InTouch HMI Руководство по разработке скриптов и логике

Содержание

Глава 1

Введение в разработку скриптов

Для построения более надежных приложений, Вы можете использовать скриптовый язык InTouch – QuickScript. Существует семь типов скриптов и множество встроенных скриптовых функций. Семь типов скриптов различаются причиной их выполнения. Например, скрипт приложения выполняется, когда приложение запускается, закрывается или работает. Скрипты по изменению данных выполняются, когда изменяется значение конкретного элемента. Скрипты окон выполняются, когда окно открывается, закрывается или открыто.
Встроенные скриптовые функции включают математические функции, тригонометрические функции, функции работы со строками и другие. Использование таких функций, позволяет сэкономить время при разработке проекта.
Скрипты InTouch
Могут включать Object Linking и Embedding
(OLE) объекты, а так же ActiveX объекты.
Для создания более сложных конструкций в приложении, можно также использовать локальные переменные, условные операторы, циклы при создании скриптов.

Базовые понятия

Перед тем как начать писать скрипт, необходимо понимать:

- **Скрипт** – это набор инструкций, которые указывают приложению выполнить какое-либо действие.
- **QuickScript –** скриптовый язык InTouch HMI
- **Функция** - это скрипт, который может быть вызван другим скриптом. В состав программного пакета InTouch HMI входит набор стандартных функций, которые можно использовать.
- **QuickFunctions** – это многократно используемые функции, написанные на скриптовом языке и хранящиеся библиотеке QuickFunctions. Для создания QuickFunction, необходимо просто создать QuickScript и назвать его. QuickFunction может быть вызвана другим скриптом или анимационной связью.

Типы скриптов

Скрипты в InTouch разделяются причиной, которая вызывает выполнение скрипта. Например, если необходимо, чтобы скрипт выполнялся, после того как оператор нажмет определенную кнопку на клавиатуре, то надо создать “key script” (скрипт по нажатию кнопки).

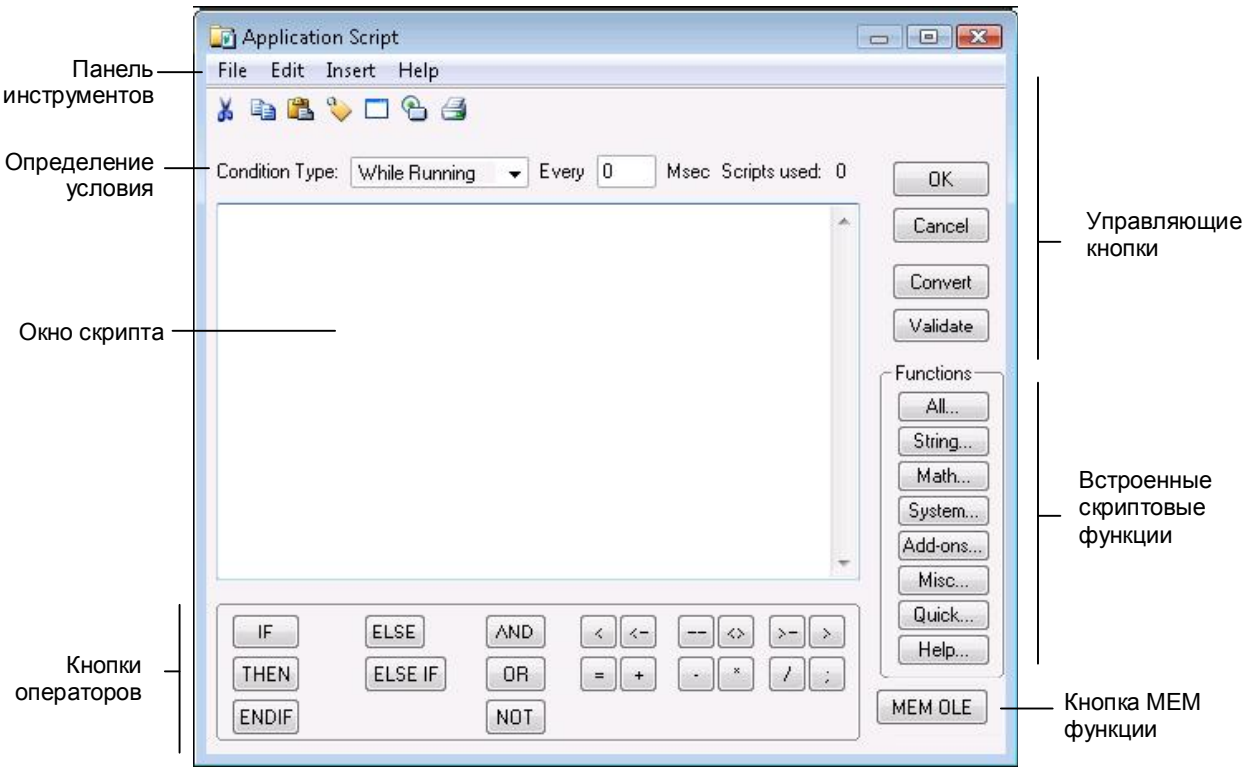
После того как выбран тип скрипта, необходимо далее определить критерий или условия, которые заставляют скрипт выполняться. Например, необходимо чтобы скрипт выполнился, когда оператор кнопку отпустит, а не когда нажмет.

Типы скриптов:

- **Application scripts (Скрипты приложения) –** выполняются непрерывно пока работает WindowViewer или только один раз при запуске или закрытии WindowViewer.
- **Window scripts (Скрипты окон) –** выполняется периодически, когда открыто окно InTouch проекта, или один раз при открытии или закрытии окна.
- **Key scripts (Скрипты кнопок)** – выполняется один раз или периодически когда определенная кнопка или сочетание кнопок нажата или отпущена.
- **Condition scripts (Скрипты по условию) –** выполняются один раз или периодически когда выполняется или не выполняется определенное условие.
- **Data change scripts (Скрипты по изменению данных) –** выполняется один раз когда изменяется значение определенного тега или выражения.
- **Action scripts (Скрипты действий) -** выполняются один раз или периодически когда оператор нажимает на графический объект в окне InTouch HMI
- **ActiveX event scripts (Скрипты событий ActiveX) –** выполняется один раз когда возникает событие ActiveX объекта, такое как нажатие на ActiveX объект.

Редактирование и создание скриптов

Для создания и редактирования скриптов в InTouch WindowMaker, существует специальный редактор.



Данные пример для скрипта приложения. Каждый тип скрипта имеет свой собственную версию вида окна редактирования, с опциями и полями уникальными для данного типа скрипта.

В заголовке окна редактора идентифицируется, с каким типом скрипта Вы работаете. Для более подробной информации см. раздел Типы скриптов.

В редакторе имеются кнопки вставки условных, математических и эквивалентных операторов. Можно просто нажав, на кнопку вставить функцию, символ, или ключевое слово в скрипт.

В поле условие, представлены доступные условия выполнения для данного типа скрипта.

Кнопка MEM OLE в правом нижнем углу редактора возникает, только если вместе с InTouch установлен Manufacturing Engineering Module (MEM). Нажатие на данную кнопку позволяет использовать в скрипте функции для работы с MEM.

Расширенные понятия

Некоторые расширенные возможности скриптов позволяют достигать усложненных функций, над базовыми возможностями InTouch HMI.

OLE объекты и ActiveX объекты позволяют получить доступ к системным функциям или взаимодействовать с другими программами, такими как Manufacturing Engineering Module.

OLE объекты

В скрипте Вы можете вызывать OLE объекты. OLE объекты позволяют получить доступ к системным функциям или взаимодействовать с другими программами, такими как Manufacturing Engineering Module.

Например, при помощи OLE, Вы можете:

- Генерировать случайные числа.
- Создавать диалоговые окна интерфейса пользователя.
- Открывать панель свойств даты и времени Windows.
- Записывать и считывать регистры.
- Скрывать окна.

Создание скриптов с ActiveX объектами

Несколько ActiveX объектов поставляются вместе с InTouch HMI в меню Wizards (Мастера). Так как InTouch HMI основан на операционной среде Windows, то Вы можете использовать практически любой ActiveX объект с InTouch HMI.

Глава 2

Создание и редактирование скриптов

Этапы создания скрипта варьируются в зависимости от типа скрипта. В основном, Вы открываете редактор, выбираете тип условия, вводите операторы и сохраняете скрипт.

Для более подробной информации по созданию скриптов каждого типа, смотрите следующие разделы:

- Конфигурирование скриптов приложения
- Конфигурирование скриптов окон
- Конфигурирование скриптов кнопок
- Конфигурирование скриптов по условию
- Конфигурирование скриптов по изменению
- Конфигурирование скриптов действий
- Конфигурирование скриптов событий ActiveX

Смотрите разделы ниже по основным операциям редактирования, а так же по некоторым дополнительным возможностям, которым помогут сэкономит время.

- Открытие скрипта для редактирования.
- Сохранение или Отмена изменений в скрипте.
- Копирование, Вырезание и Вставка текста.
- Поиск и/или замена текста
- Вставка элементов кода
- Доступ к справке по скриптовым функциям

Открытие скрипта для редактирования

Открытие уже существующего скрипта немного отличается, в зависимости от типа скрипта.

Открытие скрипта приложения

1. Сделать это можно несколькими методами:
   - В панели навигации, в меню Script (Скрипт), двойным нажатием на Application (Приложение).
   - В меню Special (Специальные), выбрать Script (Скрипт), и Application Scripts (Скрипты приложения).
2. В поле выбора типа условия скрипта, выбрать тип скрипта, которые необходимо редактировать.

Открытие скрипта окна:

1. Сделать это можно несколькими методами:
   - В панели навигации, в меню Script (Скрипт), двойным нажатием на Window Scripts (Скрипты окон).
   - В меню Special (Специальные), выбрать Script (Скрипт), и Window Scripts (Скрипты окон).
   - Открыть окно, с которым связан скрипт. Нажать правой кнопкой мыши на пустом поле окна и контекстном меню выбрать Window Scripts (Скрипты окон).
2. В поле условия выбрать условие, которое будет вызывать скрипт.

Открытие скрипта событий ActiveX объекта:

1. Сделать это можно несколькими методами:
   - В панели навигации, в меню Script (Скрипт), двойным нажатием на ActiveX Event (Скрипты окон).
   - Двойным нажатием на ActiveX компонент, с которым связан скрипт. Выбрать закладку Events (События) и затем двойным нажатием открыть скрипт.

Открытие скрипта действия:

1. Открыть окно, которое содержит графический элемент, к которому привязан скрипт.
2. Двойным нажатием на графическом элементе открыть окно анимационных связей.
3. В разделе Touch Pushbuttons, выбрать Action. Откроется окно редактора.
4. В поле условия выбрать условие, которое будет вызывать скрипт.

Открытие других типов скриптов:

1. Сделать это можно несколькими методами:
    1. В панели навигации, раскрыть меню Script (Скрипт), выбрать соответствующий тип скрипта, двойным нажатием на имени скрипта открыть его.
    2. В меню Special (Специальные), выбрать Script (Скрипт), и соответствующий тип скрипта. В открывшемся редакторе, нажать кнопку Browse, и выбрать имя скрипта.
2. При необходимости, выбрать тип условия, которое будет вызывать скрипт.

Сохранение или Отмена изменений в скрипте

При работе в редакторе скриптов или по окончанию работы, можно сохранить скрипт автоматически или вручную. Или можно отменить все изменения.

Опция восстановления не доступна для скриптов окна и приложения.

> **Примечание** Сохранение или Отмена изменений всегда применяется для всех типов условий скрипта, а не только для условия, которое в данный момент видимо.

Сохранить изменения и оставить редактор открытым

- В меню Script нажать Save (Сохранить).

Сохранить изменения и закрыть редактор

- Нажать ОК.

Отменить изменения и оставить редактор открытым

- Нажать Restore (Восстановить)

Отменить изменения и закрыть редактор

- Нажать Cancel (Отмена).

Копирование, Вырезание и Вставка текста

Копирование, вырезание и вставка текста в редактора работает абсолютно также как и в других приложениях Windows. Можно использовать стандартные горячие кнопки Ctrl+C, Ctrl+X, Ctrl+V или кнопки с панели инструментов.

Поиск и/или замена текста

Вы можете производить поиск и замену текста в скрипте.

Для поиска и/или замены текста

- В меню Edit (Редактирование), нажать Find (Поиск). Откроется окно замены текста.

Опции данного окна работают абсолютно таким же образом, как и в других приложениях Windows, таких как Notepad (Блокнот).

Вставка элементов кода

Вы можете автоматически вставлять в скрипт различные элементы кода, просто выбрав их из списка. Это поможет сэкономить время и уменьшит количество возможных ошибок.

Для того чтобы вставить функцию в скрипт

1. В меню Insert (Вставка), выбрать Functions (Функции) и выбрать категорию функции. Появится соответствующее окно.

   Если функции, которую необходимо вставить нет в списке, нажмите на кнопку Next Page (След. страница) в нижнем левом углу для перехода на следующую страницу списка функций.

2. Выберите необходимую функцию. Диалоговое окно закроется, и функция будет вставлена в скрипт, на местоположение курсора.

Вставить тег в скрипт

1. В меню Insert (Вставка), выбрать Tagname (Теги). Появится окно Select Tag (Выбор тега).
2. И выбрать тег.

   В качестве альтернативы, можно вставить тег в скрипт, просто двойным нажатием на теге и выбранном из списка поле.

3. Для включения поля тега, нажмите на список полей тега.
4. Нажмите OK. Окно Select Tag (Выбор тега) закроется и тег (с полем, если есть) вставится в скрипт на местоположение курсора.

Для более подробной информации по работе с окном Select Tag (Выбор тега), включая установку нескольких источников тегов, смотрите раздел Выбор тега InTouch Глава 4, Анимационные Объекты, в *Руководстве по визуализации InTouch HMI*.

Вставка поля тега в скрипт

1. Написать имя тега и поставить точку.
2. Двойным нажатием справа от точки открывается диалоговое окно Choose Field (Выбор поля).
3. Выбрать поле, которое необходимо использовать. Диалоговое окно закроется, и поле вставится в скрипт на местоположение курсора.

Вставка имени окна в скрипт

1. В меню Insert (Вставка), выбрать Window (Окно). Появится окно Window Name (Имя окна).
2. Выбрать необходимое окно. Диалоговое окно закроется, и имя окна вставится в скрипт.

Вставка свойства или метода ActiveX в скрипт

1. В меню Insert (Вставка), выбрать ActiveX. Появится диалоговое окно ActiveX Control Browser.
2. В списке объектов, выбрать необходимый ActiveX объект.
3. В списке свойств и методов справа, выбрать необходимое свойство или метод.
4. Нажать ОК. Диалоговое окно закроется, и свойство или метод вставиться в скрипт.

Вставка ключевого слова или оператора в скрипт

- Нажать соответствующую кнопку в нижней части редактора скриптов. Ключевое слово или оператор будут вставлены в скрипт.

Доступ к справке по скриптовым функциям

Если необходимо просмотреть справочную информацию по скриптовым функциям, то к ним можно получить доступ непосредственно из редактора скриптов.

Для просмотра справки по определенной функции

1. В нижнем правом углу редактора скрипта, нажмите Help (Справка). Появится список функций.
2. Если необходимой функции нет в списке, то нажмите на кнопку Next Page (След. страница) в нижнем левом углу для перехода на следующую страницу списка функций.
3. Нажмите на необходимую функцию. Появится справка по соответствующей функции.

Проверка синтаксиса в скрипте

При сохранении скрипта, редактор скриптов автоматически проверяет синтаксис. Если имеются ошибки, то появится сообщение с более подробной информацией. Необходимо исправить все синтаксические ошибки перед сохранением скрипта. Проверку также можно запустить вручную, при редактировании скрипта.

Чтобы вручную проверить синтаксис в скрипте

- Нажмите Validate (Проверить).

Печать скриптов

Вы можете распечатать каждый скрипт отдельно в редакторе, или можно распечатать все скрипты определенного типа, при помощи опции в WindowMaker.

Печать отдельного скрипта

1. Открыть скрипт в редакторе. См. раздел Открытие скрипта для редактирования
2. Нажать кнопку Print (Печать) в панели инструментов. Скрипт распечататься на стандартном принтере Windows.

Печать всех скриптов определенного типа

1. В меню File (Файл) WindowMaker, нажать Print (Печать). Откроется диалоговое окно вывода на печать.
2. Выбрать типы скриптов, которые необходимо вывести на печать. Для вывода на печать всех скриптов, нажать All Scripts (Все скрипты).
3. Нажать Next (Далее). Появится диалоговое окно Select Output Destination.
4. Сделать одно из следующих:
    3. Нажать Send output to Printer (Распечатать на принтере).
    4. Нажать Send output to Text File (Распечатать в файл).
5. Нажать кнопку Browse (Обзор), выбрать принтер, или найти файл.
6. Нажать Print (Печать).

Удаление скриптов

Действия по удалению скриптов могут варьироваться в зависимости от типа скрипта Смотрите следующие разделы:

- Конфигурирование скриптов приложения
- Конфигурирование скриптов окон
- Конфигурирование скриптов кнопок
- Конфигурирование скриптов по условию
- Конфигурирование скриптов по изменению
- Конфигурирование скриптов действий
- Конфигурирование скриптов событий ActiveX

Глава 3

Типы скриптов

Все скрипты в InTouch HMI выполняются по срабатыванию какого-либо условия. Каждый тип скрипта имеет одно или несколько условий для запуска выполнения скрипта.

В редакторе скриптов, можно выбрать тип условия срабатывания, по которому будет выполняться скрипт. При выборе условия, выбирается когда, и как будет выполняться скрипт.

Вы можете сконфигурировать различные типы условий, основанные на действиях пользователя, внутренних состояниях, и изменениях значений тегов. Действия пользователей включают нажатия кнопок, нажатие на графические элементы. Условия по внутренним состояниям может включать в себя запуск WindowViewer.

Скрипты могут запускаться по следующим действиям:

- Запуск и закрытие WindowViewer. Смотрите раздел Конфигурирование скриптов приложения
- По открытию и закрытию окна. Смотрите раздел Конфигурирование скриптов окон
- По нажатию кнопок или комбинации кнопок. Смотрите раздел Конфигурирование скриптов кнопок
- По удовлетворению определенным условиям – тегу или значению выражения. Смотрите раздел Конфигурирование скриптов по условию
- По изменению значения тега или значения поля тега. Смотрите раздел Конфигурирование скриптов по изменению
- По нажатию на графический объект. Смотрите раздел Конфигурирование скриптов действий
- По нажатию на ActiveX объект. Смотрите раздел Конфигурирование скриптов событий ActiveX.

Так же можно приостановить выполнение скрипта. По умолчанию, при запуске WindowViewer, запускается логика и выполняются скрипты. Можно приостановить выполнения скрипта в режиме исполнения, путем остановки логики. После остановки, можно возобновить выполнение скрипта. Для более подробной информации, смотрите раздел Приостановка выполнения скрипта в режиме исполнения, страница 40.

Типы условий срабатывания скрипта

В InTouch HMI, скрипты разделяются на семь типов. Каждый тип скрипта имеет одно или более условий срабатывания, которые можно для запуска скрипта.

- **Application script** (Скрипт приложения) – имеет три типа условий срабатывания: on startup (на запуске), on shut down (на выключении), while running (пока работает). Каждое условие может выполнять различный скрипт.
- **Window script (Скрипт окна)** имеет три типа условий срабатывания: on open (на открытие), on close (на закрытие), и while open (пока открыто).
- **Key script (Скрипт кнопки)** имеет три типа условий срабатывания: on key up (по отпусканию кнопки), on key down (по нажатию кнопки), или while key down (пока кнопка нажата).
- **Condition script (Скрипт по условию)** имеет четыре типа условий срабатывания: on true (как только условие удовлетворяется), while true (пока условие удовлетворяется), on false (как только условие не удовлетворяется), and while false (пока условие не удовлетворяется).
- **Data change script (Скрипт по изменению) –** выполняется как только изменяется значение определенного тега или выражения.
- **Action script (Скрипт по действию) –** выполняется один раз или периодически, когда оператор нажимает на графический объект InTouch HMI.
- **ActiveX event script (Скрипт ActiveX объектов) –** выполняется один раз, когда возникает событие ActiveX объекта, например нажатие на ActiveX объект.

Использование нескольких условий срабатывания скрипта

Для большинства типов скриптов можно использовать несколько условий срабатывания и привязать разные скрипты для каждого условия.

Например, можно сконфигурировать один скрипт на выполнение при запуске WindowViewer, а другой скрипт на выполнение периодически, когда WindowViewer запущен и работает.

Для того чтобы просмотреть существующий скрипт для условия срабатывания, необходимо выбрать условие в списке Condition Type.

Периодическое выполнение скриптов

Периодически выполняемые скрипты, не выполняются сразу после того, как срабатывает условие, их выполнение начинается после определенного в условии промежутка времени.

Например, если сконфигурировать скрипт по нажатию кнопки, который должен выполняться каждые 5000 мс, пока нажата определенная кнопка, то скрипт выполниться через 5 секунд **после** нажатия кнопки, а затем будет выполняться каждые 5 секунд, пока нажата кнопка.

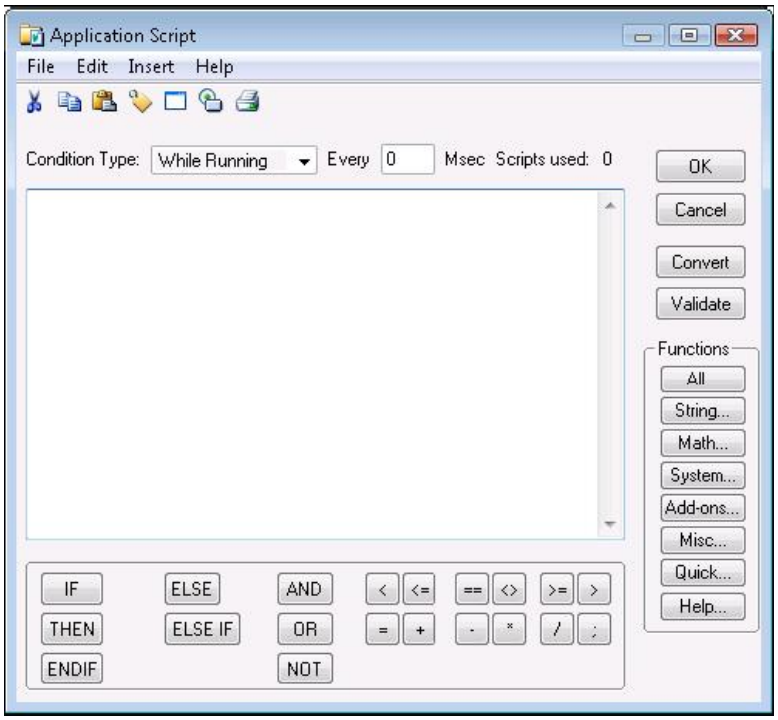
Конфигурирование Application скриптов

Скрипты приложения связаны со всем приложением InTouch HMI. Можно использовать скрипты приложения, чтобы:

- Выполнить скрипт однократно, при запуске WindowViewer.
- Выполнять скрипт периодически пока работает WindowViewer.
- Выполнить скрипт однократно, при выключении WindowViewer.

Чтобы сконфигурировать скрипт приложения:

1. В панели навигации, в разделе Scripts, нажать на Application и в контекстном меню выбрать Open (Открыть). Появится Окно Application Script (Скрипт приложения).



