдет авторизация под пользователем, символьное имя которого указанно в свойстве `Text` элемента мнемосхемы `TextNode1`. Обратите внимание, что пароль указан пустой.

Прим.: *Метод доступен с версии 3.3.41.xxxx-Release.*

Глава 10. Теги

Свойства

Имя	Описание
Value (45)	Возвращает значение тега
StatusCode (45)	Возвращает код статуса значения
Timestamp (45)	Возвращает отметку времени получения значения тега

Value

Синтаксис:

```
tag.Value
```

Результат:

Возвращает значение тега.

Пример использования: смотрите в описании метода `GetTag()`.

StatusCode

Синтаксис:

```
tag.StatusCode
```

Результат:

Возвращает код статуса значения.

Пример использования: смотрите в описании метода `GetTag()`.

Timestamp

Синтаксис:

```
tag.Timestamp
```

Результат:

Возвращает отметку времени получения значения тега.

Пример использования: смотрите в описании метода `GetTag()`.

Примеры работы с timestamp:

Выражение	Результат
`FormatString('{0:y yy yyy yyyy}',[/Tags/tag:Timestamp])`	'18 18 2018 2018' - **год**
`FormatString('{0:M MM MMM MMMM}',[/Tags/tag:Timestamp])`	'9 09 Sep September' - **месяц**
`FormatString('{0:d dd ddd dddd}',[/Tags/tag:Timestamp])`	'11 11 Tue Tuesday' - **день месяца\недели**
`FormatString('{0:h hh H HH}',[/Tags/tag:Timestamp])`	'8 08 20 20' - **час 12\24**
`FormatString('{0:m mm}',[/Tags/tag:Timestamp])`	'2 02' - **минуты**
`FormatString('{0:s ss}', [/Tags/tag:Timestamp])`	'2 02' - **секунды**
`FormatString('{0:f ff fff ffff}',[/Tags/tag:Timestamp])`	'9 98 985 9850' - **мс**
`FormatString('{0:F FF FFF FFFF}',[/Tags/tag:Timestamp])`	'9 98 985 985' - **мс без нуля**
`FormatString('{0:t tt}', [/Tags/tag:Timestamp])`	'P PM' - **время AM\PM**
`FormatString('{0:z zz zzz}',[/Tags/tag:Timestamp])`	'+3 +03 +03:00' - **часовой пояс**

Методы

Метод	Описание
SetValue() (46)	Устанавливает значение со статусом тегу tag

SetValue()

Синтаксис:

```
err = tag:SetValue(newValue, statusCode)
```

Результат:

Устанавливает значение `newValue` со статусом `statusCode` тегу `tag`. Аргумент `statusCode` необязательный (по умолчанию код статуса равен нулю). В случае успеха в `err` возвращает **nil**, иначе сообщение об ошибке.

Пример использования:

```
function SetTagValue()
    local node = Nodes.TextNode1
```

```
    local tag, err = Client:GetTag('/tag1')
    err = tag:SetValue(100)
    if err ~= nil then node.Text = err end
end
```

В примере при вызове метода мнемосхемы `SetTagValue` получаем элемент мнемосхемы типа `TextNode` и тег. Затем записываем в тег значение 100. В случае ошибки записываем сообщение об ошибке в свойство объекта мнемосхемы.

Глава 11. Раскладка клавиатуры

Свойства

Имя	Описание
InputLanguage.Current (48)	Возвращает значение текущей раскладки клавиатуры в виде объекта
InputLanguage.Default (49)	Возвращает значение раскладки клавиатуры установленное по умолчанию в виде объекта
InputLanguage.Installed (49)	Возвращает массив всех установленных языков ввода в виде объектов

InputLanguage.Current

Синтаксис:

```
language = InputLanguage.Current
```

Результат:

Возвращает значение текущей раскладки клавиатуры в виде объекта со свойствами:

- `Name` - название раскладки клавиатуры (например, ru, en, и так далее);
- `Layout` - название раскладки клавиатуры, отображаемое в региональных настройках операционной системы на компьютере (например, РУС, США, и так далее);
- `Handle` - номер дескриптора для языка ввода в виде строки.

Пример использования:

```
local current = InputLanguage.Current
local default = InputLanguage.Default
if current.Name ~= default.Name then
    local err = InputLanguage:SetCurrent(default)
        if err ~= nil then print(err) return end
end
```

В примере получаем текущее значение раскладки клавиатуры и значение по умолчанию. Затем сравниваем их свойства и если они отличаются, меняем значение раскладки клавиатуры на значение по умолчанию.