2. В разделе Condition Type, выберите условие выполнения скрипта:
   - On startup – конфигурирование скрипта на однократное выполнение, при запуске WindowViewer.
   - While running – конфигурирование скрипта на выполнение периодически, пока работает WindowViewer
   - On shut down – конфигурирование скрипта на однократное выполнение, при выключении WindowViewer.
3. Если на предыдущем этапе выбран тип условия While running, необходимо в поле **Every** ввести значение времени в интервале между 1 и 360000 миллисекунд. Это значение будет определять, как часто будет выполняться скрипт.
4. Напечатать в редакторе скрипт.
5. Нажать ОК.

Удаление скрипта приложения

1. В панели навигации, в разделе Scripts, нажать на Application и в контекстном меню выбрать Open (Открыть). Появится Окно Application Script (Скрипт приложения).
2. В разделе Condition Type, выберите условие, для которого необходимо удалить скрипт. В окне редактора появится текст скрипта.
3. В меню Edit, выбрать Clear. Текст скрипта удалится и соответственно удалится и сам скрипт.

Ограничения скриптов приложения

Скрипты приложения, которые выполняются при запуске и при выключении WindowViewer, имеют некоторые ограничения по взаимодействию с другими объектами.

Нельзя использовать скрипты приложения On startup:

- Для обращения к свойствам, методам и событиям ActiveX объекта.
- Считывать или записывать в элементы управления, теги ввода-вывода или удаленные теги.
- Запускать скрипты Data change (По изменению) и Condition (По условию).

Нельзя использовать скрипты приложения On shut down:

- Считывать или записывать в элементы управления, теги ввода-вывода или удаленные теги.
- Запускать другие приложения.

Конфигурирование Window скриптов

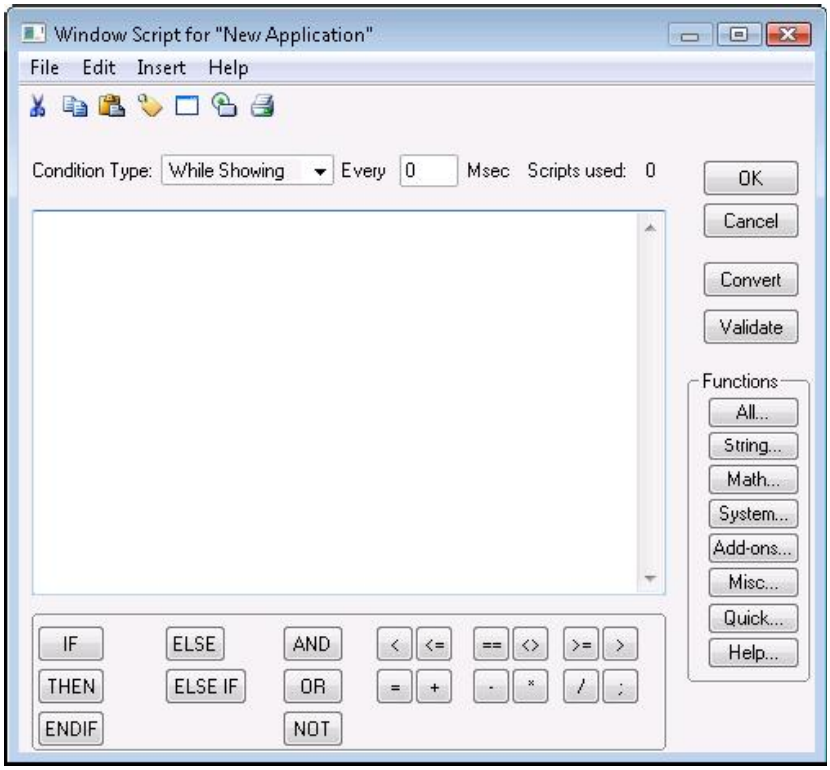
Скрипты окон – это скрипты, которые определены для конкретных окон. Скрипты окон можно использовать чтобы:

- Выполнить скрипт однократно, при открытии окна InTouch HMI.
- Выполнять скрипт периодически, пока открыто окно InTouch HMI.
- Выполнить скрипт однократно, при закрытии окна InTouch HMI.

Примечание Показать окно InTouch (Show) рассматривается также как и открытие окна. Скрыть окно InTouch (Hide) рассматривается также как и закрытие окна.

Чтобы сконфигурировать скрипт окна:

1. В панели навигации, в разделе Windows, выбрать окно и в контекстном меню выбрать Windows scripts (Скрипты окна). Появится окно редактора.



2. В разделе Condition Type, выберите условие выполнения скрипта:
   - On Show – конфигурирование скрипта на однократное выполнение, при открытии соответствующего окна.
   - While Showing – конфигурирование скрипта на выполнение периодически, пока открыто соответствующее окно.

- On Hide – конфигурирование скрипта на однократное выполнение, при закрытии соответствующего окна.

3. Если на предыдущем этапе выбран тип условия While Showing, то необходимо в поле **Every** ввести значение времени в интервале между 1 и 360000 миллисекунд. Это значение будет определять, как часто будет выполняться скрипт.
4. Напечатать в редакторе скрипт.
5. Нажать ОК.

Удаление скрипта окон:

1. В панели навигации, в разделе Windows, выбрать окно и в контекстном меню выбрать Windows scripts (Скрипты окна). Появится окно редактора.
2. В разделе Condition Type, выберите условие, для которого необходимо удалить скрипт. В окне редактора появится текст скрипта.
3. В меню Edit, выбрать Clear.

Конфигурирование Key скриптов

Скрипты кнопок - это скрипты которые, привязаны к нажатию определенных кнопок или комбинации кнопок. Скрипты кнопок можно использовать чтобы:

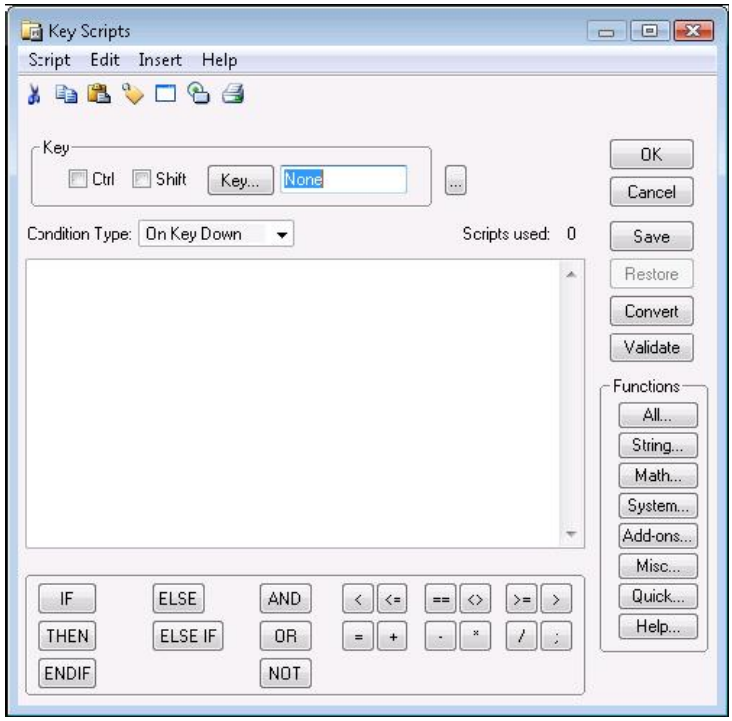
- Выполнить скрипт однократно, при нажатии определенной кнопки или комбинации кнопок.
- Выполнять скрипт периодически, пока нажата определенная кнопка или комбинация кнопок.
- Выполнить скрипт однократно, когда определенная кнопка или комбинация кнопок отпущена.

Скрипты кнопок идентифицируются по имени кнопки запускающей выполнение скрипта. Например, Ctrl+q.

> **Примечание** Если сконфигурирован Action (Действия) скрипт, который использует для вызова выполнения те же кнопки или комбинацию кнопок, то Key скрипт игнорируется и вместо него выполняется Action скрипт.

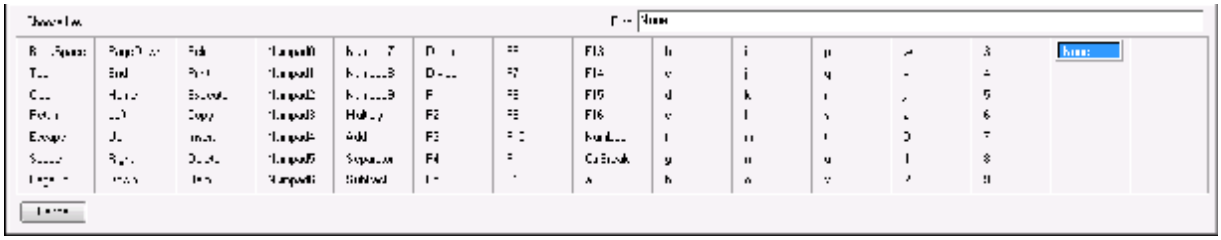
Чтобы сконфигурировать скрипт кнопки:

1. В панели навигации, в разделе Scripts, сделать одно из следующих:
   - Для конфигурирования нового скрипта кнопки, нажать на Key и в контекстном меню выбрать New (Новый). Появится редактора.



   - Для конфигурирования уже существующего скрипта, необходимо раскрыть вкладку Key, выбрать имя скрипта и нажать Edit. Откроется окно редактора.

2. Нажмите кнопку Key и выберите кнопку в открывшемся диалогом окне Choose Key.



3. Для назначения комбинации кнопок выберите Ctrl и/или Shift и необходимую кнопку.

4. В разделе Condition Type, выберите условие выполнения скрипта:
   - On Key Down – конфигурирование скрипта на однократное выполнение, при нажатии соответствующей кнопки или комбинации кнопок.
   - While Down – конфигурирование скрипта на выполнение периодически, пока нажата соответствующая кнопка или комбинация кнопок.
   - On Key Up – конфигурирование скрипта на однократное выполнение, при отпускании соответствующей кнопки или комбинации кнопок.
5. Если на предыдущем этапе выбран тип условия While Down, то необходимо в поле **Every** ввести значение времени в интервале между 1 и 360000 миллисекунд.
6. Напечатать в редакторе скрипт.
7. Нажать ОК.

Удаление всех Key скриптов, привязанных к определенной кнопке

- В панели навигации, в разделе Scripts, раскрыть вкладку Key, выбрать имя скрипта и нажать Delete. Когда появится сообщение, нажать Yes (Да).

Удаление Key скрипта, привязанного к определенной кнопке

1. В панели навигации, в разделе Scripts, раскрыть вкладку Key, выбрать имя скрипта и нажать Edit. Отроется окно редактора.
2. В разделе Condition Type, выберите условие, для которого необходимо удалить скрипт. В окне редактора появится текст скрипта.
3. В меню Edit, выбрать Clear. Текст скрипта удалится и соответственно удалится и сам скрипт.

Конфигурирование Condition скриптов

Скрипты по условию – это скрипты, выполняются, когда удовлетворяется определенное логическое условие. Скрипты по условию можно использовать чтобы:

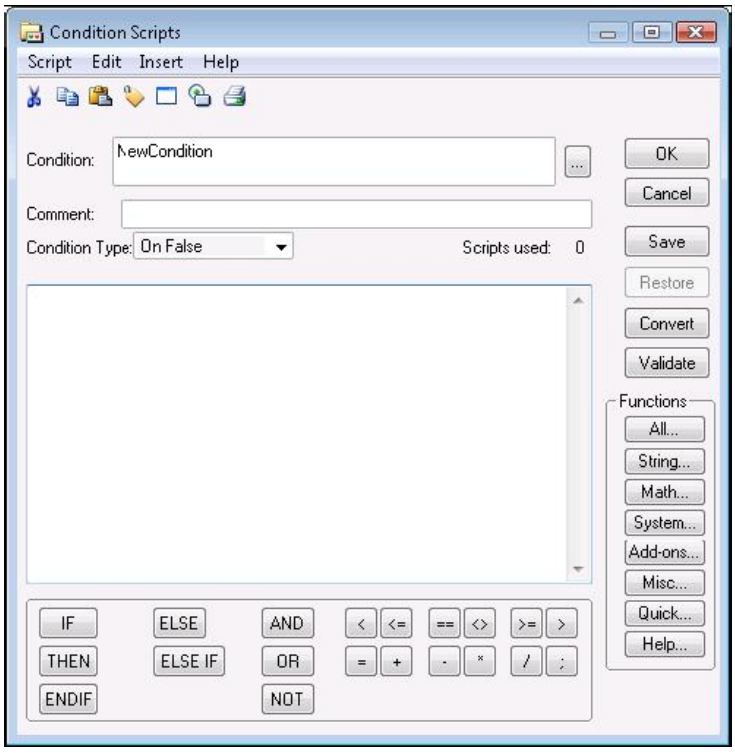
- Один раз, когда условие удовлетворяется.
- Один раз, когда условие не удовлетворяется.
- Периодически, пока определенное условие удовлетворяется.
- Периодически, пока определенное условие не удовлетворяется.

Скрипт по условию идентифицируется условием, которое запускает выполнение скрипта. Например, Tag1>=13.

> **Примечание** Если скрипт сконфигурирован на On True (когда условие удовлетворяется), то он выполняется, когда условие переходит из состояния False в состояние True. Если скрипт сконфигурирован на On False (когда условие не удовлетворяется), то он выполняется, когда условие переходит из состояния True в состояние False.

Чтобы сконфигурировать скрипт кнопки:

1. В панели навигации, в разделе Scripts, сделать одно из следующих:
   - Для конфигурирования нового скрипта кнопки, нажать на Condition и в контекстном меню выбрать New (Новый). Появится редактора.



- Для конфигурирования уже существующего скрипта, необходимо раскрыть вкладку Condition, выбрать имя скрипта и нажать Edit. Откроется окно редактора.

2. В поле Condition, необходимо ввести имя условия или отредактировать существующее.
3. Ввести комментарий в поле Comment.
4. Из списка Condition Type, выбрать:
   - On False - скрипт выполняется однократно, как только условие не удовлетворяется.
   - While false – скрипт выполняется периодически пока условие не удовлетворяется.
   - On true - скрипт выполняется однократно, как только условие удовлетворяется.
   - While true – скрипт выполняется периодически пока условие удовлетворяется.
5. Если на предыдущем этапе выбран тип условия While true или While false, то необходимо в поле **Every** ввести значение времени в интервале между 1 и 360000 миллисекунд.
6. Напечатать в редакторе скрипт или отредактировать существующий.
7. Нажать OK.

Удаление всех Condition скриптов, привязанных к определенному условию:

- В панели навигации, в разделе Scripts, раскрыть вкладку Condition, выбрать имя скрипта и нажать Delete. Когда появится сообщение, нажать Yes (Да).

Удаление Condition скрипта, привязанного к определенному условию:

1. В панели навигации, в разделе Scripts, раскрыть вкладку Condition, выбрать имя скрипта и нажать Edit. Отроется окно редактора.
4. В разделе Condition Type, выберите условие, для которого необходимо удалить скрипт. В окне редактора появится текст скрипта.
5. В меню Edit, выбрать Clear. Текст скрипта удалится и соответственно удалится и сам скрипт.

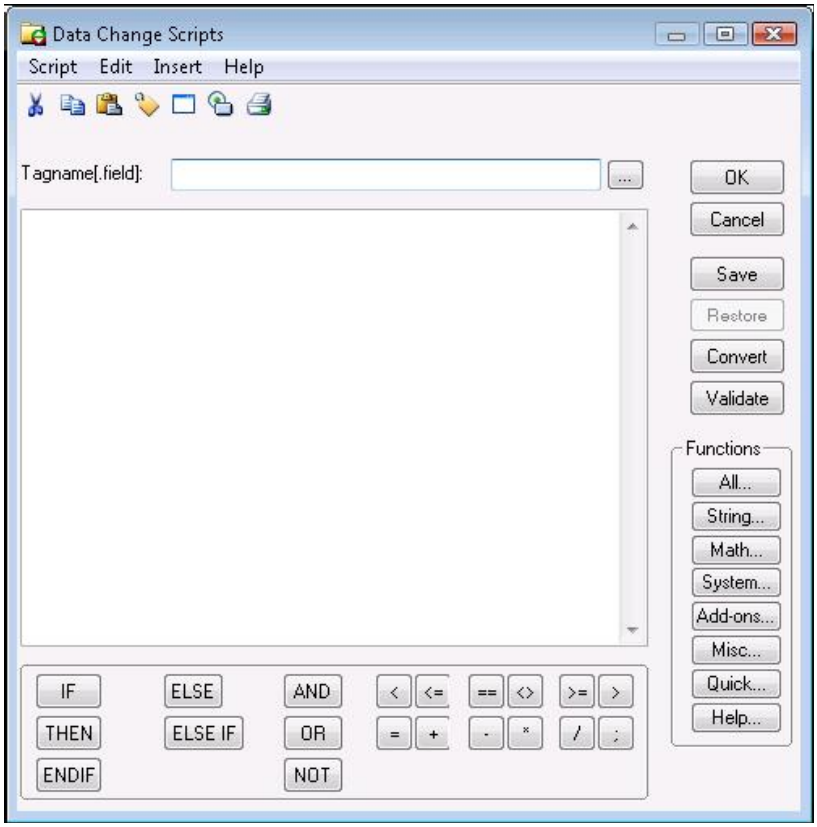
Конфигурирование Data Change скриптов

Можно использовать скрипты по изменению, для выполнения скрипта один раз при изменении значения определенного тега или поля тега более чем на определенное значение нечувствительности.

Data Change скрипты идентифицируются по имени тега или полю тега, которое инициирует выполнение скрипта. Например: Tag1.

Чтобы сконфигурировать скрипт приложения:

1. В панели навигации, в разделе Scripts, нажать на Data Change и в контекстном меню выбрать New (Новый). Появится окно редактора.



2. В поле Tagname[.field], ввести имя тега или поле тега.
3. Напечатать в редакторе скрипт.
4. Нажать OK.

Удаление Data Change скрипт:

- В панели навигации, в разделе Scripts, раскрыть вкладку Condition, выбрать имя скрипта и нажать Delete. Когда появится сообщение, нажать Yes (Да).

Конфигурирование Action скриптов

Можно скрипты, ассоциировать с действиями оператора над графическими объектами. Можно сконфигурировать одно или несколько событий с графическими объектами:

- Нажатие правой, левой или центральной кнопки мыши.
- Нажатие и удерживание правой, левой или центральной кнопки мыши.
- Отпускание правой, левой или центральной кнопки мыши.
- Двойное нажатие правой, левой или центральной кнопки мыши.
- Нажатие комбинации кнопок.
- Нажатие и удержание комбинации кнопок.
- Помещение указателя мыши над графическим объектом.

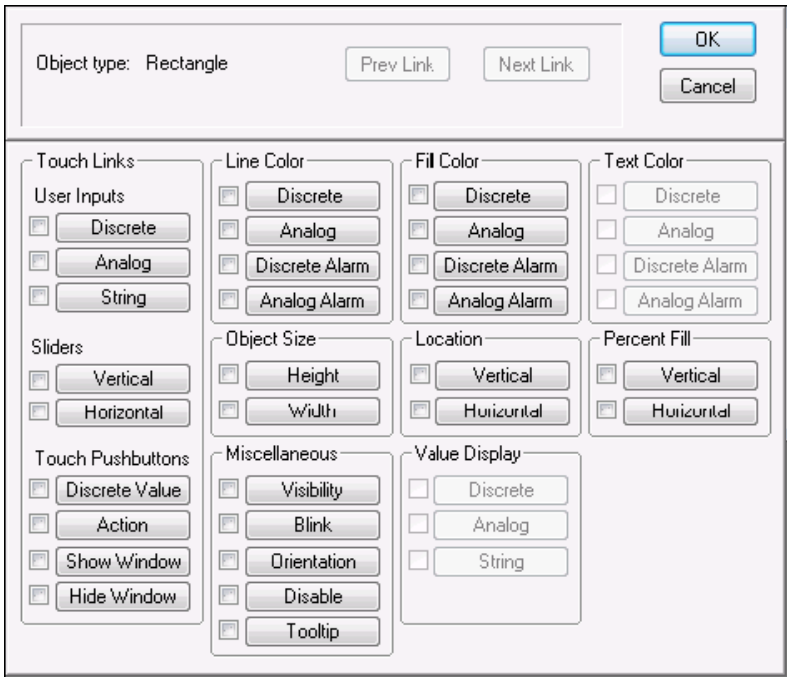
Скрипт действия может быть сконфигурирован только из панели анимационных связей самого объекта.

Примечание Если сконфигурирован Action (Действия) скрипт, который использует для вызова выполнения те же кнопки или комбинации кнопок, то Key скрипт игнорируется и вместо него выполняется Action скрипт.

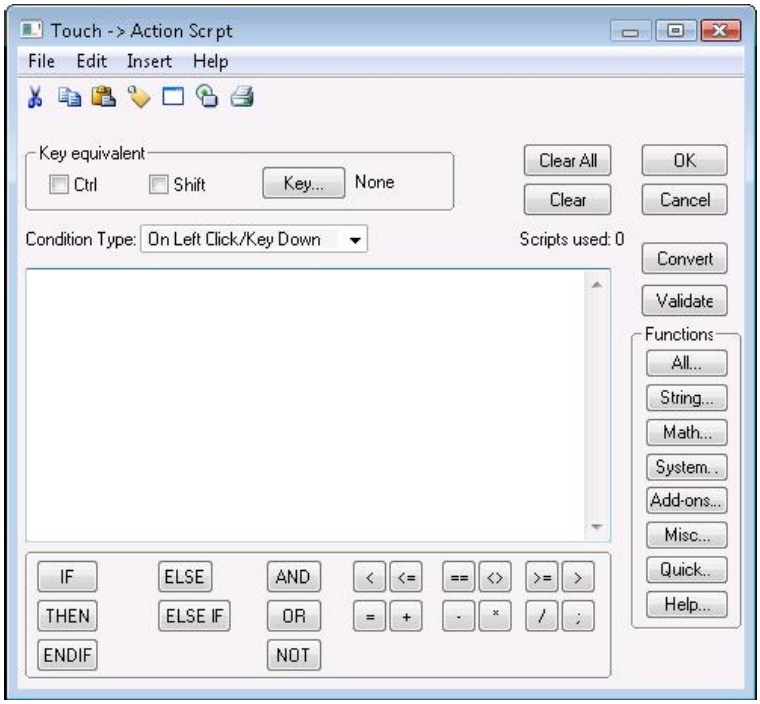
Допустимо выполнение скрипта действия, который вызывает Quick, скрипт аргументом которого является другой Quick скрипт. Когда скрипт выполняется, соответствующее сообщение записывается в логгер.

Чтобы сконфигурировать скрипт действия:

1. Двойное нажатие на графический объект. Появляется окно анимационных связей.



2. Нажать кнопку Action (Действие) -> появится окно редактора.



3. В списке Condition Type необходимо выбрать одно из условий:

Конфигурирование скрипта для выполнения по данному событию	Событие
Один раз, когда нажата левая кнопка мыши или определенная кнопка/комбинация кнопок.	On Left Click/Key Down
Периодически, пока нажата левая кнопка мыши или определенная кнопка/комбинация кнопок.	While Left/Key Down
Один раз, когда отжата левая кнопка мыши или определенная кнопка/комбинация кнопок.	On Left/Key Up
Один раз, когда дважды нажата левая кнопка мыши	On Left Double Click
Один раз, когда нажата правая кнопка мыши	On Right Click
Периодически, пока нажата правая кнопка мыши	While Right Down
Один раз, когда отжата правая кнопка мыши	On Right Up
Один раз, когда дважды нажата правая кнопка мыши	On Right Double Click
Один раз, когда нажата центральная кнопка мыши	On Center Click
Периодически, пока нажата центральная кнопка мыши	While Center Down
Один раз, когда отжата центральная кнопка мыши	On Center Up
Один раз, когда дважды нажата центральная кнопка мыши	On Center Double Click
Один раз, когда указатель мыши находится над объектом.	On Mouse Over

4. Если выбрано On Left Click/Key Down, While Left/Key Down или On Left/Key Up:

   a. Нажать Key. Появится диалоговое окно выбора кнопки.

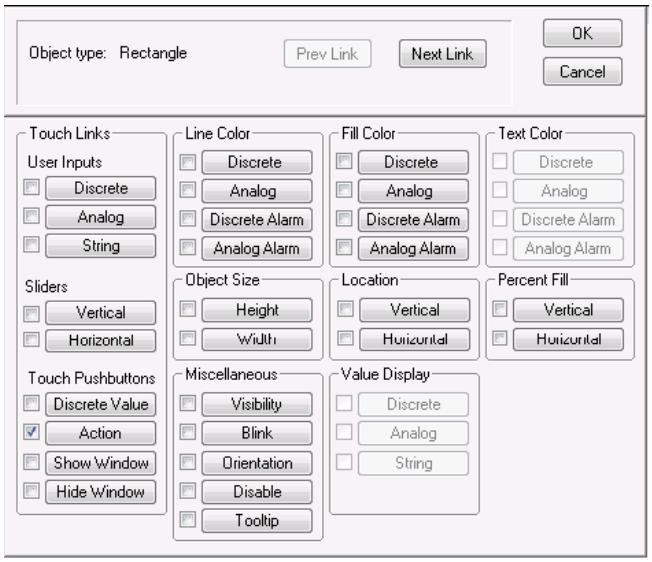
   b. Выбрать кнопку

   c. Выбрать Ctrl и/или Shift для назначения к выбранной кнопке комбинации кнопок.

5. Если выбрано While Left/Key Down or While Right Down, необходимо ввести временной интервал в поле Every, в диапазоне от 1 до 360000 миллисекунд.
6. Если выбрано On Mouse Over в поле After, ввести значение в диапазоне от 1 до 360000 миллисекунд. Это время, спустя которое после наведения курсора мыши на объект, будет выполнен скрипт.
7. Ввести текст скрипта.
8. Нажать OK.

Для удаления всех скриптов, ассоциированных с графическим объектом InTouch:

1. Двойным нажатием мыши на объекте, открыть окно его анимационных свойств.



2. Нажмите на поле рядом с кнопкой Action, для снятия галочки. Скрипт действия не будет выполняться во время исполнения проекта. Если галочку поставить, то в редакторе появится последний достоверный скрипт, который будет выполняться в режиме исполнения.

Для удаления отдельного скрипта, ассоциированного с графическим объектом InTouch:

1. Двойным нажатием мыши на объекте, открыть окно его анимационных свойств.
2. Нажать кнопку Action (Действие) -> появится окно редактора.
3. В списке Condition Type необходимо выбрать одно из условий.
4. В меню Edit, выбрать Clear. Скрипт для данного события удалится.

Конфигурирование скриптов событий ActiveX компонента

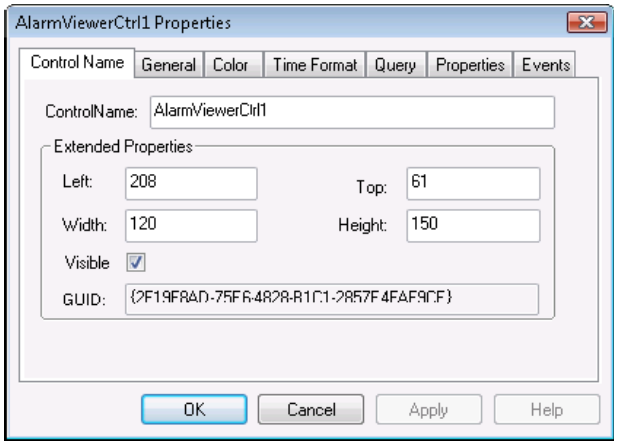
Скрипты ActiveX компонента выполняются, когда происходит событие ActiveX компонента. В зависимости от ActiveX компонента, такими событиями могут быть:

- Запуск ActiveX компонент: Startup
- Закрытие ActiveX компонент: Shutdown
- Нажатие на ActiveX компонент: Click
- Двойное нажатие на ActiveX компонент: Doubleclick

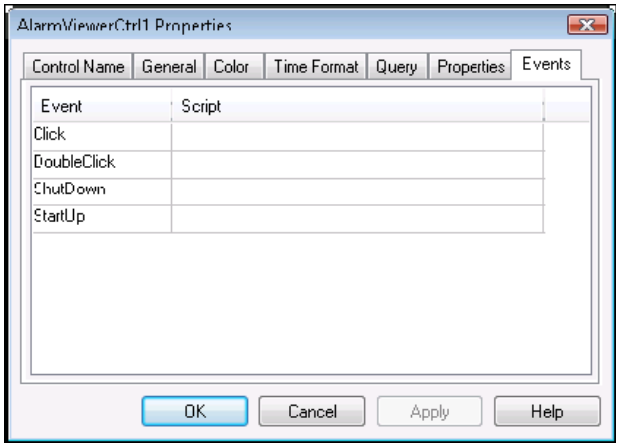
Скрипт события ActiveX компонента идентифицируется по имени, которое является комбинацией имени ActiveX компонента и события. Например, MyActiveXScript (AlarmViewerCtrl1::Click).

Для конфигурирования нового скрипта ActiveX компонента

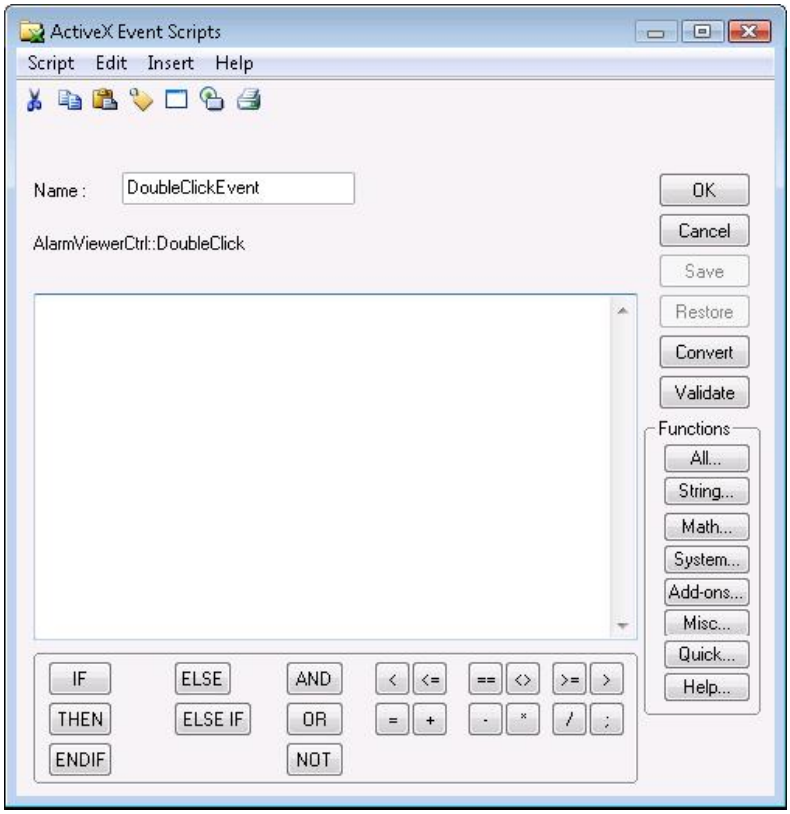
1. Двойным нажатием на ActiveX компонент, открыть окно его свойств.



2. Раскрыть вкладку Events.



3. Выбрать событие: click, double-click, shut down или start up.
4. Нажать в поле Scripts для данного события. Появятся квадратные скобки.
5. Ввести имя нового скрипта события и нажать OK. Появится окно редактора.



6. В поле Name, можно внести изменения в имя скрипта события ActiveX компонента.
7. Ввести текст скрипта.
8. Нажать OK.

Для редактирования существующего скрипта ActiveX компонента.

1. В панели навигации, в разделе Scripts, выбрать раздел ActiveX Event. Выбрать имя скрипта и в контекстном меню, нажать Edit. Появится окно редактора.
2. Внесите необходимые изменения и нажмите OK.

Для удаления существующего скрипта ActiveX компонента.

1. Убедитесь в том, что ни один ActiveX компонент не использует скрипт, который необходимо удалить. Если такой ActiveX компонент существует, то сделать следующее:
   a. Удалить ссылку на скрипт события ActiveX компонента во вкладке Events, каждого ActiveX компонента, который может его использовать.
   b. Закрыть все окна и выполнить команду Update Use Counts.
2. В панели навигации, в разделе Scripts, выбрать раздел ActiveX Event. Выбрать имя скрипта и в контекстном меню, нажать Delete. Скрипт удалится.

Приостановка выполнения скрипта в режиме исполнения

По умолчанию, при запуске WindowViewer, запускается выполнение логики, и выполняются синхронные скрипты. Можно приостановить выполнение скрипта в режиме исполнения. После остановки можно возобновить выполнение скрипта.

Невозможно приостановить или возобновить выполнение скриптов ArchestrA символов, из меню Logic в панели WindowViewer или просто записать значение в системный тег $LogicRunning.

Для приостановки выполнения скрипта в режиме исполнения из меню

- В меню Logic, выбрать Halt Logic. Выполнение синхронных скриптов остановиться. Асинхронные скрипты продолжат выполняться, запуск новых асинхронных скриптов осуществляться не будет.

Для приостановки выполнения скрипта в режиме исполнения скриптом

- Записать значение 0 в дискретный системный тег $LogicRunning. Выполнение синхронных скриптов остановиться. Асинхронные скрипты продолжат выполняться, запуск новых асинхронных скриптов осуществляться не будет.

Для возобновления выполнения скриптов в режиме исполнения

- В меню Logic, выбрать Start Logic. Возобновится выполнение скрипта.

Для возобновления выполнения скриптов в режиме исполнения скриптом

- Записать значение 1 в дискретный системный тег $LogicRunning. Системный тег $LogicRunning должен содержатся в асинхронном скрипте, который выполняется в момент, когда выполнение логики приостановлено.

Системный тег $LogicRunning

Данный тег осуществляет мониторинг и/или управляет выполнением скриптов.

Использование

$LogicRunning

Примечание

Установка значения в 0 запускает выполнение скриптов. Установка значения в 1 останавливает выполнение скриптов.

Изменение значение данного системного тега идентично нажатию в Halt Logic или Start Logic в меню Logic приложения WindowViewer.

Невозможно остановить выполнение асинхронные скрипты, которые выполняются в данный момент. Однако, запуск новых асинхронных скриптов осуществляться не будет.

Невозможно остановить или запустить скрипты ArchestrA символов, нажав Halt Logic или Start Logic или записав значение в системный тег $LogicRunning.

Тип

Discrete (Read/Write) Дискретный (Чтение/Запись)

Глава 4

Язык программирования

Ниже приведены правила синтаксиса и способы написания скриптов, используя скриптовый язык программирования InTouch HMI.

- Базовые правила синтаксиса. См. раздел Базовые правила синтаксиса.
- Вызов заранее определенных или специальных функций. См. раздел Стандартные функции и Вызов QuickFunctions.
- Использование присвоения значений и различные операторы. См. раздел Присвоение значений и Операторы.
- Использование условных операторов. См. раздел Использование условных операторов в разветвленных структурах.
- Использование циклов. См. раздел Использование циклов.
- Использование локальных переменных. См. раздел Использование локальных переменных.

Для более подробной информации по работе с редактором скриптов, см. Глава 2, Создание и редактирование скриптов.

Для более подробной информации по различным типам условий срабатывания скриптов, см. Глава 3, Типы скриптов.

Для более подробной информации по стандартным функциям, см. Глава 6, Встроенные функции.

Базовые правила синтаксиса

Базовые правила синтаксиса касаются следующих аспектов скриптового языка InTouch HMI.

- Начало и конец скрипта
- Подпрограммы
- Предписания
- Структурирование текста
- Комментарии
- Ссылки на теги
- Литерные данные
- Выражения
- Проверка синтаксиса

Начало и конец скрипта

Начало и конец скрипта декларировать не надо. “Скрипт” – это текст, содержащийся в окне редактора скриптов.

Подпрограммы

Внутри одного скрипта нет такого понятия как отдельная подпрограмма, как процедура “Sub” в VisualBasic. Для создания скрипта с несколькими подпрограммами, необходимо использовать QuickFunctions для каждой подпрограммы. См. Глава 5 QuickFunctions.

Инструкция

- Инструкция может быть присвоением значения, вызовом функции или управляющей структурой.
- Каждая инструкция в скрипте должна заканчиваться точкой с запятой (;).
- Можно в одной строке разместить несколько инструкций, только каждое инструкция должна заканчиваться точкой с запятой.
- Можно распределить инструкции на несколько строк, нажатием кнопки Enter.

Структурирование текста

Можно структурировать код скрипта любым способом. Структурирование текста не имеет функциональной значимости.

Комментарии

Для того чтобы ввести текст в качестве комментария, необходимо ограничить его двумя фигурными скобками {}.

Ссылки на теги

Существует несколько способов сделать ссылки на теги.

- Для ссылки на тег, который определен в локальном словаре тегов, просто ввести имя тега.
- Для ссылки на определенное поле тега, необходимо использовать следующий формат: `Tagname.Dotfield`.
- Для ссылки на тег удаленного узла, необходимо использовать следующий формат: `AccessName:Item`.
- Можно также использовать локальные переменные, но их область действия ограничена текущим скриптом. См. раздел Использование локальных переменных.

Литеральные данные

- Можно задавать целочисленные значения в десятичном или шестнадцатеричном представлении. Например, `255` или `0xFF`.
- Можно задавать значения с плавающей точкой в десятичном или экспоненциальном представлении. Например, `0.001` или `1E-3`.
- Можно задавать логическое значение, используя 0 для FALSE и 1 для TRUE.
- Можно задавать строковое значение, просто введя текст ограниченный кавычками. Например, `“This is a string.”`

Выражения

Выражения могу включать в себя литерные данные, ссылки на теги и вызовы функций. Все это соединяется в единое выражение соответствующими операторами. См. раздел Присвоение значений и Операторы.

Проверка синтаксиса

При сохранении скрипта, редактор автоматически проверяет его на корректность синтаксиса. Можно также осуществить проверку вручную, нажав кнопку Validate. См. раздел Проверка синтаксиса в скрипте.

Вызов стандартных функций

Стандартные функции это заранее определенные в InTouch HMI функции. Вызов специальных функций (QuickFunctions).

Синтаксис для вызова стандартных функций

Синтаксис для вызова стандартных функций зависит от того, будет ли и как функция возвращать результат.

Некоторые функции не возвращают результата; некоторые функции возвращают необязательный результат, который может быть присвоен тегу или использован в выражении; некоторые функции возвращают результат, который должен быть присвоен тегу или использован в выражении.

Для определения типа функции, необходимо просмотреть описание функции. Каждое описание функции имеет описание синтаксиса, в котором указано, возвращает ли функция результат и является результат необязательным.

Для вызова функции, которая не возвращает результата

- Использовать только имя функции (и параметры, если это необходимо) в предписании. Например:

```
FunctionName(Parameters);
```

Для вызова функции, которая требует присвоения результата

- Использовать имя функции (и параметры, если это необходимо) в любом месте в скрипте, где можно использовать литерное значение или тег соответствующего типа данных. Например, в присвоении значения:

```
ResultTagname = FunctionName(Parameters);
```

Или вложенный вызов функции, используя результат как параметр для другой функции:

```
OtherFunction(FunctionName(Parameters));
```

Для вызова функции, которая возвращает необязательный результат

- Использовать одну из двух вышеперечисленных процедур.

Передача параметров в функцию

Параметры в стандартную функцию обычно передаются в качестве *значения*. Это значит, что Вы можете передать в качестве параметра любое корректное выражение, если только результат выражения имеет тот же тип данных, что и параметр. Такие выражения могут включать литерные значения, ссылки на теги и вызовы функций, а все вместе это соединено соответствующими операторами. Для более подробной информации по выражениям и операторам, см. раздел Присвоение значений и Операторы.

Когда в скрипте вызывается функция, выражение вычисляется и результат передается в функцию.

Однако, существуют некоторые функции которые требуют тег в качестве параметра. Например:

```
RecipeSelectRecipe(Filename, RecipeName, Number);
```

В данном примере, параметр RecipeName должен быть тегом (то есть, надо использовать строковый тег для параметра RecipeName). Вы не можете вместо этого использовать строковое выражение, даже если выражение определяет правильный тег.

Примечание Некоторые стандартные функции, унаследованные от предыдущих версий, имеют только один параметр (например, функция Ack()), и не придерживаются стандартного синтаксиса передачи в скобочках. Вместо этого, параметр отделен от имени функции пробелом. Смотрите описание функции в документации, если имеются сомнения в написании синтаксиса функции.

Вызов пользовательских Quick функций

Вызов пользовательских Quick функций немного отличается от вызова стандартных функций.

- Ключевое слово CALL должно стоять перед именем Quick функции.
- Результаты, возвращаемые Quick функцией всегда опциональны; можно использовать их, но это необязательно.

Чтобы вызвать Quick функцию, которая не возвращает результата

♦ В предписании необходимо ввести ключевое слово CALL перед именем функции (и параметром, если есть).

```
CALL QuickFunctionName (Parameters);
```

Чтобы вызвать Quick функцию, которая возвращает результат

♦ можно сделать следующими способами:

- Вызвать Quick функцию, так же как и если бы она не возвращала результат (см. процедуру выше).
- Использовать ключевое слово CALL и имя функции (и параметр, если есть) в любом месте в скрипте, где можно использовать литерное значение или тег соответствующего типа. Например, присвоение значение:

```
ResultsTagname = CALL
QuickFunctionName (Parameters);
```

Или при вложенном вызове функции, стандартная функция используется в качестве параметра.

```
OtherFunction(CALL FunctionName(Parameters));
```

Примечание Нельзя использовать вложенные Quick функции. То есть нельзя вызывать Quick функцию, которая в качестве параметра вызывает другую Quick функцию. Например:

```
CALL QF1(CALL QF1()); - Неправильно.
```

Передача параметров в Quick функцию

Параметры в Quick функцию всегда передаются в качестве значения. Нельзя передать параметр в Quick функцию ссылкой.

Можно передать в качестве параметра любое корректное выражение, если только результат выражения имеет тот же тип данных, что и параметр. Такие выражения могут включать литерные значения, ссылки на теги и вызовы функций, а все вместе это соединено соответствующими операторами. Для более подробной информации по выражениям и операторам, см. раздел Присвоение значений и Операторы.
Когда в скрипте вызывается функция, выражение вычисляется и результат передается в функцию.

Примечание Нельзя использовать вложенные Quick функции. То есть нельзя вызывать Quick функцию, которая в качестве параметра вызывает другую Quick функцию. Например:

```
CALL QF1(CALL QF1()); - Неправильно.
```

Присвоение значений и Операторы

В скрипте можно использовать присвоение значений для записи значения в тег. Синтаксис присвоения значения будет следующим:

```
Tagname = ValueExpression;
```

После выполнения данной инструкции `ValueExpression` будет записано в тег `Tagname`. `ValueExpression` может быть любым корректным выражением, тип данных которого совпадает с типом данных тега. `ValueExpression` может включать литерные значения, ссылки на теги и вызовы функций, а все вместе это соединено соответствующими операторами.

См. раздел Поддерживаемые Операторы.

См. раздел Установка порядка выполнения Операторов.

См. раздел Примеры выражений.

Поддерживаемые Операторы

В таблице ниже приведены все поддерживаемые операторы. Для более подробной информации по использованию определенного оператора, смотрите соответствующий раздел.

Оператор	Подробная информация
+	Сложение или конкатенация: +
-	Вычитание или обратный знак: -
*	Умножение: *
/	Деление: /
**	Возведение в степень: **
MOD	Остаток целочисленного деления: MOD
~	Дополнение: ~
SHL	Сдвиг влево: SHL
SHR	Сдвиг вправо: SHR
&	Побитовая операция AND: &
|	Побитовая операция OR: |
^	Побитовая операция XOR: ^
AND	Логическое умножение: AND
OR	Логическое сложение: OR
NOT	Логическое отрицание: NOT
<	Операция сравнения: <
>	Операция сравнения: >
<=	Операция сравнения: <=
>=	Операция сравнения: >=
==	Операция сравнения: ==
<>	Операция сравнения: <>

Примечание Для числовых вычислений, всегда необходимо выбирать операторы таким образом, чтобы результат вычислений был внутри диапазона значений вещественного числа. Иначе, результат не будет правильным.

Сложение или конкатенация: +

Складывает два числовых операнда или конкатенирует (соединяет) два строковых операнда.

Допустимые типы операндов:

Для сложения: любое значение типа Integer или Real.

Для конкатенации: Любое значение типа Message.

Тип данных, возвращаемого значения:

Для сложения: Integer или Real.

Для конкатенации: Message.

Пример:

```
MessageTag = "Setpoint value: " + Text(SetpointTag,
"#.##");
```

Вычитание или обратный знак: -

При использовании с двумя операндами, осуществляет обычное числовое вычитание. При использовании с одним операндом, меняет знак операнда.

Допустимые типы операндов:

Любое значение типа Integer или Real.

Тип данных, возвращаемого значения:

Integer или Real.

Пример: В данном примере OriginalValue равное 70, а InvertedValue становится равным -70. Если OriginalValue равное -70, то InvertedValue становится равным 70.

```
InvertedValue = -OriginalValue;
```

Умножение: *

Обычное численное умножение.

Допустимые типы операндов:

Любое значение типа Integer или Real.

Тип данных, возвращаемого значения:

Integer или Real.

Деление: /

Обычное численное деление. При попытке разделить на 0, результат будет равным 0.

Допустимые типы операндов:

Любое значение типа Integer или Real.

Тип данных, возвращаемого значения:

Integer или Real.

Возведение в степень: **

Возводит левый операнд в степень правого операнда.

Допустимые типы операндов:

Любое значение типа Integer или Real. Невозможно взвести 0 в отрицательную степень, или отрицательное число в дробную степень. В таких случаях, возвращаемый результат всегда будет равным 0.

Тип данных, возвращаемого значения:

Integer или Real.

Пример:

`8 ** (1/3)` выражение будет равно 2 (корень кубический из 8)

Остаток целочисленного деления: MOD

Возвращает остаток от деления двух целочисленных значений.

Допустимые типы операндов:

Любое значение типа Integer.

Тип данных, возвращаемого значения:

Integer

Пример:

`37 MOD 4` вернет `1.`

Дополнение: ~

Возвращает поразрядное дополнение целочисленного числа. То есть конвертирует каждый бит 0 в 1, и наоборот 1 в 0.