InputLanguage.Default

Синтаксис:

```
language = InputLanguage.Default
```

Результат:

Возвращает значение раскладки клавиатуры установленное по умолчанию в виде объекта со свойствами `Name`, `Layout` и `Handle`. Описание свойств и пример использования смотрите в *InputLanguage.Current (48)*.

InputLanguage.Installed

Синтаксис:

```
languages = InputLanguage.Installed
```

Результат:

Возвращает массив всех установленных языков ввода в виде объектов со свойствами `Name`, `Layout` и `Handle`. Описание свойств смотрите в *InputLanguage.Current (48)*.

Пример использования:

```
for language in each(InputLanguage.Installed) do
if language.Name == "ru" then
    local err = InputLanguage:SetCurrent(language)
        if err ~= nil then print(err) end
    end
end
```

В примере с помощью функции `each` перебираем все установленные в системе языки ввода. Если название языка ввода соответствует `ru`, то устанавливаем этот язык в качестве текущего языка ввода.

Методы

Метод	Описание
InputLanguage:SetCurrent() (49)	Возвращает массив всех установленных языков ввода в виде объектов

InputLanguage:SetCurrent()

Синтаксис:

```
err = InputLanguage:SetCurrent(language)
```

Результат:

Устанавливает текущее значение раскладки клавиатуры, равное `language`. Где `language` - объект языка ввода, полученный с помощью *InputLanguage.Current (48)*, *InputLanguage.Default (49)* или элемент массива *InputLanguage.Installed (49)*. В случае успеха возвращает **nil**, в случае ошибки возвращает в `err` сообщение об ошибке. Пример использования смотрите в описании *InputLanguage.Current (48)* и *InputLanguage.Installed (49)*.

Глава 12. Воспроизведение звука

Свойства

Имя	Описание
Sound.LoopingFile-Name (51)	Содержит имя файла, который циклически воспроизводится функцией StartPlay()

Sound.LoopingFileName

Синтаксис:

```
path = Sound.LoopingFileName
```

Результат:

Свойство `Sound.LoopingFileName` содержит имя файла, который циклически воспроизводится функцией `StartPlay()`. Если ничего не воспроизводится, значение равно пустой строке.

Пример использования:

```
function StopPlaySound()
    if Sound.LoopingFileName ~= ' ' then
        local err = Sound:StopPlay()
        if err ~= nil then print(err) end
    end
end
```

При вызове метода `StopPlaySound()` проверяем значение свойства `Sound.LoopingFileName` и, если значение не равно пустой строке, вызываем метод остановки цикличного воспроизведения звука.

Методы

Метод	Описание
Play() (52)	Запускает однократное воспроизведение звука из файла, доступного по пути path
StartPlay() (52)	Запускает цикличное воспроизведение звука из файла, доступного по пути path

Метод	Описание
StopPlay() (52)	Останавливает цикличное воспроизведение звука из файла, вызванное с помощью функции StartPlay()

Play()

Синтаксис:

```
err = Sound:Play(path)
```

Результат:

Функция запускает однократное воспроизведение звука из файла, доступного по пути `path`. В случае успеха возвращает **nil**, иначе сообщение об ошибке.

Пример использования:

```
function Play()
local err = Sound:Play("C:/Windows/Media/tada.wav")
    if err ~= nil then print(err) end
end
```

При запуске функции `Play` приостанавливается работа функции `StartPlay()` (если она выполняется) мнемосхемы и всего приложения и воспроизводится файл C:/Windows/Media/tada.wav.

StartPlay()

Синтаксис:

```
err = Sound:StartPlay(path)
```

Результат:

Функция запускает цикличное воспроизведение звука из файла, доступного по пути `path`. В случае успеха возвращает **nil**, иначе сообщение об ошибке. Для воспроизведения доступны только файлы в формате WAV.

Пример использования:

```
function StartPlay()
local err = Sound:StartPlay("C:/Windows/Media/tada.wav")
    if err ~= nil then print(err) end
end
```

При вызове метода `StartPlay()` запускается цикличное воспроизведение звука из файла C:/Windows/Media/tada.wav.

StopPlay()

Синтаксис:

```
err = Sound:StopPlay()
```

Результат:

Функция останавливает цикличное воспроизведение звука из файла, вызванное с помощью функции `StartPlay()`. В случае успеха возвращает **nil**, иначе сообщение об ошибке.

Пример использования:

```
function StopPlay()
local err = Sound:StopPlay()
    if err ~= nil then print(err) end
end
```

При вызове метода `StopPlay()` осуществляется остановка цикличного воспроизведения звука из файла.