Допустимые типы операндов:

Любое значение типа Integer.

Тип данных, возвращаемого значения:

Integer

Сдвиг влево: SHL , Сдвиг вправо: SHR

Сдвигает влево или вправо двоичное представление целочисленного значения на указанное количество битовых позиций. Левый операнд – сдвигаемое значение, правый операнд – количество битовых позиций. Сдвинутые биты теряются. Свободные битовые позиции после сдвига заполняются 0.

Допустимые типы операндов:

Любое значение типа Integer.

Тип данных, возвращаемого значения:

Integer

Пример:

`IntTag = IntTag SHL 1;` следующие результаты получаются, если выполнить операцию несколько раз, для 5.

Этапы	Побитовое представление	Значение тега
Начальное значение	0[…]00000101	5
После 1 выполнения	0[…]00001010	10
После 2 выполнения	0[…]00010100	20

Побитовая операция AND: &

Сравнивает побитовое представление двух целый чисел (Integer), по битам, и возвращает результат в соответствии со следующей таблицей.

Бит первого операнда	Бит второго операнда	Бит результата
0	0	0
0	1	0
1	0	0
1	1	1

Можно использовать данный оператор, для того чтобы быстро “наложить маску” (установить в 0), определенные части битового представления. Например, следующая инструкция “заглушает” первые 24 бита тега IntTag.

```
IntTag = IntTag & 255;
```

Как показано в таблице, результирующий бит всегда 0, если один битов операнда 0. В двоичном представлении 255 – это младшие 8 бит 1, а оставшиеся 24 бита - равны 0, таким образом, в результате соответствующие биты будут установлены в 0.

Допустимые типы операндов:

Любое значение типа Integer.

Тип данных, возвращаемого значения:

Integer

Побитовая операция OR: |

Сравнивается битовое представление целых чисел (integer), побитово, и возвращает результат в соответствии с таблицей:

Бит первого операнда	Бит второго операнда	Бит результата
0	0	0
0	1	1
1	0	1
1	1	1

Такая операция называется включающее "или".

Допустимые типы операндов:

Любое значение типа Integer.

Тип данных, возвращаемого значения:

Integer

Побитовая операция XOR: ^

Сравнивается битовое представление целых чисел (integer), побитово, и возвращает результат в соответствии с таблицей:

Бит первого операнда	Бит второго операнда	Бит результата
0	0	0
0	1	1
1	0	1
1	1	0

Такая операция называется исключающее "или".

Допустимые типы операндов:

Любое значение типа Integer.

Тип данных, возвращаемого значения:

Integer

Логическое умножение: AND

Возвращает TRUE, если оба дискретных операнда – TRUE; иначе, возвращает FALSE. Ниже представлена таблица истинности.

p	q	p AND q
False	False	False
False	True	False
True	False	False
True	True	True

Допустимые типы операндов:

Любое значение типа Discrete.

Тип данных, возвращаемого значения:

Discrete

Логическое сложение: OR

Возвращает TRUE, если хотя бы один из дискретных операндов – TRUE; иначе, возвращает FALSE. Ниже представлена таблица истинности.

p	q	p OR q
False	False	False
False	True	True
True	False	True
True	True	True

Допустимые типы операндов:

Любое значение типа Discrete.

Тип данных, возвращаемого значения:

Discrete

Логическое отрицание: NOT

Возвращает TRUE, если дискретный операнд – FALSE и наоборот. Ниже представлена таблица истинности.

p	NOT q
False	True
True	False

Допустимые типы операндов:

Любое значение типа Discrete.

Тип данных, возвращаемого значения:

Discrete

Операция сравнения: <, >, <=, >=, ==, <>

Операции сравнения двух значений, и возвращает TRUE, если условие удовлетворяется. Операнды могут быть иметь любой тип данных. Для строковых типов данных, сравнение основано на буквенном порядке, не зависимо от регистра, где b больше a, c больше b и так далее. Для дискретных операндов TRUE больше, чем FALSE. В таблице ниже приведены все операторы сравнения вместе с условиями.

Операция	Пример	Условие
Меньше чем	a < b	a меньше чем b
Больше чем	a > b	a больше чем b
Меньше или равно	a < b	a меньше или равно b
Больше или равно	a > b	a больше или равно b
Равно	a == b	a равно b
Не равно	a <> b	a не равно b

Допустимые типы операндов:

Любое значение любого типа данных (но оба значения должны быть одного типа данных).

Тип данных, возвращаемого значения:

Discrete

Порядок выполнения операторов

В любом выражении, можно при помощи скобочек устанавливать порядок выполнения операторов. Работает это так же как и любом математическом выражении. Если не использовать скобочки, то выражение будет выполняться по базовым правилам порядка выполнения операторов. Операция с наивысшим уровнем приоритета будет выполняться в первую очередь, затем со следующим по приоритету и так далее.

В таблице ниже приведен уровень приоритета для каждого оператора. Операторы на одной строчке имеют одинаковый уровень приоритета.

-, NOT, ~	Наивысший приоритет
**	
*, /, MOD	
+, -	
SHL, SHR	
<, >, <=, >=	
<>, ==	
&	
^	
|	
AND	
OR	
=	Самый низший приоритет

Подразумеваемое преобразование типов данных

Язык программирования InTouch HMI, поддерживает подразумеваемое преобразование типов при присвоении определенных типов данных. Однако, это может привести к непредсказуемым результатам, и необходимо использовать данную возможность аккуратно.

В таблице ниже приведено, что происходит при присвоении тегу одного типа, тега другого типа.

Ожидаемый тип данных	Используемый тип данных	Примечания
Discrete	Integer	Значение 0, интерпретируется как FALSE. Любое другое значение интерпретируется как TRUE.
Discrete	Real	Значение 0, интерпретируется как FALSE. Любое другое значение интерпретируется как TRUE.
Integer	Discrete	Значение FALSE преобразуется в 0. А TRUE преобразуется в 1.
Integer	Real	Только значение до десятичной точки. Десятичные знаки отбрасываются.
Real	Discrete	Значение FALSE преобразуется в 0. А TRUE преобразуется в 1.
Real	Integer	Данные представляются без изменений.

Для дополнительной информации использования функций преобразования между другими типами данных смотрите раздел Преобразование типов данных.

Примеры выражений

В таблице ниже приведены некоторые корректные выражения, вместе с результатами выражений и результирующими типами данных.

Выражение	Типа данных результата	Результат
`37 MOD 4`	Integer	1
`37 MOD 4 == 1`	Discrete	TRUE
`NOT (37 MOD 4 == 1)`	Discrete	FALSE
`InfoAppActive(InfoAppTitle("xyz")) == 1`	Discrete	TRUE, если процесс “xyz” запущен
`"Batch " + Text(IntTag, "000")`	Message	“Batch 010”, если значение тега IntTag равно 10.

В таблице ниже приведены некоторые некорректные выражения, вместе с причинами, почему они являются не правильными.

Выражение	Типа данных результата Результат
`NOT (37 MOD 4)`	Для NOT необходим дискретный операнд
`NOT 37 MOD 4 == 1`	Так как NOT имеет более высокий приоритет обработки операций, InTouch HMI, пытается выполнить NOT для значения 37, вместо дискретного результата сравнения.
`"Batch " + IntTag`	При использовании оператора + для конкатенации строк оба операнда должны быть строками.

Использование условных операторов в разветвленных структурах

Можно динамически контролировать выполнение скрипта, если в нем используются удовлетворение определенных условий. Для этих целей InTouch HMI поддерживает управляющие структуры IF-THEN-ELSE.

Базовый синтаксис управляющей структуры IF-THEN-ELSE следующий:

Синтаксис

```
IF Condition THEN
... инструкция и/или другая IF-THEN-ELSE структура
[ELSE
... инструкция и/или другая IF-THEN-ELSE структура]
ENDIF;
```

При работе с IF-THEN-ELSE структурами необходимо помнить следующие правила:

- IF-THEN-ELSE структуры могут быть вложенными как в разделе THEN так и в разделе ELSE.
- Для каждого IF необходимо предусмотреть закрывающий ENDIF.
- *Condition* – условием должно быть корректное выражение, принимающее дискретное значение. Раздел THEN выполняется, если условие равно TRUE. Раздел ELSE выполняется, если условие равно FALSE.
- Раздел ELSE является опциональным.
- Некоторые языки программирования позволяют проверять несколько условий в одном иерархическом уровне IF-THEN-ELSE структуры и иметь один главный ELSE, который выполняется, если все условия равны FALSE (например, в Visual Basic). Но это невозможно в InTouch HMI. Для каждого проверяемого условия, необходимо открывать новую IF-THEN-ELSE структуру. Таким образом, для того чтобы получить отдельный простой раздел ELSE для всех условий, необходимо разместить раздел ELSE в IF-THEN-ELSE структуре в последнем уровне вложенности.

Простая структура с условными операторами

Ниже приведен скрипт простой структуры с условными операторами. Если тег SuccessTag равен TRUE, то открывается окно «Success», иначе открывается окно «Failure».

```
IF   SuccessTag == 1 THEN
     Show "Success";
ELSE
     Show "Failure";
ENDIF;
```

Вложенные структуры с условными операторами

Ниже приведен скрипт, показывающий проверку нескольких условий и имеющий один основной раздел ELSE, с кодом, который выполняется если ни одно из перечисленных условий не удовлетворяется.

```
IF   ChoiceTag == 1 THEN
     Show "Procedure 1";
ELSE
     IF   ChoiceTag == 2 THEN
          Show "Procedure 2";
     ELSE
          IF   ChoiceTag == 3 THEN
               Show "Procedure 3";
          ELSE
               Show "Default Procedure";
          ENDIF;
     ENDIF;
ENDIF;
```

Примеры неправильных скриптов (нет ENDIF)

Если Вы имели знакомство с Visual Basic, то возможно Вы попытаетесь написать следующую IF инструкцию.

```
IF OpenThisWindow == 1 THEN Show "This Window";
```

Данная конструкция не работает в InTouch HMI. Так как для каждого `IF` необходимо предусмотреть закрывающий `ENDIF`.

Примеры неправильных скриптов (неправильная вложенность)

Если Вы имели знакомство с Visual Basic, то возможно Вы попытаетесь написать следующую конструкцию с несколькими условиями и условием «по умолчанию»:

```
IF   ChoiceTag == 1 THEN
     Show "Procedure 1";
ELSE IF ChoiceTag == 2 THEN
     Show "Procedure 2";
ELSE IF ChoiceTag == 3 THEN
     Show "Procedure 3";
ELSE
     Show "Default Procedure";
ENDIF;
```

Данная конструкция не работает в InTouch HMI. Так как каждый `IF` открывает один уровень вложенности и должен иметь соответствующий `ENDIF`. Правильную версию примера смотрите в разделе Вложенные структуры с условными операторами.

Использование циклов

Циклы позволяют выполнить определенную часть кода многократно. InTouch HMI поддерживает только циклы FOR. Цикл FOR работает путем отслеживания значения переменной цикла, которая инкрементируется или декрементируется на каждой итерации цикла. Цикл выполняется пока значение переменной цикла не достигнет фиксированного предела.

Синтаксис

```
FOR  LoopTag = StartExpression TO EndExpression [STEP
     ChangeExpression]
... statements or another FOR loop ...
NEXT;
```

- Выражения `StartExpression, EndExpression` и `ChangeExpression` вместе определяют количество итераций.
- `StartExpression` устанавливает начальное значение диапазона цикла. `EndExpression` устанавливает конечное значение диапазона цикла.
- `STEP ChangeExpression` опционально устанавливает значение инкремента или декремента переменной цикла во время итерации; если это значение не определять, то по умолчанию оно будет равно 1.

При выполнении цикла FOR, InTouch HMI выполняет следующее:

1. Устанавливает значение LoopTag равным StartExpression.
2. Проверяет, является ли значение LoopTag больше чем EndExpression. Если это так, то InTouch HMI производит выход из цикла. (Если ChangeExpression отрицательное, то InTouch HMI проверяет, является ли значение LoopTag меньше чем EndExpression).
3. Выполняет инструкции внутри цикла.
4. Инкрементирует LoopTag на значение ChangeExpression (на 1, если не определено другое значение).
5. Повторяет шаги с 2 по 4.

При работе с циклами FOR необходимо соблюдать следующие правила:

- Циклы FOR могут быть вложенными. Максимальное количество вложенных циклов зависит от ресурсов системы и памяти.
- Для каждого FOR должен быть закрывающий NEXT. NEXT всегда применяется к ближайшему FOR, в том же уровне вложенности.
- LoopTag должен быть числовым тегом (или локальной переменной).
- Выражения StartExpression, EndExpression и ChangeExpression должны быть корректными выражениями которые возвращают числовой результат.
- Если ChangeExpression положительное число, то EndExpression должно быть больше StartExpression; если ChangeExpression отрицательное число, то StartExpression должно быть больше EndExpression . Иначе цикл работать не будет.
- Для циклов существует временной предел выполнения. Смотрите раздел Выполнение циклов.

Внимание Выполнение циклов влияет на процессе в среде исполнения. Для более подробной информации, см. раздел Влияние циклов на другие процессы в среде исполнения.

Принудительный выход из цикла

Выход из цикла можно выполнить в любой момент, вызовом следующей инструкции:

```
EXIT FOR;
```

Данная инструкция приводит к выполнению в скрипте инструкции следующей сразу после NEXT.

Пример

Ниже приведен фрагмент кода, который использует цикл для вставки большого количества пустых записей в таблицу базы данных. Если при вставке записей произошла ошибка, то осуществляется выход из цикла для предотвращения создания большего количества ошибок.

```
FOR Counter = 1 TO 1000
      ResultCode = SQLInsert(ConnectionID, "BatchDetails",
      "BindList1");
      IF ResultCode <> 0 THEN
            LogMessage("Error creating records!
            Aborting...");
            EXIT FOR;
      ENDIF;
NEXT;
```

Влияние циклов на другие процессы в среде исполнения

Пока цикл FOR выполняется, все другие процессы среды исполнения WindowViewer приостанавливаются. Это включает следующие области:

- Обновление экрана (анимационные связи, отображение значений, тренды и т.д.). Это значит, что нельзя использовать циклы FOR для анимации объектов, потому как движения производится не будет, пока не завершится цикл.
- I/O обмен данными. Например, если в цикле FOR было изменено значение тега ввода/вывода, то значение будет записано в устройство ввода/вывода только после последней итерации.
- Другие скрипты, включая асинхронные Quick функции.

Избежать приостановки выполнения других процессов среды исполнения, можно поместив цикл FOR в асинхронную Quick функцию.

Предельное время работы цикла

Для того чтобы избежать бесконечного выполнения цикла существует предельное время работы цикла FOR, за которое цикл должен завершить свою работу. Если цикл не завершился, после время истекло, то WindowViwer производит автоматическое завершение цикла и записывает соответствующее сообщение в Log Viewer.

По умолчанию значение предельного времени равно 5 секунд. Изменить это значение, можно добавив следующую строчку в файл intouch.ini в директории приложения:

LoopTimeout = x

Где x – это предельное значение времени в секундах.

> **Внимание** Предельное время проверяется только после инструкции NEXT цикла. Таким образом, первая итерация цикла будет выполнена в любом случае, даже если это займет время большее, чем предельное время.

Примеры циклов

Следующий скрипт использует простой цикл и индиректный тег для инициализации 100 тегов (с Tag001 по Tag100), значение которых равно 0.

```
DIM Counter AS INTEGER;

FOR  Counter = 1 TO 100
     IndirectInteger.Name = "Tag" + Text(Counter, "000");
     IndirectInteger.Value = 0;
NEXT;
```

Следующий скрипт использует два вложенных цикла и индиректный тег для инициализации 1000 тегов (с Line01_Tag001 по Line10_Tag100), значение которых равно 0.

```
DIM LineCounter AS INTEGER;
DIM TagCounter AS INTEGER;

FOR  LineCounter = 1 TO 10
     FOR  TagCounter = 1 TO 100
          IndirectInteger.Name = "Line" +
          Text(LineCounter, "00") + "_Tag" +
          Text(TagCounter, "000");
          IndirectInteger.Value = 0;
     NEXT;
NEXT;
```

Использование локальных переменных

В скрипте можно объявлять локальные переменные для сохранения временных или промежуточных результатов. Это позволит улучшить производительность и поможет сэкономить теги. Использовать локальные переменные, можно также как и теги в скрипте. Однако, есть определенные отличия:

- Локальные переменные существуют только пока работает скрипт, в котором они объявлены. Когда выполнение скрипта заканчивается значение в локальных переменных теряется. Эти переменные не могут иметь ссылки на другие скрипты в приложении.
- Локальные переменные не имеют полей.
- Локальные переменные не учитываются счетчиком тегов.

Перед тем как использовать локальные переменные в скрипте, необходимо их объявить; иначе, локальная переменная будет рассматриваться как тег. См. раздел Объявление локальных переменных.

Можно объявлять локальные переменные, которые будут иметь такое же имя, как и уже существующий тег. См. раздел Конфликт имен между локальными переменными и тегами.

Объявление локальных переменных

Объявлять локальные перемененные можно в любом месте в скрипте, если объявление производится раньше использования. Для того чтобы объявить локальную переменную, используется следующая инструкция:

```
DIM LocVarName [AS DataType];
```

LocalVarName – это имя локальной переменной. Имя должно следовать соглашению по присвоению имён для тегов. Для более подробной информации смотрите Главу 2 в *Руководстве по управлению данными InTouch® HMI*.

DataType – это тип данных локальной переменной. Корректными значениями являются `Discrete`, `Integer`, `Real` и `Message`. Если типа данных не определять, то по умолчанию будет тип данных `Integer`.

Не обходимо использовать отдельный `DIM` для объявления каждой локальной переменной.

Объявлять можно любое количество локальных переменных. Количество ограничено только размером доступной памяти.

Примеры

Для объявления переменной типа Integer:

```
DIM MyLocalIntVar AS Integer;
```

Для объявления нескольких переменных типа Real:

```
DIM MyLocalRealVar1 AS Real;
DIM MyLocalRealVar2 AS Real;
```

Следующая инструкция НЕ правильна:

```
DIM MyLocalRealVar1, MyLocalRealVar2 AS Real;
```

Конфликт имен между локальными переменными и тегами

Можно объявлять локальные переменные, которые будут иметь такое же имя, как и уже существующий тег. Однако, когда в скрипте производится обращение к данному имени, локальная переменная будет отрабатываться вместо тега. Например, предположим есть тег типа Integer с именем “iTag” и есть следующий скрипт:

```
DIM iTag as Integer;
iTag = 20;
```

В данном скрипте инструкция присваивает значение только локальной переменной. Значение тега с таким же именем остается неизменным.

Глава 5

Пользовательские скриптовые функции

Quick функции в InTouch HMI это скрипты , которые в других средах разработки могут быть известны как *макросы*, *подпрограммы* или *процедуры*.

О Quick функциях

Quick функции это скрипты, которые вызываются из других скриптов или анимационных связей. Главным преимуществом Quick функций, является сокращение повторяющегося кода.

Можно передавать значения в Quick функцию, которая использует эти значения и возвращает результат.

Quick функции могут выполняться асинхронно. В отличие от других скриптов, они могут выполняться в фоновом режиме, без прерывания потока главного приложения. Асинхронная Quick функция может быть использована для затратных по времени операций, например, при работе с SQL базами данных.

> **Примечание** Необходимо очень аккуратно разрабатывать Quick функции и их аргументы, так как если Вы хотите изменить аргументы в Quick функции, то сначала надо удалить все вызовы данной Quick функции из каждого скрипта. После внесения изменений, необходимо возвратить обратно все вызовы Quick функций в скрипты. См. примечания в разделе Конфигурирование Quick функций.

Имеются три базовые части Quick функций

- Имя
- Аргументы (опционально)
- Код скрипта с опциональным возвращаемым значением.

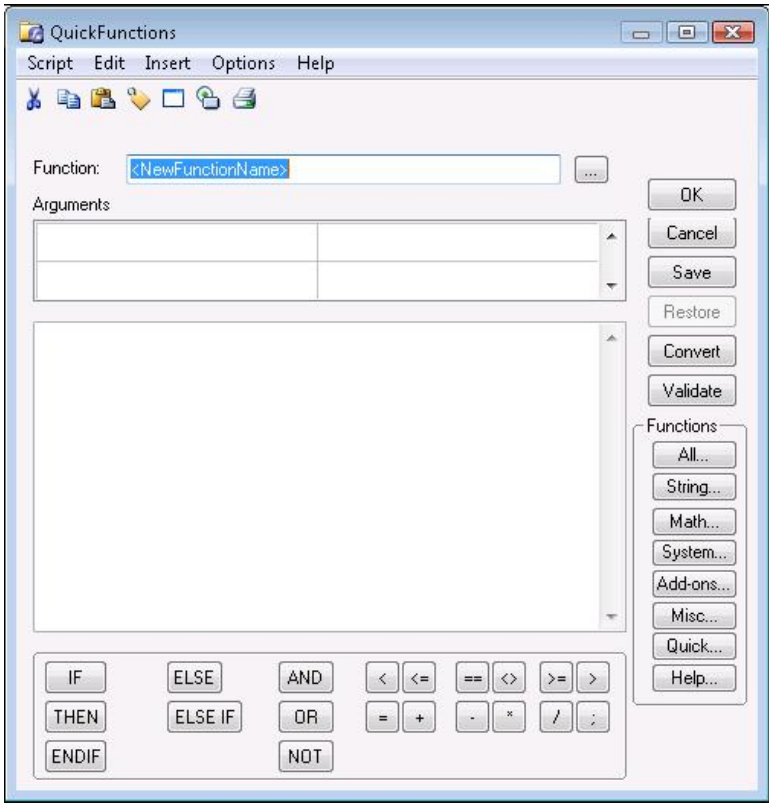
Quick функции выполняются при использовании функции CALL в скрипте или анимационной связи. См. раздел Вызов Quick функций.

Конфигурирование Quick функций

Можно создать, изменить или удалить Вызов Quick функции.

Чтобы создать Quick функцию

1. В панели навигации, в разделе Scripts, нажать на QuickFunctions и в контекстном меню выбрать New (Новый). Появится окно редактора.



2. В поле Function, ввести имя Quick функции.

3. В разделе Arguments, для каждого аргумента ввести имя слева и тип справа.

   Аргументы это локальные переменные, существуют только в Quick функции, в которой они объявлены. На одну Quick функцию можно объявить только 16 аргументов. Длина имен аргументов - 31 символ, без пробела. Имя аргумента должно быть уникальным.

4. Написать скрипт в окне.

5. Для того чтобы Quick функция возвращала результат, необходимо в скрипт добавить: RETURN *value*

   Value может быть литеральным значением, локальной или глобальной (тегом) переменной или вычисляемым выражением. Скрипт завершает работу на команде RETURN и продолжает на функции вызова.

6. Нажать ОК

Чтобы изменить Quick функцию

1. В панели навигации, в разделе Scripts, нажать на QuickFunctions и в контекстном меню выбрать Edit (Редактировать). Появится окно редактора.

2. Внести необходимые изменения и нажать ОК.

Примечание Нельзя внести изменения в список аргументов, если в приложении InTouch есть вызовы Quick функций. Необходимо сначала удалить все вызовы, закрыть все окна InTouch и выполнить операцию Update Use Counts.

Чтобы удалить Quick функцию

1. Необходимо удалить все вызовы Quick функций, закрыть все окна InTouch и выполнить операцию Update Use Counts.

2. В панели навигации, в разделе Scripts, нажать на QuickFunctions и в контекстном меню выбрать Delete (Удалить). Нажать ОК.

Вызов Quick функций

Можно сконфигурировать скрипты и анимационные связи для вызова Quick функций, а затем для отображения или обработки возвращаемого результата.

Quick функция не вызывается если значения параметра не изменились. Можно использовать $Second в качестве параметра, для того чтобы обеспечить выполнение Quick функции каждую секунду.

Для более подробной информации смотрите раздел Вызов пользовательских Quick функций.

Создание асинхронной Quick функции

Можно определить Quick функцию таким образом, чтобы она выполнялась асинхронно (т.е. в параллельном потоке) главному потоку программы.

Чтобы создать асинхронную Quick функцию

1. Создать в редакторе скриптов Quick функцию.
2. В меню Options, нажать Asynchronous.

Ограничения асинхронной Quick функции

Невозможно:

- Возвращать значение от асинхронной Quick функции.
- Запускать более одного экземпляра Quick функции одновременно.
- Остановить асинхронные Quick функции после того как, они начали работать.

Не следует:

- Запускать более трех различных асинхронных Quick функций одновременно. Запуск более трех асинхронных Quick функций одновременно значительно снизит производительность системы.
- Использовать асинхронные Quick функции, в выражениях анимационных связей, т.е. ToolTips, PushButtons и т.д.

Проверка наличия работающей асинхронной Quick функции

Можно произвести проверку на наличие работающих асинхронных Quick функции, при помощи функции IsAnyAsyncFunctionBusy(). Можно использовать данную функцию для создания Quick скрипта который вызовет асинхронную Quick функцию, после того как все асинхронные функции завершат работу.

IsAnyAsyncFunctionBusy()

Возвращает дискретное значение, отображающее имеются ли в наличии работающие асинхронные Quick функции.

Синтаксис

```
result = IsAnyAsyncFunctionBusy (timeout)
```

Аргументы

`result`

Дискретное значение, отображающее имеются ли в наличии работающие асинхронные Quick функции:

- 0 = Нет запущенных асинхронных Quick функций.
- 1 = Запущены асинхронные Quick функции

`timeout`

Количество секунд, до того как будет начата проверка на наличие работающих асинхронных Quick функций. Литеральное Integer значение, тег типа Integer или выражение типа Integer.

Примеры

Предположим, необходимо подключится к нескольким SQL базам данных, используя асинхронные Quick функции, и Вы знаете, что для того чтобы создать эти подключения необходимо 2 минуты.

Во-первых, запустите асинхронную Quick функцию для подключения к SQL базам данных.

Во-вторых, используя функцию IsAnyAsyncFunctionBusy(120) в Quick скрипте, для того чтобы оставить достаточно времени для всех подключений к SQL базам данных до того как Quick скрипт будет завершен.

Если по истечению 2 минут, подключения не осуществлены и асинхронная функция все еще выполняется пытаясь подключиться, функция IsAnyAsyncFunctionBusy() вернет 1.

Для оператора можно отобразить сообщение, что SQL подключение не удалось. Скрипт ниже выполняет данный сценарий:

```
IF IsAnyAsyncFunctionBusy(120) == 1 THEN
SHOW "SQL Connection Error Dialog";
ENDIF;
```

Остановка Асинхронных Quick функций в среде исполнения

Невозможно остановить асинхронные Quick функции, после того как они уже запущены, но можно предотвратить последующий запуск асинхронных Quick функций, путем остановки скриптовой логики. Это повлияет на все Quick скрипты в приложении InTouch.

Для более подробной информации по остановке выполнения скриптов, смотрите раздел Приостановка Выполнения скриптов в режиме исполнения.

Глава 6

Встроенные функции

Встроенные функции InTouch, позволяют выполнить команды и логические операции, основанные на определенных критериях. Встроенные функции можно использовать самостоятельно, и выполнять их в скриптах и анимационных связях.

Принудительное обновление анимационной связи

Если использовать встроенную функцию в анимационной связи, то анимационная связь обновляется, если только обновляется тег, привязанный к ним. Этот тег ведет себя как триггер, как только его значение изменяется. Можно использовать системные теги $Second и $Minute для обновления анимационных связей.

Для обновления анимационной связи:

1. Открыть анимационную связь в окне свойств объекта.
2. Добавить тег триггер (например, $Second) в вычисления. Например:
   - Если анимационная связь содержит целочисленное значение или значение с плавающей запятой, то можно выражение умножить на $Second/$Second.
   - Если анимационная связь содержит строку, можно добавить в выражение SrtringMid($TimeString, 0, 0).
   - Если анимационная связь содержит дискретное значение, то можно добавить ($Second.00 - $Second.00).

Математические вычисления

InTouch HMI поддерживает базовые математические функции, которые можно использовать в скриптах или анимационных связях, такие как:

- Вычисление синуса и косинуса.
- Округление и взятие целой части числа чисел.
- Вычисление логарифмов и экспонент.
- Вычисление квадратного корня числа.

Округление, целая часть и определение знака

В скрипте, можно использовать следующие функции для округления чисел, взятие целой части чисел и определение знака чисел:

Функция	Использование
Abs()	Вычисление абсолютное значения числа или выражения.
Int()	Вычисление целого значения числа выражения
Round()	Округление значения или выражения.
Sgn()	Определение знака (плюс, минус, ноль) значения или выражения.
Trunc()	Возвращает целое часть числа или выражения до десятичной точки.

Функция Abs()

Возвращает абсолютное значение определенного числа. Можно использовать ее преобразователь отрицательного числа в положительное.

Синтаксис

`Result =Abs (number)`

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение.

Примеры

`Abs (14)` возвращает 14

`Abs (-7.5)` возвращает 7.5

Функция Int()

Возвращает целое значение, которое меньше (или равно) указанного числа.

Синтаксис

`Result = Int (number)`

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение.

Примеры

`Int (4.7)` возвращает 4

`Int (-4.7)` возвращает -5

Примечание Для отрицательных чисел, данная функция возвращает целое значение, которое меньше, чем указанное число. Например, Int(-4.7), не -4, а -5. Для получения целой части числа используйте функции Trunc().

Функция Round()

Округляет число с определенной точностью. Результат это число с плавающей точкой.

Синтаксис

`Result = Round (number, precision)`

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение.

precision

Точность, с которой округляется число. Может быть литеральным значением, аналоговым тегом или числовым выражением.

Примеры

`Round(4.3, 1)` возвращает 4

`Round(4.3, 0.01)` возвращает 4.30

`Round(4.5, 1)` возвращает 5

`Round(-4.5, 1)` возвращает -4

`Round(106, 5)` возвращает 105

`Round(43.7, 0.5)` возвращает 43.5.

Функция Sgn()

Возвращает знак числа. Можно использовать для определения, является положительным, отрицательным или нулевым.

Синтаксис

`Result = Sgn (number)`

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение.

Примеры

`Sgn(425)` возвращает 1

`Sgn(0)` возвращает 0

`Sgn(-37.3)` возвращает -1

Функция Trunc ()

Возвращает целое значение числа. Целая часть это часть числа до десятичной точки. Использовать для работы с целой частью дробного числа.

Синтаксис

`Result = Sgn (number)`

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение.

Примеры

`Trunc(4.3)` возвращает 4

`Trunc (-4.3)` возвращает -4

Примечание Также данную функцию можно использовать для работы дробной частью числа. Например:

Result = number - Trunc(number)

Использование тригонометрических функций

В скрипте, можно использовать следующие функции, для произведения тригонометрических вычислений.

Функция	Использование
Sin()	Вычисление синуса угла.
ArcSin()	Вычисление арксинуса значения или выражения.
Cos()	Вычисление косинуса угла.
ArcCos()	Вычисление арккосинуса значения или выражения.
Tan()	Вычисление тангенса угла.
ArcTan()	Вычисление арктангенса значения или выражения.

Примечание Необходимо помнить, что тригонометрические функции в InTouch HMI, используют для вычислений углы в градусах (0-360). Для работы с радианами, необходимо преобразовать передаваемое значение и результирующее значение соответствующим образом.

Функция Sin()

Возвращает значение синуса числа. Для тригонометрических функций числа это углы в градусах.

Синтаксис

```
Result = Sin (number)
```

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение.

Примеры

`Sin (90)` возвращает 1

`Sin (0)` возвращает 0

`Sin (30)` возвращает 0.5

`100 * Sin (6 * $Second)` возвращает синус с амплитудой 100 и периодом одна минута.

Функция ArcSin()

Возвращает значение арксинуса числа. Это функция обратная функции Sin(). Использовать ArcSin() для вычисления углов от -90 до 90 в градусах, синус которых равен данному числу.

Синтаксис

`Result = ArcSin (number)`

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение в диапазоне от -1 до 1.

Примеры

`ArcSin (1)` возвращает 90

`ArcSin (0)` возвращает 0

`ArcSin (0.5)` возвращает 30

Функция Cos()

Возвращает значение косинуса числа. Для тригонометрических функций числа это углы в градусах.

Синтаксис

`Result = Cos (number)`

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение.

Примеры

`Cos (90)` возвращает 0

`Cos (0)` возвращает 1

`Cos (60)` возвращает 0.5

`20 + 50 * Cos (6 * $Second)` возвращает косинус, колеблющийся вокруг 20, с амплитудой 50 и периодом одна минута.

Функция ArcCos ()

Возвращает значение арккосинуса числа. Это функция обратная функции Cos(). Необходимо использовать ArcCos() для вычисления углов от 0 до 180 в градусах, косинус которых равен данному числу.

Синтаксис

`Result = ArcCos (number)`

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение в диапазоне от -1 до 1.

Примеры

`ArcCos (1)` возвращает 0

`ArcCos (-0.5)` возвращает 120

Функция Tan()

Возвращает значение тангенса числа. Для тригонометрических функций числа это углы в градусах.

Синтаксис

`Result = Tan (number)`

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение.

Примеры

`Tan (45)` возвращает 1

`Tan (0)` возвращает 0

Функция ArcTan ()

Возвращает значение арктангенса числа. Это функция обратная функции Tan(). Необходимо использовать ArcTan () для вычисления углов, тангенс которых равен данному числу.

Синтаксис

```
Result = ArcTan (number)
```

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение в диапазоне от -1 до 1.

Примеры

`ArcTan (1)` возвращает 45

`ArcTan (0)` возвращает 0

Возвращение значения числа p

В скрипте, можно использовать функцию Pi(), для использования константы π в вычислениях. Функция Pi() возвращает значение с точностью до 7 знака.

Синтаксис

```
Result = Pi ()
```

Примеры

`Pi ()` возвращает 3.1415927

Вычисления логарифмов

В скрипте, можно использовать следующие функции для вычислений логарифмов и экспоненциальных функций.

Функция	Использование
Log()	Вычисление натурального логарифма числа или выражения.
Exp()	Вычисление экспоненты числа или выражения.
LogN()	Вычисление логарифма значения или выражения, на основании другого значения или выражения.

Функция Log ()

Возвращает значение натурального логарифма определенного положительного числа. Функция обратная Exp().

Примечание Натуральный логарифм 0 или отрицательного числа не определяется. Если в функцию Log() передать значение 0 или отрицательное число, то она вернет результат -99.0000000

Синтаксис

```
Result = Log (number)
```

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение.

Примеры

`Log(100)` возвращает 4.6051702.

`Log(1)` возвращает 0.

Функция Exp ()

Возвращает экспоненциальное значение определенного положительного числа. Функция обратная Log () и эквивалентна возведению *e* в степень.

Примечание Если передать в функцию Exp() значение вне диапазона от -88.72 до 88.72, то она вернет результат -99.0000000

Синтаксис

```
Result = Exp (number)
```

Параметры

number

Литеральное значение, аналоговый тег или числовое выражение d диапазоне от -88.72 до 88.72.

Примеры

`Exp(1)` возвращает 2.7182818.

`Exp(0)` возвращает 1.

Функция LogN ()

Возвращает значение логарифма положительного числа по определенному основанию. Функция обратная основание в степени логарифма.

Синтаксис

`Result = LogN (number, base)`

Параметры

number

Положительное литеральное значение, аналоговый тег или числовое выражение.

base

Положительное литеральное значение, аналоговый тег или числовое выражение не равное 1.

Примеры

`Log(8,2)` возвращает 3.

`Log(num, btag)` возвращает логарифм числа по основанию btag.

Примечание Если в функциюLogN() передать неправильные параметры, она возвратит результат -99.0000000.

Вычисление квадратного корня

В скрипте можно использовать функцию извлечения из под квадратного корня определенного неотрицательного значения.

Примечание Если в функцию Sqrt() передать отрицательное значение, то она возвратит результат -99.0000000.

Синтаксис

`Result = Sqrt (number)`

Параметры

number

Неотрицательно литеральное значение, аналоговый тег или числовое выражение.

Примеры

`Sqrt (36)` возвращает 6.

`Sqrt (perftag)` извлекает из под квадратного корня значение которое содержится в теге perfteg.

Операции со строками

Можно использовать много функций для работы со строками в скриптах и анимационных связях. Можно использовать данные функции для:

- Возвращения части строк.
- Замена регистра строк.
- Работа с ASCII значениями в строках.
- Поиск и замена в строках.
- Сравнение строк.
- Возвращение информации о строках (например, длина строки).

Возвращение части строк

В скрипте можно использовать функции StringLeft(), StringMid() и StringRight() для возвращения части строк.

Функция StringLeft()

Возвращает указанное количество символов с левого края строки.

Синтаксис

`Result = StringLeft(`*`string, length`*`)`

Параметры

`string`

Литеральный текст, строковый тег или строковое выражение.

`length`

Количество символов, которое необходимо вернуть. Литеральный текст, строковый тег или строковое выражение.

Примеры

`StringLeft(“Hello World”,5)` возвращает “Hello”.

`StringLeft(“Hello World”,20)` возвращает “Hello World”.

`StringLeft(“Hello World”,0)` возвращает “Hello World”.

Примечание Если в функцию StringLeft() передать в качестве значения length 0, то она возвратит в результат целую строку.

Функция StringRight()

Возвращает указанное количество символов с правого края строки.

Синтаксис

`Result = StringRight(string, length)`

Параметры

string

Литеральный текст, строковый тег или строковое выражение.

length

Количество символов, которое необходимо вернуть. Литеральный текст, строковый тег или строковое выражение.

Примеры

`StringRight ("Hello World",5)` возвращает "World".

`StringRight ("Hello World",20)` возвращает "Hello World".

`StringRight ("Hello World",0)` возвращает "Hello World".

Примечание Если в функцию StringRight () передать в качестве значения length 0, то она возвратит в результат целую строку.

Функция StringMid()

Возвращает часть строки. Возвращает указанное количество символов с указанной стартовой точки.

Синтаксис

`Result = StringMid(string, startpos, length)`

Параметры

string

Литеральный текст, строковый тег или строковое выражение.

startpos

Стартовая позиция в строке. Литеральный текст, строковый тег или строковое выражение.

length

Количество символов, которое необходимо вернуть. Литеральный текст, строковый тег или строковое выражение.

Примеры

`StringMid ("Hello World",5,4)` возвращает "o Wo".

`StringMid ("Hello World",7,20)` возвращает "World".

`StringMid ("Hello World",4,0)` возвращает "lo World".

Примечание Если в функцию StringMid () передать в качестве значения length 0, то она возвратит в результат целую строку после стартовой позиции.

Изменение регистра строк

В скриптах можно использовать функции StringLower(), StringUpper() для преобразования определенных строк в нижний регистр или верхний регистр.

Функция StringLower()

Возвращает строку, преобразованную в нижний регистр.

Синтаксис

```
Result = StringLower(string)
```

Параметры

string

Литеральный текст, строковый тег или строковое выражение.

Примеры

`StringLower (“TURBINE”)` возвращает “turbine”.

`StringLower (“The value is 22.2”)` возвращает “the value is 22.2”.

`Mtag = StringMid (Mtag)` преобразует значение строкового тега в нижний регистр.

Функция StringUpper()

Возвращает строку, преобразованную в верхний регистр.

Синтаксис

```
Result = StringUpper(string)
```

Параметры

string

Литеральный текст, строковый тег или строковое выражение.

Примеры

`StringUpper (“abcd”)` возвращает “ABCD”.

`StringUpper (“The value is 22.2”)` возвращает “THE VALUE IS 22.2”.

`Mtag = StringUpper (Mtag)` преобразует значение строкового тега в верхний регистр.

Удаление пробелов из строк

В скрипте, функцией StringTrim() можно и убирать пробелы в начале и конце текстовых строк. Данной функцией можно удалять нежелательный пробелы из строки, например, в введенном пользователем тексте.

Функция StringTrim()

Удаляет пробелы в начале и в конце строки. Можно использовать для удаления нежелательная пробелов из строки, например, в введенном пользователем тексте

Синтаксис

```
Result = StringTrim (string, trimtype)
```

Параметры

string

Литеральный текст, строковый тег или строковое выражение.

trimtype

Литеральное значение, аналоговый тег или числовое выражение, которое определяет какие пробелы удалять.

- 1 = пробелы в начале строки
- 2 = пробелы в конце строки
- 3 = пробелы в начале и в конце строки

Примечание

Удаление пробелов в начале и в конце из строки. Пробелы это символы с ASCII кодами 0x20 и с 0x09 по 0x0D.

Примеры

Для удаления всех пробелов в строковом теге (mtag), используется следующий скрипт:

```
DIM i AS INTEGER;
DIM tmp AS MESSAGE;
mtag = StringTrim(mtag,3); {обрабатывается mtag}
FOR i = 1 TO StringLen(mtag) {переменная цикла i с
                              первого символа до
                              последнего строки тега
                              mtag}
IF StringMid(mtag, i, 1)<>" " THEN {i-ый символ не
                                    пробел }
tmp = tmp + StringMid(mtag, i, 1); {добавить этот
                                    символ в тег tmp}
ENDIF;
NEXT;
mtag = tmp;
```

Другие примеры:

```
StringTrim(“ Joe ”,1) returns “Joe ”.
StringTrim(“ Joe ”,2) returns “ Joe”.
```

Данный скрипт удаляет все пробелы с правого и левого края строки, содержащейся в теге, mtag:

```
Mtag = StringTrim(Mtag,3)
```

Форматирование строк пробелами

В скрипте можно использовать функцию StringSpace() для добавления пробелов в строки.

Синтаксис

```
Result = StringSpace(number)
```

Параметры

number

Литеральное число, числовой тег или числовое выражение.

Примеры

`StringSpace (4)` возвращает строку, состоящую из четырех пробелов

`"Pump" + StringSpace (1) + "Station"` возвращает "Pump Station".

Преобразование символов и ASCII кодов

В скрипте при помощи функции StringChar() и StringASCII() можно преобразовывать символы строки в ASCII коды и обратно ASCII коды в символы.

Данные функции не поддерживают наборы многобайтовых символов. Поддерживаются только символы в диапазоне 0-255.

ASCII коды удобно использовать, если осуществляются некоторых числовых преобразований над строками (например, для шифрования строки).

Функция StringChar()

Возвращает один символ, соответствующий указанному ASCII коду.

Синтаксис

```
Result = StringChar(ASCIIChar)
```

Параметры

ASCIIChar

Литеральное число, числовой тег или числовое выражение в диапазоне от 0 до 255.

Примечание

Эта функция очень удобна при передаче управляющих символов во внешние устройства (принтеры или модемы) или двойные кавычки в SQL запросах.

Примеры

`StringChar (65)` возвращает “A”

Данный скрипт возвращает строку “Hello World” в двойных кавычках:

```
StringChar(34)+”Hello World”+StringChar(34)
```

Данный скрипт возвращает строку “Hello World”, в которой оба слова разделены символом возврата каретки и символом перемещения строки.

```
“Hello”+StringChar(13)+StringChar(10)+”World”
```

Функция StringASCII()

Возвращает ASCII код первого символа строки.

Синтаксис

```
Result = StringASCII(string)
```

Параметры

string

Литеральная строка, строковый тег или строковое выражение.

Примеры

`StringChar (“A”)` возвращает 65.

Поиски и замена текста в строках

Для языков, в которых используются однобайтовые наборы символов (например в английском), можно использовать функции StringInString() и StringReplace() в скрипте для осуществления поиска и замены символов в строковых тегах.

Функция	Использование
StringInString()	Поиск строки в другой строке и возврат номера позиции в качестве результата
StringReplace()	Замена определенных символов или слов другими символами или словами в определенной строке и возврат результата в новую строку.

Функция StringInString()

Возвращает номер позиции первого символа указанной строки в другой строке.

Синтаксис

```
Result = StringInString(string, searchfor, strtpos, casesens)
```

Параметры

string
Строка, в которой осуществляется поиск. Литеральная строка, строковый тег или строковое выражение.

searchfor
Искомая строка. Литеральная строка, строковый тег или строковое выражение.

strtpos
Стартовая позиция поиска. Литеральная строка, строковый тег или строковое выражение.

casesens
Определяет, будет ли поиск производится с учётом регистра. Может быть 0 или 1, дискретным тегом или числовым выражением.

0 – поиск будет производится без учета регистра (верхний регистр и нижний регистр рассматриваются одинаково).

1 – поиск будет производится с учётом регистра (верхний регистр и нижний регистр рассматриваются по-разному).

Примечание

Данную функцию можно использовать для определения, содержится ли определенная строка в строковом теге. Можно указать стартовую позицию поиска и будет ли учитываться при поиске регистр.

Примеры

Скрипт возвращает 5 – потому что первая “M” в “MTX” находится на пятой позиции в строке.

```
StringInString(“DBO MTX-010”,“MTX”,1,0)
```

Скрипт возвращает 3 – потому что первая “M” в “MTX” находится на третьей позиции в строке:

```
StringInString("T-MTX 010 MTX","MTX",1,0)
```

Скрипт возвращает 11 – потому что первая “M” в “MTX” находится после 8-ой позиции на 11-ой позиции в строке:

```
StringInString("T-MTX 010 MTX","MTX",8,0)
```

Скрипт возвращает 11 – потому что первая совпадение “MTX” произойдет на 11-ой позиции в строке:

```
StringInString("t-mtx 030 MTX", "MTX",1,1)
```

Скрипт возвращает 0, потому что в строке нет “Mty” строки:

```
StringInString("t-mtx 030 MTY-Mtx","Mty",1,1)
```

Функция StringReplace()

Ищет строку в строке, при обнаружении, заменяет ее на другую строку. Можно указать:

- Чувствительность к состоянию регистра – определяет, будут ли буквы верхнего и нижнего регистра обрабатываться как одинаковые буквы.
- Количество совпадений для замены – используется, если при поиске обнаружено более одного совпадения строки, будет ли учитываться регистр или нет.
- Совпадение слов целиком – использовать, если искомая строка это целое слово.

Примечание Данная функция не поддерживает работу с набором двухбайтовых символов.

Синтаксис

```
Result = StringReplace (string, searchfor, replacewith,
casesens, numtoreplace, matchwholewords)
```

Параметры

string
Строка, в которой осуществляется поиск. Литеральная строка, строковый тег или строковое выражение.

searchfor
Искомая строка. Литеральная строка, строковый тег или строковое выражение.

replacewith
Строка, на которую будет производиться замена. Литеральная строка, строковый тег или строковое выражение.

casesens
Определяет, будет ли поиск производиться с учётом регистра. Может быть 0 или 1, дискретным тегом или числовым выражением.

0 – поиск будет производиться, без учета регистра (верхний регистр и нижний регистр рассматриваются одинаково).

1 – поиск будет производиться, с учётом регистра (верхний регистр и нижний регистр рассматриваются по-разному).

numtoreplace
Количество производимых замен. Установка параметра в -1, заменяет все обнаруженные совпадения строк. Литеральное целое число, тег integer или целочисленное выражение.

matchwholewords
Определяет, будет ли обнаруживаться только целое слово. Может быть 0 или 1, дискретным тегом или числовым выражением.

0 – функция ищет символы искомой строки везде в строке.

1 – поиск только целых слов.

Примеры

Скрипт заменяет только первое совпадение строки и возвращает "MTY 030 MTX".

```
StringReplace("MTX 030 MTX","MTX","MTY",0,1,0)
```

Скрипт заменяет все совпадения строки и возвращает "MTY 030 MTY".

```
StringReplace("MTX 030 MTX","MTX","MTY",0,-1,0)
```

Скрипт заменяет все совпадения строки, которые соответствуют по регистру, и возвращает "MTY 030 mtx".

```
StringReplace("MTX 030 mtx","MTX","MTY",1,-1,0)
```

Скрипт заменяет все совпадения в строке целых слов и возвращает "MTY 030 QMTX".

```
StringReplace("MTX 030 QMTX","MTX","MTY",0,-1,1)
```

Возвращение информации о строках

В скрипте можно использовать функции StringLen() и StringTest() для возвращения длинны указанной строки и проверки наличия символа в определенной группе символов.

Функция StringLen()

Возвращение длинны указанной строки, включая невидимые символы.

Синтаксис

```
Result = StringLen (string)
```

Параметры

string
Литеральная строка, строковый тег или строковое выражение.

Примеры

`StringLen ("Twelve percent")` возвращает 14

`StringLen ("12%")` возвращает 3

`StringLen ("The end." + StringChar(13))` возвращает 9.

Функция StringTest()

Проверяет, имеется ли первый символ строки в определенной группе символов.

Синтаксис

```
Result = StringTest (string, group)
```

Параметры

string
Литеральная строка, строковый тег или строковое выражение.

group
Номер группы, в которой проверяется символ. Литеральное значение, целочисленный тег, целочисленное выражение в диапазоне от 1 до 11.

1 - Алфавитно-цифровые символы (A-Z, a-z, 0-9)

2 - Символы цифр (0-9)

3 - Символы букв (A-Z, a-z)

4 - Символы букв верхнего регистра (A-Z)

5 - Символы букв нижнего регистра (a-z)

6 - Символы пунктуации (ASCII 0x20 – 0x2F), например !,@,#,$,%,^,&,* и так далее.

7 - ASCII символы (ASCII 0x00 – 0x7F)

8 - Шестнадцатеричные символы (0-9, A-F, a-f)

9 - Печатаемые символы (ASCII 0x20 – 0x7E)

10 - Управляющие символы (ASCII 0x00 – 0x1F и 0x7F)

11 - Разделители (ASCII 0x09 – 0x0D и 0x20)

Примеры

Скрипт возвращает 1 – так как “A” это алфавитно-цифровой символ:

```
StringTest ("ACB123", 1)
```

Скрипт возвращает 0 – так как “A” это не символ нижнего регистра:

```
StringTest ("ACB123", 5)
```

Сравнение строк

В скрипте можно использовать функции StringCompare(), StringCompareNoCase(), StringCompareEncrypted() для сравнения двух строк.

Функция	Использование
StringCompare()	Осуществляет сравнение с учетом регистра.
StringCompareNoCase()	Осуществляет сравнение без учета регистра.
StringCompareEncrypted()	Осуществляет сравнение зашифрованной строки с незашифрованной строкой.

Функция StringCompare()

Сравнивает две строки между собой и возвращает дискретный результат (0 = строки одинаковые). Учитывается регистр каждого символа, таким образом “A” не равно “a”.

Синтаксис

```
Result = StringCompare (string1, string2)
```

Параметры

String1
Литеральная строка, строковый тег или строковое выражение.

String2
Литеральная строка, строковый тег или строковое выражение.

Пример

`StringCompare (“Apple”, “Apple”)` возвращает 0.

`StringCompare (“Apple”, “apple”)` возвращает 1.

Данный скрипт сравнивает два строковых тега и возвращает дискретный результат (0 или 1).

```
StringCompare (mtag1, mtag2)
```

Функция StringCompareNoCase()

Сравнивает две строки между собой и возвращает целочисленный результат (0 = строки одинаковые). Не учитывается регистр каждого символа, таким образом “A” рассматривается равным “a”.

Целочисленные результаты:

- 0, если обе строки идентичны (независимо от регистра)
- Не ноль. Результат – разница ASCII значений между отличающимися символами (независимо от регистра).

Примечание Результат функции StringCompareNoCase() может использоваться как дискретный, все ненулевые значения рассматриваются равным TRUE в скриптах InTouch.

Синтаксис

```
Result = StringCompareNoCase (string1, string2)
```

Параметры

String1
Литеральная строка, строковый тег или строковое выражение.

String2
Литеральная строка, строковый тег или строковое выражение.

Пример

Возвращает 0 так как строки рассматриваются как идентичные.

```
StringCompare (“Apple”, “apple”)
```

Возвращает -6 так как строки рассматриваются как не идентичные и разница ASCII значений первых различающихся символов “p” и “v” равна -6.

```
StringCompare (“Apple”, “Avocado”)
```

Функция StringCompareEncrypted()

Сравнивает зашифрованную строку с незашифрованной строкой и возвращает дискретный результат. Можно использовать данную функцию для верификации пароля.

Синтаксис

```
Result = StringCompareEncrypted (encrypted, plain)
```

Параметры

encrypted
Тег зашифрованной строки.

plain
Литеральная строка, строковый тег или строковое выражение.

Пример

Скрипт возвращает 1, когда открытый текст и зашифрованный текст идентичны, иначе возвращает 0. Passwd - это строковый тег, содержащий значение от введенного пользователем зашифрованного текста. PlainTxt – это строковый текст, с которым производится сравнение.

```
StringCompare (PlainTxt, Passwd)
```

Конвертирование типов данных

В скрипте можно конвертировать значения, содержащиеся в тегах в другие типы данных при помощи функций в скриптах. Это позволит обрабатывать строковыми данными с математическими функциями или записывать значения в ArchestraA® Log Viewer для поиска и устранения неисправностей.

- Функция Text()
- Функция StringFromInt()
- Функция StringFromReal()
- Функция StringToInt()
- Функция StringToReal()
- Функция DText()

Функция Text()

Функция Text() возвращает значение числа как строку в соответствии с указанным форматом. Можно использовать для форматирования значений определенным образом или для комбинирования результата с другими строками для дальнейшей работы.

Синтаксис

```
Result = Text (number, format)
```

Параметры

number

Литеральное числовое значение, аналоговый тег или числовое выражение.

format

Использовать “#”, “0”, “.” и ”,”.

Использовать “#” - для представления числа, ”.” - для представления десятичного знака, “0” – для установки нулевого старшего разряда и ”,” – для вставки запятой.

Если использовать ноль в формате, за ним должны следовать нули. Все позиции справа от десятичной точки, всегда должны быть нулями. Например, 000.00 - правильно, однако #0#0.# - неправильно.

При необходимости функция округляет значение. Литеральная строка, строковый тег или строковое выражение.

Пример

`Text (66, “#.00”)` возвращает “66.00”

`Text (1234, “#”)` возвращает “1234”

`Text (123.4, “#,##0.00”)` возвращает “123.4”

`Text (12.3, “0,000.0”)` возвращает “0,012.3”

`Text (3.57, “#.#”)` возвращает “3.6”

Данный скрипт вернет строку “Reactor Pressure is 1690.3 mbar”, если тег “pressure” содержит 1690.2743.

```
“Reactor Pressure is ” + Text(pressure, “#.#”) + “mbar”
```

Функция StringFromInt()

В скрипте можно конвертировать целочисленное значение в строковое значение при помощи функции StringFromInt().

Данная функция возвращает строковое значение целочисленного значения и одновременно осуществляет конвертирование по основанию. Это может быть использовано, например, для отображения текста вместе целочисленными значениями или для конвертирования целочисленных значений в шестнадцатеричные.

Синтаксис

```
Result = StringFromInt (number, base)
```

Параметры

number

Литеральное числовое значение, аналоговый тег или числовое выражение.

base

База для конвертирования. Используется для конвертирования значений по различным базам: 2 – двоичный вид, 10 – десятичный, 16 – шестнадцатеричный. Литеральное целочисленное значение, целочисленный тег или целочисленное выражение.

Пример

```
StringFromInt (26, 2)
```
возвращает “11010”

```
StringFromInt (26, 8)
```
возвращает “32”

```
StringFromInt (26, 10)
```
возвращает “26”

```
StringFromInt (26, 16)
```
возвращает “1A”

Функция StringFromReal()

В скрипте можно конвертировать вещественное значение в строковое значение при помощи функции StringFromReal().

Можно так же:

- Округлять значение с определенной точностью.
- Представлять значение в экспоненциальном представлении.

Данная функция может быть использована, например, для отображения текста вместе вещественными значениями или для конвертирования целочисленных значений в шестнадцатеричные.

Синтаксис

`Result = StringFromReal (number, precision, type)`

Параметры

number
Литеральное числовое значение, аналоговый тег или числовое выражение.

precision
Определяет количество знаков после десятичной точки, которое будет использоваться. Литеральное целочисленное значение, целочисленный тег или целочисленное выражение.

type
Определяет, будет ли использоваться экспоненциальное представление Литеральная строка, строковый тег или строковое выражение.

“f” – использовать представление с плавающей точкой.

“e” – использовать экспоненциальное представление с “e” нижнего регистра.

“E” – использовать экспоненциальное представление с “E” верхнего регистра.

Пример

`StringFromReal(263.355, 2,”f”)` возвращает “263.36”

`StringFromReal(263.355, 2,”e”)` возвращает “2.63e2”

`StringFromReal(263.55, 3,”E”)` возвращает “2.636E2”

`StringFromReal(0.5723, 2,”E”)` возвращает “5.72E-1”

Функция StringToInt()

В скрипте можно конвертировать строковое значение в целочисленное значение при помощи функции StringToInt().

Можно использовать для считывания значения содержащегося в начале строки в целочисленный тег для дальнейшей математической обработки.

Синтаксис

`Result = StringToInt (`*`string`*`)`

Параметры

string

Литеральная строка, строковый тег или строковое выражение.

Примечание

Функция проверяет первый символ строки. Если это число, то функция пытается считать этот и последующие символы как целое число, пока не встретится нецифровой символ. Функция не учитывает пробелы в начале строки.

Пример

`StringToInt (“ABCD”)` возвращает 0

`StringToInt (“13.4 mbar”)` возвращает 13

`StringToInt (“Pressre is 13.4”)` возвращает 0

```
DIM i AS INTEGER;
DIM tmp AS INTEGER;
FOR  i = 1 TO StringLen(mtag)
     tmp = StringASCII(StringMid(mtag, i, 1)) - 48;
     IF (tmp>=0 AND tmp<10) THEN
          itag = StringToIntg(StringMid(mtag, i, 0));
     EXIT FOR;
ENDIF;
NEXT;
```

Функция StringToReal()

В скрипте можно конвертировать строковое значение в вещественное значение при помощи функции StringToReal().

Можно использовать для считывания значения содержащегося в начале строки в вещественный тег для дальнейшей математической обработки.

> **Примечание** Данная функция поддерживает экспоненциальное представление и преобразовывает строковое выражение 1e+6 в 1000000.

Синтаксис

```
Result = StringToReal (string)
```

Параметры

string
Литеральная строка, строковый тег или строковое выражение.

Примечание

Функция проверяет первый символ строки. Если это число, то функция пытается считать этот и последующие символы как вещественное число, пока не встретится нецифровой символ. Функция не учитывает пробелы в начале строки.

Для извлечения первого вещественного символа из строки (строковый тег mtag), который находится не в начале строки и сохранить его в вещественный тег rtag1, можно использовать следующий скрипт:

```
DIM i AS INTEGER;
DIM tmp AS INTEGER;
FOR i = 1 TO StringLen(mtag)
    tmp = StringASCII(StringMid(mtag, i, 1)) - 48;
    IF (tmp>=0 AND tmp<10) THEN
         rtag = StringToReal(StringMid(mtag, i, 0));
         EXIT FOR;
   ENDIF;
NEXT;
```

Пример

`StringToReal (“ABCD”)` возвращает 0

`StringToReal (“13.4 mbar”)` возвращает 13.4

`StringToReal (“Pressre is 13.4”)` возвращает 0

Функция DText()

В скрипте можно конвертировать дискретное значение в строковое значение при помощи функции DText().

Данная функция возвращает различные строковые значения в зависимости от дискретных значений.

Синтаксис

```
Result = DText(Boolean, stringtrue, stringfalse)
```

Параметры

Boolean

Литеральное дискретное значение, дискретный тег или дискретное выражение.

stringtrue

Строка, которая будет возвращена, если *Boolean* значение true. Литеральная строка, строковый тег или строковое выражение.

stringfalse

Строка, которая будет возвращена, если *Boolean* значение false. Литеральная строка, строковый тег или строковое выражение.

Пример

Скрипт возвращает “Running” если дискретное значение тега *switch* – 1, в противном случае возвращает “Stopped”.

```
DText (switch, “ Running”, “Stopped”)
```

Скрипт возвращает On и Off сообщения другого дискретного тега *switch2*, в зависимости от дискретного значения *switch1*.

```
DText (switch1, switch2.OffMsg, switch2.OnMsg)
```

Работа с окнами InTouch в режиме исполнения

В скрипте можно управлять поведением и появлением окон InTouch. Можно написать скрипт, используя редактор, который будет распечатывать отдельные окна или все что находится на экране.

Отображение списка открытых окон

В скрипте можно отобразить диалоговое окно, содержащее список окон проекта InTouch HMI, которые открыты в данный момент, при помощи функции OpenWindowList().

Использовать данную функцию в анимационной связи нельзя.

Функция OpenWindowList()

Отображает диалоговое окно со списком окон InTouch, которые открыты в данный момент.

Можно использовать данную функцию в анимационной связи.

Синтаксис

```
[result = ]OpenWindowList()
```

Пример

Данный скрипт открывает диалоговое окно Open Window List и отображает все открытые в данный момент окна InTouch.

```
OpenWindowList()
```

Проверка открыто/закрыто/существует окно

В скрипте можно проверить состояние окна InTouch – открыто окно, закрыто окно и существует ли окно, при помощи функции WindowState().

Функция WindowState()

Отображает диалоговое окно со списком окон InTouch, которые открыты в данный момент.

Можно использовать данную функцию в анимационной связи.

Синтаксис

```
result = WindowState(windowname)
```

Параметры

Windowname

Имя окна. Литеральная строка, строковый тег или строковое выражение.

Возвращаемое значение

Целочисленное значение со следующим смыслом:

0 – Окно InTouch существует и в настоящий момент закрыто.

1 – Окно InTouch существует и в настоящий момент открыто.

2 – Окно InTouch не существует.

Пример

Скрипт вернет 0, если окно InTouch *Main* существует, но не открыто.

```
WindowState(“Main”)
```

Открытие окон InTouch

В скрипте можно открывать окно InTouch при помощи одной из следующих встроенных функций:

Функция	Использование
Show	Открыть окно InTouch в координатах определенных в установках положения окна.
ShowAt()	Открыть окно InTouch в координатах определенных специально. Открытое окно выравнивается на указанной позиции. Эта функция также может быть использована для перемещения открытого окна.
ShowHome	Открыть окно(а) InTouch, которые указаны в разделе Home Windows, в разделе WindowViewer Properties и закроет любые другие окна.
ShowAtTopLeft()	Открыть окно InTouch в координатах определенных специально. Левый верхний угол открытого окна выравнивается по указанной позиции. Эта функция также может быть использована для перемещения открытого окна.

Функция Show()

Открывает окно InTouch в координатах определенных в установках положения окна.

Синтаксис

`Show windowname`

Параметры

Windowname

Имя окна, которое будет открываться. Литеральная строка, строковый тег или строковое выражение.

Пример

Скрипт откроет окно *Main*..

```
Show “Main”;
```

Скрипт откроет окно имя которого хранится в строковом теге wname.

```
Show wname;
```

Функция ShowAt()

Открывает окно InTouch в координатах определенных специально. Эта функция также может быть использована для перемещения открытого окна. Открытое окно выравнивается на указанной позиции.

> **Примечание** Окно не будет выровнено, если один из краев окна выходит за пределы экрана.

Синтаксис

```
ShowAt (windowname, xpos, ypos)
```

Параметры

Windowname
Имя окна, которое будет открыто или перемещено.

xpos
Положение в пикселях по горизонтали, на которое будет перемещен центр окна. Литеральное значение, аналоговый тег или числовое выражение.

ypos
Положение в пикселях по вертикали, на которое будет перемещен центр окна. Литеральное значение, аналоговый тег или числовое выражением

Пример

Скрипт откроет окно *Main*, таким образом чтобы оно было выровнено по позиции x: 450, y:130

```
ShowAt (“Main”,450,130);
```

Скрипт откроет окно “*UserDialog*” и выровняет его, так чтобы его центр находился над центральной позицией объекта, который вызывается данной функцией (например кнопкой).

```
ShowAt(“UserDialog”,$ObjHor,$ObjVer);
```

Функция ShowHome

Открывает окно(а), которые указаны в разделе Home Windows, в разделе WindowViewer Properties и закроет любые другие окна.

Синтаксис

```
ShowHome
```

Функция ShowTopLeftAt()

Открывает окно InTouch в координатах определенных специально. Эта функция также может быть использована для перемещения открытого окна.

Синтаксис

```
ShowTopLeftAt (windowname, xpos, ypos)
```

Параметры

Windowname
Имя окна, которое будет открыто или перемещено.

xpos
Положение в пикселях по горизонтали, левого верхнего угла окна. Литеральное значение, аналоговый тег или числовое выражение.

ypos
Положение в пикселях по вертикали, левого верхнего угла окна. Литеральное значение, аналоговый тег или числовое выражением

Пример

Скрипт откроет окно *Main*, таким образом, чтобы его левый верхний угол находился на позиции x: 450, y:130

```
ShowTopLeftAt (“Main”,450,130);
```

Перемещение и изменение размера окон

В скрипте можно перемещать и изменять размер окон при помощи функции WWMoveWindow(). Новое местоположение и новый размер окна применяются временно, пока открыто указанное окно.

Функция WWMoveWindow()

Перемещает или изменяет размер открытого окна InTouch на указанную позицию и до указанных размеров. Новое местоположение и новый размер окна применяются временно, пока открыто указанное окно.

Синтаксис

```
WWMoveWindow (windowname, xpos, ypos, xsize, ysize)
```

Параметры

Windowname
Имя окна, которое будет открыто или перемещено.

xpos
Положение в пикселях по горизонтали, левого верхнего угла окна. Литеральное значение, аналоговый тег или числовое выражение.

ypos
Положение в пикселях по вертикали, левого верхнего угла окна. Литеральное значение, аналоговый тег или числовое выражением

xsize
Размер в пикселях по горизонтали указанного окна. Литеральное значение, аналоговый тег или числовое выражение.

ysize
Размер в пикселях по вертикали указанного окна. Литеральное значение, аналоговый тег или числовое выражение.

Скрытие окон InTouch

В скрипте можно скрывать окна InTouch при помощи одной из следующих встроенных функций:

Функция	Использование
Hide	Скрыть определенное окно.
HideSelf	Скрыть активное в настоящий момент окно.

Функция Hide()

Скрывает (закрывает) InTouch окно.

Синтаксис

```
Hide windowname
```

Параметры

Windowname
Имя окна, которое будет скрыто. Литеральная строка, строковый тег, строковое выражение.

Пример

Скрипт скроет окно с именем *UserConfirmation*.

```
Hide “UserConfirmation”;
```

Функция HideSelf ()

Скрывает (закрывает) активное в настоящий момент окно InTouch.

Синтаксис

```
HideSelf
```

Параметры

Windowname
Имя окна, которое будет скрыто. Литеральная строка, строковый тег, строковое выражение.

Пример

```
HideSelf;
```

Изменение цвета окна

В скрипте можно изменить цвет открытого окна InTouch при помощи функции ChangeWindowColor().

Функция ChangeWindowColor()

Изменяет цвет открытого окна InTouch и возвращает результирующий код.

Синтаксис

```
Result = ChangeWindowColor (windowname, rValue, gValue, bValue)
```

Параметры

Windowname
Имя окна, цвет которого будет изменен. Литеральная строка, строковый тег, строковое выражение.

rValue
Интенсивность красного цвета. Литеральное целочисленное значение, целочисленный тег, целочисленное выражение в диапазоне от 0 до 255.

gValue
Интенсивность зеленого цвета. Литеральное целочисленное значение, целочисленный тег, целочисленное выражение в диапазоне от 0 до 255.

bValue
Интенсивность синего цвета. Литеральное целочисленное значение, целочисленный тег, целочисленное выражение в диапазоне от 0 до 255.

Возвращаемое значение

0 – Операция завершена неудачно, окно не определено или RGB значение вне диапазона.

1 – Операция завершена успешно.

2 – Операция завершена неудачно. Окно существует но не открыто.

Печать окон в режиме исполнения

В скрипте можно распечатывать отдельные окна InTouch или весь экран WindowViewer при помощи функции PrintWindow() или PrintScreen(). Можно также установить принтер для использования при помощи SetWindowPrinter().

Функция SetWindowPrinter()

В режиме исполнения, можно установить принтер для использования при помощи SetWindowPrinter().

> **Примечание** Принтер, устанавливаемый данной функцией, будет принтером, который будет использоваться функцией PrintHT().

Синтаксис

```
SetWindowPrinter (printername)
```

Параметры

printername
Имя принтера возникает или как общий ресурс сети или как имя принтера возникающее в окне свойств принтера. Литеральная строка, строковый тег, строковое выражение.

Пример

В данном примере PRTSRV1 это имя узла сети, а PRT22SW1 это сетевое имя принтера

```
SetWindowPrinter (“\\PRTSRV1\ PRT22SW1”);
```

В данном примере Epson LX-300 это имя принтера, возникающее в окне свойств принтера.

```
SetWindowPrinter (“Epson LX-300”);
```

В данном примере MyPrinter это тег, который содержит имя установленного принтера или путь к общему ресурсу.

```
SetWindowPrinter (MyPrinter);
```

Рекомендации для печати

Ниже приведен список того, что надо учитывать при печати (применимо как для печати одного окна, так и всего экрана WindowViewer).

1. Открыть окно, которое будет распечатываться, перед тем как начать печать. Иначе, ActiveX объекты могут распечататься не корректно.
2. Нельзя распечатывать на тот же принтер, который в данный момент производить распечатку алармов.
3. Необходимо избегать перекрывания окон и объектов в окнах, при печати.
4. Использовать True Type шрифты, там где это возможно. Стандартный шрифт InTouch – это System шрифт, не является True Type шрифтом.
5. Для более быстрой печати, необходимо использовать белый фон, меньшее количество объектов, и текст вместо графики.
6. WindowViewer ждет определенное количество времени перед тем как послать окно в очередь на принтер. В течение этого времени WindowViewer обновляет любых DDE значений для данного окна в фоновом режиме. Для изменения этого времени ожидания, необходимо открыть файл Intouch.ini и изменить (или добавить) следующую строчку (в миллисекундах):
   `PrintWindowWait=10000`

Функция PrintWindow()

В скрипте, можно распечатывать окно InTouch при помощи PrintWindow().

Синтаксис

```
[result = ] PrintWindow (windowname, leftmargin,
topmargin, width, height, options)
```

Параметры

windowname
Имя окна, которое будет распечатано. Литеральная строка, строковый тег, строковое выражение.

leftmargin
Смещение левого края (в дюймах). Литеральное числовое значение, аналоговый тег, числовое выражение.

topmargin
Смещение верхнего края (в дюймах). Литеральное числовое значение, аналоговый тег, числовое выражение.

width
Ширина, выводимая на принтер (в дюймах). Установить значение в 0, для наибольшей разрешающей способности. Литеральное числовое значение, аналоговый тег, числовое выражение.

height
Высота, выводимая на принтер (в дюймах). Установить значение в 0, для наибольшей разрешающей способности. Литеральное числовое значение, аналоговый тег, числовое выражение.

options

Дискретное значение 0 или 1, используется только если *height* и *width* равны 0. Литеральное дискретное значение, дискретный тег, дискретное выражение.

1 – Окно распечатывается с наибольшей разрешающей способностью, которое является многократным увеличением размера окна.

0 – Окно распечатывается с наибольшей разрешающей способностью, умещающейся на странице.

Примечание Если окно содержит изображение, установка параметра *options* в 1, предотвратит растягивание рисунка.

Возвращаемые значения

0 – Задача печати не выполнена, или окно не существует.

1 – Задача печати успешно завершена.

Функция PrintScreen()

В скрипте, можно распечатывать весь экран WindowViewer при помощи функции PrintScreen().

Синтаксис

```
PrintScreen (ScreenOption, PrintOption)
```

Параметры

ScreenOption

Определяется какое что экрана WindowViewer будет распечатываться. Литеральное целочисленное значение, целочисленный тег, целочисленное выражение.

1 – Печать рабочей области, без меню (по умолчанию).
2 – Печать всей области окна, включая меню.

PrintOption

Определяется как распечатываемое изображение будет растянуто для размещения при выводе на принтер.

- 1 – Максимальное соответствие:
  Изображение растягивается таким образом, чтобы и по горизонтали и по вертикали соответствовать размещению при выводе на принтер без изменения коэффициента отношения (по умолчанию).
- 2 – Соответствие по вертикали:
  Изображение растягивается таким образом, чтобы по вертикали соответствовать размещению при выводе на принтер без изменения коэффициента отношения. При этом изображение может быть обрезано по горизонтали.
- 3 – Соответствие по горизонтали:
  Изображение растягивается таким образом, чтобы по вертикали соответствовать размещению при выводе на принтер без изменения коэффициента отношения. При этом изображение может быть обрезано по горизонтали.
- 4 – Растянуть по странице:
  Изображение растягивается таким образом, чтобы по горизонтали соответствовать размещению при выводе на принтер. Коэффициента отношения может изменяться, но изображение при этом не обрезается.
- Неправильная опция, включая 0, по умолчанию принимает значение Максимальное соответствие.

  Примечание Всплывающие окна, которые возникают за пределами экрана WindowViewer, обрезаются.

Пример

Данный скрипт отсылает на печать текущий экран WindowViewer, без меню. Область печати содержит, экран растянутый так чтобы полностью заполнять область печати.

```
PrintScreen (1,4);
```

Функция PrintHT()

Можно создать скрипт (например, на кнопку), который будет распечатывать исторический тренд.

Если необходимо распечатать все окно полностью вместо только тренда, используйте PrintWindow() вместо PrintHT().

> **Примечание** Печать исторического тренда при помощи опции Print или функции PrintHT() не распечатают значения X и Y. Используйте функции PrintWindow() и PrintScreen() для печати значений X и Y.

Синтаксис

```
PrintHT(HistTrendTagname);
```

Параметры

HistTrendTagname
Тег исторического тренда, который будет распечатываться.

Обработка информации о дате и времени

В скрипте можно использовать системные теги и встроенные функции для использования системного установок системного времени и даты для вычислений. Скрипты InTouch также поддерживают многочисленные временные пояса и перевод на летнее время.

Получение информации о дате и времени в численном виде

В скрипте можно использовать различные числовые системные теги и одну встроенную функцию для получения информации о системном времени и дате. Данные теги и функция могут быть использованы для других математических вычислений.

Тег/Функция	Использование
$Year	Возвращает текущий год
$Month	Возвращает текущий месяц года
$Day	Возвращает текущий день месяца
$Hour	Возвращает текущий час дня
$Minute	Возвращает текущую минуту часа
$Second	Возвращает текущую секунду минуты
$Msec	Возвращает текущую миллисекунду
$Time	Возвращает текущее время в миллисекундах, прошедшее с полуночи в настоящем временном поясе.
$Date	Возвращает количество целых дней прошедшее с 1 января 1970 в настоящем временном поясе.
$DateTime	Возвращает количество дней (включая часть дня) прошедшее с 1 января 1970 в настоящем временном поясе.
DateTimeGMT()	Возвращает количество дней (включая часть дня) прошедшее с 1 января 1970 в универсальном глобальном времени (UTC).

Системный тег $Year

Возвращает текущий год.

Синтаксис

```
$Year
```

Тип данных

Целочисленный (только чтение).

Пример

В данном примере, скрипт присваивает тегу Welcome строку “Welcome to xxxx”, где xxxx это текущий год.

```
Welcome = “Welcome to “ + StringFromIntg($Year,10);
```

Системный тег $Month

Возвращает текущий месяц.

Синтаксис

```
$Month
```

Тип данных

Целочисленный (только чтение).

Пример

В данном примере, скрипт присваивает тегу MonthName строку “October”, если текущий месяц 10.

```
IF $Month==10 THEN
MonthName=”October”;
ENDIF;
```

Системный тег $Day

Возвращает текущий день месяца

Синтаксис

```
$Day
```

Тип данных

Целочисленный (только чтение).

Пример

В данном примере, скрипт присваивает тегу Msg2User строку “It is a leap year!”, если текущая дата 29 Февраля..

```
IF $Day==29 AND $Month==2 THEN
Msg2Usr=”It is a leap year!”;
ENDIF;
```

Системный тег $Hour

Возвращает текущий час дня.

Синтаксис

```
$Hour
```

Тип данных

Целочисленный (только чтение).

Пример

В данном примере, скрипт проверяет текущее время, если время 20:00 и функция BackUp не запущена (выражается дискретным тегом BackupAlreadyRun), то запускается функция BackUp и дискретный тег BackupAlreadyRun устанавливается в 1.

```
IF $Hour==20 AND BackupAlreadyRun==0 THEN
CALL RunBackup();
BackupAlreadyRun=1;
ENDIF;
```

Системный тег $Minute

Возвращает текущую минуту часа.

Синтаксис

```
$Minute
```

Тип данных

Целочисленный (только чтение).

Пример

В данном примере, скрипт проверяет текущее время, если время 16:50, то появляется окно ShiftEnd.

```
IF $Minute==50 AND $Hour==16 THEN
Show “Shift End”;
ENDIF;
```

Системный тег $Second

Возвращает текущую секунду минуты

Синтаксис

```
$Second
```

Тип данных

Целочисленный (только чтение).

Пример

В данном примере, скрипт генерирует синусоиду с амплитудой 100 и периодом 1 минута.

```
100*Sin(6*$Second)
```

В данном примере, скрипт генерирует 1 и 0 каждую секунду.

```
$Second.00
```

Системный тег $Msec

Возвращает текущую миллисекунду

> **Примечание** По умолчанию InTouch обновляет все теги каждые 1000 миллисекунд. Поэтому системный тег $Msec с виду не изменяется . Если увеличить скорость обновления в свойствах WindowViewer, можно увидеть обновление тега $Msec.

Синтаксис

```
$Msec
```

Тип данных

Целочисленный (только чтение).

Системный тег $Time

Возвращает текущее время в миллисекундах, прошедшее с полуночи в настоящем временном поясе.

Синтаксис

```
$Time
```

Тип данных

Целочисленный (только чтение).

Пример

В данном примере, скрипт возвращает количество секунд, прошедшее с полуночи.

```
$Time/1000
```

Системный тег $Date

Возвращает количество целых дней прошедшее с 1 января 1970 в настоящем временном поясе.

Синтаксис

```
$Date
```

Тип данных

Целочисленный (только чтение).

Пример

В данном примере, скрипт возвращает текущее время.

```
StringFromTime(($Date*86400)+($Time/1000),3);
```

Системный тег $DateTime

Возвращает количество дней (включая часть дня) прошедшее с 1 января 1970 в настоящем временном поясе.

Синтаксис

```
$DateTime
```

Тип данных

Целочисленный (только чтение).

Пример

В данном примере, скрипт возвращает текущее время.

```
StringFromTime($DateTime*86400,3);
```

Функция DateTimeGMT()

Возвращает количество дней (включая часть дня) прошедшее с 1 января 1970 в универсальном глобальном времени (UTC).

> **Примечание** Функция не может быть использована в анимационных связях отображения.

Синтаксис

```
Result = DateTimeGMT()
```

Возвращаемое значение

Количество дней, прошедших с 1 января 1970 в в универсальном глобальном времени (UTC). Литеральное вещественное значение.

Тип данных

Целочисленный (только чтение).

Пример

В данном примере, скрипт возвращает текущую дату и время в (UTC).

```
StringFromTime(DateTimeGMT() * 86400.0, 3);
```

Получение информации о дате и времени в численном виде

В скрипте можно получать информацию о дате и времени в строковом виде. Это полезно использовать, когда необходимо отобразить в окне информацию дате и времени или требуются вычисления над полной строкой дата/время.

Тег/Функция	Использование
$DateString	Возвращает системную дату в коротком формате.
$TimeString	Возвращает системное время
UTCDateTime	Возвращает UTC время и/или дату и временной пояс локальной машины.

Системный тег $DateString

Возвращает системную дату в коротком формате, так как определено в региональных настройках операционной системы.

Синтаксис

```
$DateString
```

Тип данных

Строка (только чтение).

Пример

В данном примере, скрипт может вернуть 4/28/2006, в зависимости от настроек короткого формата даты в региональных настройках операционной системы.

```
$DateString
```

Системный тег $TimeString

Возвращает системную время, так как определено в региональных настройках операционной системы.

Синтаксис

```
$TimeString
```

Тип данных

Строка (только чтение).

Пример

В данном примере, скрипт может вернуть 02:40:37 PM, в зависимости от настроек короткого формата даты в региональных настройках операционной системы.

```
$TimeString
```

Функция UTCDateTime()

Возвращает UTC время и/или дату и временной пояс локальной машины.

Синтаксис

```
Result = UTCDateTime(format)
```

Параметры

format

Определяет, какое содержание будет возвращено. Литеральное строковое значение, строковый тег, строковое выражение.
UTC_SHORT - функция возвращает UTC время.
UTC_LONG - функция возвращает UTC время и дату.

UTC_LOCAL - функция возвращает имя временного пояса так, как определено в региональных настройках операционной системы.

Любые другие значения возвращают UTC дату и время в формате по умолчанию (день недели месяц число чч:мин:сек год)

Пример

В 09:24 AM Понедельник Январь 6 2003 по Тихоокеанскому временному поясу, функция UTCDateTime вернет следующее:

`UTCDateTime("UTC_SHORT")` вернет 17:24:05

`UTCDateTime("UTC_LONG")` вернет 01/06/2003 17:24:05

`UTCDateTime("UTC_LOCAL")` вернет тихоокеанское стандартное время -8:0:1

`UTCDateTime("Invalid")` вернет Mon Jan 06 17:24:05 2003.

Преобразование информации о дате и времени в строковый вид

В скрипте можно преобразовать информацию о дате и времени в строковый вид, для более удобного интерпретирования и отображения.

Можно использовать следующие функции:

Тег/Функция	Использование
StringFromTime()	Конвертирование метки времени UTC в локальное время и возвращение в виде строки.
wwStringFromTime()	Конвертирование локальной метки времени в UTC и возвращение в виде строки.
StringFromTimeLocal()	Конвертирование метки времени в строку.

Функция StringFromTime()

Конвертирует метку времени, приведенную в UTC, в локальное время и возвращает результат в виде строки. Данная функция принимает во внимание.

Примечание Функция эквивалентна функции StringFromGMTTimeToLocal().

Синтаксис

```
Result = StringFromTime (timestamp, format)
```

Параметры

timestamp

Количество секунд, прошедшее с полуночи 1 января 1970 в UTC временной зоне. Литеральное целочисленное значение, целочисленный тег или целочисленное выражение.

format

Определяет, как будет отображаться строковый результат. Литеральное целочисленное значение, целочисленный тег или целочисленное выражение в диапазоне от 1 до5.

1 - Отображает дату в соответствии с форматом установленном в региональных настройках локальной операционной системы.

2 - Отображает время в соответствии с форматом установленном в региональных настройках локальной операционной системы.

3 - Отображает время и дату в качестве 24 символьной строки (день недели, месяц число чч:мм:сс год)

4 - Отображает день недели в короткой форме

5 - Отображает день недели в длинной форме

Пример

В данном примере предполагается, что временная зона на локальной машине - Тихоокеанское Стандартное время (PST, UTC-08:00). UTC время передаваемое в функцию – 12:0000 AM Friday, January 2, 1970. Так как PST это UTC-08:00, то результат будет следующий:

возвращает “1/1/70”

```
StringFromTime(86400,1)
```

возвращает “04:00:00 PM”

```
StringFromTime(86400,2)
```

возвращает “Thu Jan 01 16:00:00 1970”

```
StringFromTime(86400,3)
```

возвращает “Thu”

```
StringFromTime(86400,4)
```

возвращает “Thursday”

```
StringFromTime(86400,5)
```

Функция wwStringFromTime()

Конвертирует метку времени, приведенную в локальном времени, в UTC, и возвращает результат в виде строки. Данная функция принимает во внимание.

Синтаксис

```
Result = wwStringFromTime (timestamp, format)
```

Параметры

timestamp

Количество секунд, прошедшее с полуночи 1 января 1970 в локальной временной зоне. Литеральное целочисленное значение, целочисленный тег или целочисленное выражение.

format

Определяет, как будет отображаться строковый результат. Литеральное целочисленное значение, целочисленный тег или целочисленное выражение в диапазоне от 1 до5.

1 - Отображает дату в соответствии с форматом установленном в региональных настройках локальной операционной системы.

2 - Отображает время в соответствии с форматом установленном в региональных настройках локальной операционной системы.

3 - Отображает время и дату в качестве 24 символьной строки (день недели, месяц число чч:мм:сс год)

4 - Отображает день недели в короткой форме

5 - Отображает день недели в длинной форме

Пример

В данном примере предполагается, что временная зона на локальной машине - Тихоокеанское Стандартное время (PST, UTC-08:00). Локальное время передаваемое в функцию – 04:0000 AM Friday, January 1, 1970. Так как PST это UTC-08:00, то результат будет следующий:

возвращает “1/2/70”

```
wwStringFromTime(57600,1)
```

возвращает “12:00:00 AM”

```
wwStringFromTime(57600,2)
```

возвращает “Fri Jan 02 00:00:00 1970”

```
wwStringFromTime(57600,3)
```

возвращает “ Fri ”

```
wwStringFromTime(57600,4)
```

возвращает “ Friday”

```
wwStringFromTime(57600,5)
```

Функция StringFromTimeLocal()

Конвертирует метки времени во время, и возвращает результат в виде строки.

Синтаксис

```
Result = StringFromTimeLocal (timestamp, format)
```

Параметры

timestamp

Количество секунд, прошедшее с полуночи 1 января 1970 в локальной временной зоне. Литеральное целочисленное значение, целочисленный тег или целочисленное выражение.

format

Определяет, как будет отображаться строковый результат. Литеральное целочисленное значение, целочисленный тег или целочисленное выражение в диапазоне от 1 до5.

1 - Отображает дату в соответствии с форматом установленном в региональных настройках локальной операционной системы.

2 - Отображает время в соответствии с форматом установленном в региональных настройках локальной операционной системы.

3 - Отображает время и дату в качестве 24 символьной строки (день недели, месяц число чч:мм:сс год)

4 - Отображает день недели в короткой форме

5 - Отображает день недели в длинной форме

Пример

В данном примере предполагается, что временная зона на локальной машине - Тихоокеанское Стандартное время (PST, UTC-08:00). Локальное время, передаваемое в функцию – 04:0000 AM Friday, January 1, 1970. Так как PST это UTC-08:00, то результат будет следующий:

возвращает “1/2/70”

```
StringFromTimeLocal (86400,1)
```

возвращает “12:00:00 AM”

```
StringFromTimeLocal (86400,2)
```

возвращает “Fri Jan 02 00:00:00 1970”

```
StringFromTimeLocal (86400,3)
```

возвращает “ Fri ”

```
StringFromTimeLocal (86400,4)
```

возвращает “ Friday”

```
StringFromTimeLocal (86400,5)
```

Проверка статуса перевода на летнее время

Настройка перевода на летнее время определяет, будет ли операционная система подстраивать время при переходе на летнее время. В скрипте можно проверять установку операционной системы по переходу на летнее время, при помощи функции wwIsDaylightSaving().

Данная функция возвращает статус установки Daylight Saving Time (переход на летнее время) в операционной системе, которая автоматически подстраивает системное время при переводе часов.

Функция wwIsDaylightSaving()

Возвращает статус установки Daylight Saving Time (переход на летнее время) в операционной системе, которая автоматически подстраивает системное время.

Синтаксис

```
Result = wwIsDaylightSaving()
```

Возвращаемое значение

Дискретное значение.

0 – Опция перевода на летнее время не выбрана

1 – Опция перевода на летнее время выбрана.

Взаимодействие с другими приложениями

В скрипте можно осуществлять взаимодействие с другими приложениями Windows при помощи встроенных функций. Например, можно:

- Запускать приложение, такое как Notepad (Блокнот)
- Проверять имя заголовка приложения
- Активировать запущенное приложение
- Эмулировать нажатие кнопок
- Закрывать, сворачивать и разворачивать окно приложения.
- Выполнять команды и обмениваться данными с приложениями, поддерживающими DDE.

Запуск Windows приложения

В скрипте можно запускать приложение, при помощи функции StartApp.

Синтаксис

`StartApp` *`appname`*

Параметры

Appname

Путь и имя файла приложения, которое необходимо запустить. Литеральное строковое значение, строковый тег, или строковое выражение.

> **Примечание** Необходимо знать путь и имя файла приложения. Если приложение находится в каталоге, являющемся частью переменных окружения Windows (Windows PATH environment variable), можно указывать только имя файла, без указания пути.

Пример

Скрипт запускает Microsoft Calculator (Калькулятор).

```
StartApp “calc”
```

Получение заголовка работающего приложения

В скрипте можно находить заголовок или идентификатор списка задачи Windows определенного работающего приложения при помощи функции InfoAppTitle(). Данная информация, например, требуется для проверки запущено ли данный момент приложение или активации приложения.

Функция InfoAppTitle()

Возвращает заголовок или идентификатор списка задачи Windows определенного работающего приложения.

Синтаксис

```
result = InfoAppTitle (appname)
```

Параметры

Appname

Имя файла приложения, которое необходимо запустить без .exe расширения. Литеральное строковое значение, строковый тег, или строковое выражение.

Пример

Скрипт возвращает “Calculator”.

```
InfoAppTitle (“calc”)
```

Скрипт возвращает “Microsoft Excel”.

```
InfoAppTitle (“excel”)
```

Проверка статуса работы приложения

В скрипте можно осуществлять проверку работает ли в настоящий момент приложение при помощи функции InfoAppActive(). Для этого необходимо знать заголовок приложения или идентификатор списка задач Windows.

Функция InfoAppActive()

Возвращает статус работающего приложения.

Синтаксис

```
result = InfoAppActive (apptitle)
```

Параметры

apptitle

Имя приложения или идентификатор списка задачи Windows, для которого запрашивается статус. Литеральное строковое значение, строковый тег, или строковое выражение.

Возвращаемое значение

Дискретное значение.

0 – Приложение не работает.

1 – Приложение работает.

Пример

Скрипт запрашивает приложение Notepad, если оно уже работает, то скрипт активирует его. Иначе, запускает Notepad. Таким образом, производится избежание запуска нескольких экземпляров Notepad.

```
IF InfoAppActive(InfoAppTitle("Notepad"))==1
THEN
      ActivateApp InfoAppTitle( "Notepad" );
ELSE
StartApp "Notepad";
ENDIF;
```

Активация работающего приложения Windows

В скрипте можно активировать Windows приложение при помощи функции ActivateApp(). Данная функция вызывает определенное приложение из фонового режима и активирует его.

Перед активацией работающего Windows приложения, необходимо сделать следующее:

- Необходимо найти имя приложения или идентификатор списка задач Windows. Смотрите раздел Получение заголовка работающего приложения.
- Убедится в том, что Windows приложение уже не запущено. Смотрите раздел Проверка работы приложения.

Функция ActivateApp

Активация уже работающего приложения.

Синтаксис

`ActivateApp` *`apptitle`*

Параметры

`apptitle`

Имя приложения или идентификатор списка задачи Windows, которое необходимо активировать.

Пример

Скрипт проверяет, открыто ли окно с командной строкой, и если открыто, то активирует его. Иначе, запускает окно с командной строкой.

```
IF    InfoAppActive( InfoAppTitle("cmd")) == 1 THEN
      ActivateApp InfoAppTitle("cmd");
ELSE
      StartApp "cmd";
ENDIF;
```

Отсылка эмулированного нажатия кнопок приложению

В скрипте можно эмулировать нажатие кнопок на клавиатуре. Можно, например, сделать следующее:

- Автоматический ввод данных в открытое приложение.
- Управление любым приложением (включая InTouch HMI)

Функция SendKeys

Синтаксис

SendKeys *sequence*

Параметры

sequence

Последовательность кнопок, которая будет эмулироваться. Литеральное строковое значение, строковый тег, или строковое выражение.

В дополнение к обычным символам на клавиатуре (буквенно-цифровые символы), можно также задавать управляющие кнопки как код.

{BACKSPACE} – эмулирует кнопку Backspace
{BREAK} - эмулирует кнопку Break
{CAPSLOCK} - эмулирует кнопку Caps Lock
{DELETE} - эмулирует кнопку Delete (или {DEL})
{DOWN} - эмулирует кнопку Arrow Down
{END} - эмулирует кнопку End
{ENTER} - эмулирует кнопку Enter (или ~)
{ESCAPE} - эмулирует кнопку ESC (или {ESC})
{F1} .. {F12} - эмулирует кнопки F1 .. F12
{HOME} - эмулирует кнопку Home
{INSERT} - эмулирует кнопку Insert
{LEFT} - эмулирует кнопку Arrow Left
{NUMLOCK} - эмулирует кнопку Num Lock
{PGDN} - эмулирует кнопку Page Down
{PGUP} - эмулирует кнопку Page Up
{PRTSC} - эмулирует кнопку Print Screen
{RIGHT} - эмулирует кнопку Arrow Right
{TAB} - эмулирует кнопку Tab
{UP} - эмулирует кнопку Up
+ - эмулирует кнопку Shift
Использовать со скобками, если необходимо использовать вместе с кнопкой Shift.
^ - эмулирует кнопку Ctrl
Использовать со скобками, если необходимо использовать вместе с кнопкой Ctrl.
% - эмулирует кнопку Alt
Использовать со скобками, если необходимо использовать вместе с кнопкой Alt.

Примечания

Перед тем как посылать приложению эмуляцию нажатия комбинации кнопок, необходимо сначала активировать его командами StartApp и/или ActivateApp().

Пример

Скрипт эмулирует нажатие кнопки B.

```
SendKeys “B”;
```

Скрипт эмулирует нажатие комбинации кнопок Ctrl и B, который может быть использован для вызова диалогового окна Печать в другом приложении.

```
SendKeys “^(p)”;
```

Скрипт эмулирует нажатие F1 (который открывает Справку), нажатие кнопки TAB, для перехода в поле поиска, ввод HAL, и нажатие ENTER инициирующего поиск.

```
SendKeys “{F1}{TAB}HAL{ENTER}”;
```

Скрипт эмулирует нажатие Ctrl, Shift и кнопки 1, это сочетание кнопок для перехода WindowViewer.

```
SendKeys “^(+(1))”;
```

Закрытие, сворачивание и разворачивание окон Windows приложения

В скрипте можно закрывать, скрывать или разворачивать другое Windows приложение при помощи команды WWControl().

Перед тем как закрывать, скрывать или разворачивать Windows приложение необходимо сделать следующее:

- Необходимо найти имя приложения или идентификатор списка задач Windows. Смотрите раздел Получение заголовка работающего приложения.
- Убедится в том, что Windows приложение уже не запущено. Смотрите раздел Проверка работы приложения.

Функция WWControl()

Закрывать, скрывать или разворачивать другое Windows приложение.

Синтаксис

```
WWControl (apptitle, control)
```

Параметры

apptitle

Имя приложения или идентификатор списка задачи Windows, которое необходимо активировать. Литеральное строковое значение, строковый тег, или строковое выражение.

control

Определяет действие, которое необходимо предпринять над Windows приложением. Литеральное строковое значение, строковый тег, или строковое выражение.

Restore – активирует и показывает окно приложения.

Minimize – активирует и скрывает окно приложения.

Maximize – активирует и разворачивает окно приложения.

Close – закрывает приложение.

Примечания

Для использования данной функции в Windows Server 2003, необходимо чтобы пользователь являлся членом группы Administrators, группы Performance Log или группы Performance Monitor Users на локальной машине или пользователю должны быть делегированы соответствующие полномочия для записи в реестр.

Пример

Скрипт проверяет восстанавливает окно приложения Calculator, если оно уже запущено.

```
WWControl (“Calculator”,”Restore”);
```

Скрипт закрывает WindowViewer

```
WWControl (InfoAppTitle(“View”),”Close”);
```

Выполнение команд и обмен данными с приложениями через DDE

Можно использовать скрипт для взаимодействия с приложениями, поддерживающими DDE.

Функция	Использование
WWExecute()	Посылка выполнение команды.
WWRequest()	Чтение данных из DDE элементов.
WWPoke()	Запись данных в DDE элементы.

Функция WWExecute()

Отсылает команду приложению, выполняете и возвращает результат статуса.

Синтаксис

```
Result = WWExecute (appname, topic, command)
```

Параметры

appname
Имя приложения, которому будет отсылаться команда. Литеральное строковое значение, строковый тег, или строковое выражение.

topic
Имя топика в приложении, которому будет отсылаться команда. Литеральное строковое значение, строковый тег, или строковое выражение.

Command
Команда, которая будет отсылаться. Литеральное строковое значение, строковый тег, или строковое выражение.

Возвращаемое значение

-1 – команда выполнилась неудачно. Возможная причина: приложение не запущено, топик не существует или команда содержит ошибку.

0 – команда выполнилась неудачно. Приложение занято.

1 – команда выполнилась успешно.

Пример

Скрипт приказывает Microsoft Excel, выполнить макрос *Macro1*, путем отсылки команды *[Run(“Macro1”,0)]* в Excel.

```
Macro="Macro1";
Command="[Run(" + StringChar(34) + Macro +
StringChar(34) + ",0)]";
```

Функция WWRequest()

Чтение данных из DDE элементов приложения. Можно использовать эту функцию для считывания значения из листа Microsoft Excel.

Синтаксис

```
Result = WWRequest (appname, topic, item, messagetag)
```

Параметры

appname
Имя приложения. Литеральное строковое значение, строковый тег, или строковое выражение.

topic
Имя топика в приложении. Литеральное строковое значение, строковый тег, или строковое выражение.

item
Имя элемента, принадлежащего топику и приложению. Литеральное строковое значение, строковый тег, или строковое выражение.

messagetag
Строковый тег, в который передается значение элемента. Значение строкового тега может быть преобразовано в целочисленное или вещественное значение при помощи функций StringToInt() или StringToReal().

Возвращаемое значение

-1 – данные считались неудачно. Возможная причина: приложение не запущено или топик или элемент не существует.

0 – данные считались неудачно. Приложение занято.

1 – данные считались успешно.

Пример

Скрипт считывает значение содержимое ячейки A1 (Row1, Column 1) лист *Sheet1*, файл *Book1.xls* в тег *CellValue*.

```
Result = WWRequest("excel","[Book1.xls]sheet1",
"r1c1",Mtag);
CellValue=StringToReal(MTag);
```

Функция WWPoke()

Запись данных в элемент приложения. Можно использовать эту функцию для считывания значения из листа Microsoft Excel.

Синтаксис

```
Result = WWPoke (appname, topic, item, string)
```

Параметры

appname

Имя приложения. Литеральное строковое значение, строковый тег, или строковое выражение.

topic

Имя топика в приложении. Литеральное строковое значение, строковый тег, или строковое выражение.

item

Имя элемента, принадлежащего топику и приложению. Литеральное строковое значение, строковый тег, или строковое выражение.

string

Значение, которое будет записано. Литеральное строковое значение, строковый тег, или строковое выражение. Помощи функций StringFromInt(), StringFromReal() или Text(),целочисленное или вещественное значение может быть преобразовано в строковый тег.

Возвращаемое значение

-1 – данные записаны неудачно. Возможная причина: приложение не запущено или топик или элемент не существует.

0 – данные записаны неудачно. Приложение занято.

1 – данные записаны успешно.

Примечание

Не использовать функции WWPoke() или WWRequest() для чтения и записи данных между приложениями InTouch различных узлов сети или сессий. Для чтения и записи данных между различными узлами сети или сессиями, необходимо использовать Access Name.

Пример

Данный скрипт помещает значение вещественного тега *CellValue* в строковый тег *Mtag* и записывает значение в ячейку *A1* (Row1, Column 1) лист *Sheet1*, файл *Book1.xls* .

```
MTag = Text(CellValue,”0”);
Result = WWPoke("excel","[Book1.xls]sheet1",
"r1c1",Mtag);
```

Работа с файлами

Можно написать скрипт для различных операций с файлами.

Функция	Использование
FileCopy()	Копирование файлов
FileDelete()	Удаление файлов
FileMove()	Перемещение файлов
FileReadFields() FileWriteFields()	Чтение/запись csv данных
FileReadMessage() FileWriteMessage ()	Чтение/запись текстовых данных

Управление файлами

В скрипте можно копировать, удалить, перемещать файлы.

Функция FileCopy()

Копирует файл-источник в принимающий файл и возвращает статус результата. Данная функция может выполняться достаточно долго и выполняется несколькими этапами:

1. Вызывается функция FileCopy() и немедленно возвращает результат, отображающий успешность инициализации операции копирования файлов.
2. Функция FileCopy()выполняет процедуру копирования в фоновом режиме. Таким образом скрипты InTouch продолжают выполняться, пока происходит копирование. Можно контролировать ход выполнения операции копирования, при помощи целочисленного тега.
3. Функция FileCopy() возвращает результат копирования, отображающий успешность или неудачи операции копирования.

Если папка-получатель недоступна, функция ждет 10 секунд, а затем записывает сообщение в логгер.

> **Примечание** Не использовать функцию FileCopy() в асинхронных функциях.

Синтаксис

```
Result = FileCopy (sourcefile, destfile, progresstag)
```

Параметры

sourcefile

Полный путь и имя файла, который будет копироваться. Литеральное строковое значение, строковый тег, или строковое выражение. Можно использовать трафаретные символы (* и ?) в данном параметре для копирования файлов, отвечающему определенному критерию. Путь может быть также UNC путем.

destfile

Полный путь и имя файла (или просто путь), куда будет копироавться файл. Литеральное строковое значение, строковый тег, или строковое выражение. Путь может быть также UNC путем.

progresstag

Имя целочисленного тега в двойных кавычках, который содержит значение, отображающее процесс копирования файлов. Литеральное строковое значение, строковый тег (например, строковый тег содержащий значение “IntTag.name”), или строковое выражение.

0 – процедура FileCopy() все еще выполняется.

1 – процедура FileCopy() завершена успешно.

-1 – процедура FileCopy() завершена с ошибками.

Возвращаемое значение

1 – FileCopy() функция вызвана успешно.

0 – Ошибка при вызове функции FileCopy(), в связи с тем, что в данный момент выполняется другая функция FileCopy().

-1 – Ошибка при вызове функции FileCopy(), в связи с тем, что файл-источник не существует или каталог-приёмник открыт только для чтения.

Пример

Скрипт копирует файл c:\MyData\output.log в каталог d:\archive и переименовывает файл в output.txt. Состояние копирования файла записывается в целочисленный тег *Monitor*.

```
Status=FileCopy(“c:\MyData\output.txt”,”d:\archive\
output.txt”,”Monitor”);
```

Скрипт копирует все файлы, у которых расширение .txt из корневого каталога c:\ в директорию c:\backup.

```
Status=FileCopy("c:\*.txt", "c:\Backup", "Monitor");
```

Скрипт копирует файл, полный путь и имя файла которых содержится в теге LogMessage в каталог d:\results\ и переименовывает в log*xxx*.txt, где *xxx* это метка времени.

```
Status = FileCopy(LogFile, “c:\results\log” +
$DateString + $TimeString + ”.txt”, “Monitor”);
```

Функция FileDelete()

Удаляет отдельный файл.

Синтаксис

```
Result = FileDelete (filename)
```

Параметры

filename

Путь и имя файла, который необходимо удалить. Литеральное строковое значение, строковый тег, или строковое выражение.

Примечание

Не использовать трафаретные символы (* и ?) с функцией FileDelete() и не использовать функцию FileDelete() в в асинхронных функциях.

Функция FileDelete() не удаляет каталоги.

Возвращаемое значение

1 – файл удален успешно.

0 – операция удаления выполнена не удачно. Возможная причина – удаление несуществующего файла или файл открыт только для чтения.

Пример

Скрипт удаляет файл c:\ Data.txt и возвращает 1, если файл был найден и удален успешно.

```
Status = FileDelete("c:\Data.txt");
```

Функция FileMove()

Перемещает файл-источник в принимающий файл и возвращает статус результата. Может быть также использована для переименования файла. Данная функция может выполняться достаточно долго и выполняется несколькими этапами:

1. Вызывается функция FileMove() и немедленно возвращает результат, отображающий успешность инициализации операции перемещения файлов.
2. Функция FileMove() выполняет процедуру перемещения в фоновом режиме. Таким образом, скрипты InTouch продолжают выполняться, пока происходит перемещение. Можно контролировать ход выполнения операции перемещения, при помощи целочисленного тега.
3. Функция FileMove() возвращает результат перемещения, отображающий успешность или неудачи операции.

Не использовать функцию FileMove() в асинхронных функциях

Синтаксис

```
Result = FileMove (sourcefile, destfile, progresstag)
```

Параметры

sourcefile

Полный путь и имя файла, который будет перемещаться. Литеральное строковое значение, строковый тег, или строковое выражение. Можно использовать трафаретные символы (* и ?) в данном параметре для перемещения файлов, отвечающему определенному критерию. Путь может быть также UNC путем.

destfile

Полный путь и имя файла (или просто путь), куда будет перемещаться файл. Литеральное строковое значение, строковый тег, или строковое выражение. Путь может быть также UNC путем.

progresstag

Имя целочисленного тега в двойных кавычках, который содержит значение, отображающее процесс перемещения файлов. Литеральное строковое значение, строковый тег (например, строковый тег, содержащий значение “IntTag.name”), или строковое выражение.

0 – процедура FileMove() все еще выполняется.

1 – процедура FileMove() завершена успешно.

-1 – процедура FileMove() завершена с ошибками.

Возвращаемое значение

1 – FileMove() функция вызвана успешно.

0 – Ошибка при вызове функции FileMove(), в связи с тем, что в данный момент выполняется другая функция FileMove().

-1 – Ошибка при вызове функции FileMove(), в связи с тем, что файл-источник не существует.

Пример

Скрипт перемещает файл c:\MyData\output.log в каталог d:\archive и переименовывает файл в output.txt. Состояние копирования файла записывается в целочисленный тег *Monitor*.

```
Status=FileMove(“c:\MyData\output.txt”,”d:\archive\
output.txt”,”Monitor”);
```

Скрипт перемещает все файлы, у которых расширение .txt из корневого каталога c:\ в директорию c:\backup.

```
Status=FileMove("c:\*.txt", "c:\Backup", "Monitor");
```

Скрипт перемещает файл, полный путь и имя файла которых содержится в теге LogMessage в каталог d:\results\ и переименовывает в log*xxx*.txt, где *xxx* это метка времени.

```
Status = FileMove(LogFile, “c:\results\log” +
$DateString + $TimeString + ”.txt”, “Monitor”);
```

Чтение и запись CSV данных

Можно написать скрипт для чтения и записи данных, содержащихся в csv файле, в/из набор тегов при помощи функций FileReadFields() и FileWriteFields().

Функции FileReadFields() и FileWriteFields() поддерживают только запятую в качестве разделителя.

Функция FileReadFields()

Считывает значения, содержащиеся в csv файле, в набор тегов,. Можно использовать данную функцию для подгрузки значений в набор тегов.

Поддерживаются только запятая в качестве разделителя.

Синтаксис

```
Result = FileReadFields (filename, offset, starttag, numberfields)
```

Параметры

filename
Имя csv файла, из которого будут считываться данные. Литеральное строковое значение, строковый тег, или строковое выражение.

offset
Положение (в байтах) в файле, с которого будет начинаться считывание. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение.

starttag
Имя первого тега, содержащего первые значения, которые будут записаны. Имя тега должно быть в двойных кавычках и заканчиваться числом, например “MyTag1”. Литеральное строковое значение, строковый тег, или строковое выражение.

numberfields
Количество данных, считываемое из csv файла. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение. Первое значение считывается из файла в тег, определенный параметром starttag, далее данные считываются последовательно в теги, с инкрементированным числовым суффиксом начиная со стартового параметра (MyTag1, MyTag2, MyTag3,…).

Возвращаемое значение

Дополнительное новое смещение в фале (в байтах), после считывания данных. Может быть использовано для считывания следующей последовательности данных.

Пример

Скрипт считывает значения “Flour” в тег RecipeTag1, 27.23 в RecipeTag2, 14, в RecipeTag3, 1 в RecipeTag4 и возвращает новое смещение в файле. Если файл c:\set.csv содержит следующие данные: Flour, 27.23, 14, 1 и если заданы теги RecipeTag1:message, RecipeTag2:real, RecipeTag3:integer, RecipeTag4:descrete.

```
FileReadMessage(“c:\set.csv”,0,”RecipeTag1”,4);
```

Функция FileWriteFields()

Записывает значения, содержащиеся в наборе тегов, в csv файл. Можно использовать данную функцию для сохранения значений набора тегов.

Поддерживаются только запятая в качестве разделителя.

Синтаксис

```
Result = FileWriteFields (filename, offset, starttag, numberfields)
```

Параметры

filename
: Имя csv файла, в который будут записываться данные. Если файла не существует, то создается новый файл. Литеральное строковое значение, строковый тег, или строковое выражение.

offset
: Положение (в байтах) в файле, с которого будет начинаться запись. Использовать -1 для записи в конец файла. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение.

starttag
: Имя первого тега, содержащего первые значения, которые будут записаны. Имя тега должно быть в двойных кавычках и заканчиваться числом, например “MyTag1”. Литеральное строковое значение, строковый тег, или строковое выражение.

numberfields
: Количество данных, записываемых в csv файл. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение. Первое значение записывается в файл из тега в параметре starttag, далее данные записываются последовательно из тегов с инкрементированным числовым суффиксом начиная со стартового параметра (MyTag1, MyTag2, MyTag3,…).

Возвращаемое значение

Дополнительное новое смещение в фале (в байтах), после записывания данных. Может быть использовано для записи следующей последовательности данных.

Пример

Набор тегов определен таким образом:

Тег	Тип данных	Значение
RecipeTag1	Message	Flour
RecipeTag2	Real	27.23
RecipeTag3	Integer	14
RecipeTag4	Discrete	1

Данный скрипт записывает значения, содержащиеся в тегах от RecipeTag1 до RecipeTag4 в .csv файл c:\set.csv.

```
FileWriteFields (“c:\set.csv”,0,”RecipeTag1”,4);
```

Таким образом, файл c:\set. .csv будет содержать следующие данные:

```
Flour, 27.23, 14, 1
```

Чтение и запись текстовых данных

В скрипте можно читать и записывать данные, содержащиеся в csv файле, в/из набора тегов при помощи функций FileReadMessage() и FileWriteMessage (). Также можно читать/записывать определенное количество байт или целую строку текста (разграниченную знаком смещения строки).

Функция FileReadMessage()

Считывает определенное количество байт (или одну строку) или строковые данные из файла.

Синтаксис

```
Result = FileReadMessage (filename, offset, messagetag, charstoreed)
```

Параметры

filename
: Имя файла, из которого будут считываться данные. Литеральное строковое значение, строковый тег, или строковое выражение.

offset
: Положение (в байтах) в файле, с которого будет начинаться считывание. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение.

messagetag
: Строковый тег, который получает первую строку или определенное количество байт из файла.

charstoreed
: Количество байт, которое будет считано из файла. Установить в 0, если необходимо считать до следующей символа смещения строки. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение.

Возвращаемое значение

Содержит новое байтовое положение после чтения. Можно использовать для последовательного считывания из файла.

Пример

Скрипт считывает первую строку данных в файла c:\Data\File.txt в строковый тег *MsgTag*.

```
FileReadMessage ("c:\Data\File.txt",0,MsgTag, 0);
```

Функция FileWriteMessage()

Записывает определенное количество байт (или одну строку) или строковые данные в файл.

Синтаксис

```
Result = FileWriteMessage (filename, offset, messagetag, linefeed)
```

Параметры

filename
Имя файла, в который будут записываться данные. Литеральное строковое значение, строковый тег, или строковое выражение.

offset
Положение (в байтах) в файле, с которого будет начинаться записывание. Установить в -1, для записи данных в конец файла. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение.

messagetag
Строковый тег, содержащий данные, которые будут записаны в файл.

linefeed
Определяет, необходимо ли записывать символ смещения строки (LF), после окончания записи данных в файл. Установить в 1 для записи символа перемещения строки, иначе установить в 0. Литеральное дискретное значение, дискретный тег, или дискретное выражение.

Возвращаемое значение

Содержит новое байтовое положение после записи данных. Может быть использовано для последовательной записи данных.

Пример

Данный скрипт записывает значения содержащиеся в теге *MsgTag* в конец файла c:\Data\File.txt.

```
FileWriteMessage ("c:\Data\File.txt",-1,MsgTag, 1);
```

Получение системной информации

В скрипте можно получать системную информацию.

Функция	Использование
GetNodeName()	Получение имени узла сети для данного компьютера.
InfoDisk()	Получение информации о пространстве на диске.
InfoFile()	Получение информации о файле.
InfoResources()	Получение информации о операционное среде Windows.

Получение имени узла сети для компьютера

В скрипте можно получать информацию об имени узла компьютера при помощи функции GetNodeName(). Данная функция может быть использована, например, для того чтобы сделать приложение InTouch динамическим при работе с именами доступа.

Функция GetNodeName()

Возвращает имя узла сети для данного компьютера.

Синтаксис

```
GetNodeName (messagetag, nodenum)
```

Параметры

messagetag
Строковый тег, который будет содержать имя узла сети.

nodenum
Количество символов, получаемое от имени узла. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение в диапазоне от 0 до 131.

Пример

Данный скрипт получает имя узла и присваивает его строковому тегу NodeName. Так как 131 символ это максимальная длинна строкового тега в InTouch HMI, имя узла сети может быть обрезано.

```
GetNodeName("NodeName",131);
```

Получение информации о пространстве на диске

В скрипте можно получать информацию о пространстве на диске при помощи функции InfoDisk(). Можно получить:

- Общий объем диска (в байтах или килобайтах).
- Доступное свободное место на диске (в байтах или килобайтах).

Можно также определить когда и как часто обновляется информация (в анимационной связи) при помощи назначения тега.

Функция InfoDisk()

Возвращает общее свободное место либо на локальном диске либо на удалённый диск в локальной сети.

Синтаксис

```
Result = InfoDisk (drive, infotype, trigger)
```

Параметры

drive

Буква (имя) диска для которого необходимо получить информацию. Используется только первый символ строки. Литеральное строковое значение, строковый тег, или строковое выражение

infotype

Определяет тип информации. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение со следующим значением:

1 - Функция возвращает общий размер диска (в байтах)

2 - Функция возвращает размер свободного места на диске (в байтах)

3 - Функция возвращает общий размер диска (в килобайтах)

4 - Функция возвращает размер свободного места на диске (в килобайтах)

trigger

Тег (или выражение), который работает как триггер для пересчета информации о диске. Если триггер переключается, то производится обновление информации о диске. Дискретный или аналоговый тег, или дискретное или аналоговое выражение.

Примечание

Триггерный тег имеет смысл использовать, только если функция InfoDisk() задействована в анимационной связи. Если функция используется в скрипте, то можно определять любое литеральное числовое значение, аналоговый тег, или числовое выражение.

Пример

Скрипт используется в анимационной связи, отображает свободное пространство на диске C и обновляется каждую минуту.

```
InfoDisk(“C”, 4, $Minute)
```

Получение информации о файле или каталоге

В скрипте можно получать информацию о определенном файле или каталоге при помощи функции InfoFile(). При помощи различных параметров, можно узнать:

- Существует ли файл.
- Находится ли определенный файл в каталоге.
- Размер (в байтах) файла.
- Временную метку файла или каталога
- Количество файлов, которые совпадают при поиске по шаблону.

Функция InfoFile()

Возвращает общее свободное место либо на локальном диске либо на удалённый диск в локальной сети.

Синтаксис

```
Result = InfoFile (filename, infotype, trigger)
```

Параметры

filename

Полное имя файла или каталога, о котором необходимо получить информацию. Литеральное строковое значение, строковый тег, или строковое выражение. Могут быть также шаблонными символами, такими как “*” и “?”.

infotype

Определяет тип информации, которую необходимо получить о определенном файле или каталоге. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение со следующим значением:

1 - Существование. Функция возвращает 1 - если файл существует, 2 - если каталог существует, 0 – если файл или каталог не существуют.

2 - Размер. Функция возвращает размер файла в байтах.

3 - Временная метка создания. Функция возвращает временную метку как время в секундах прошедшее с 1 января 1970. Использовать функцию StringFromTimeLocal() для конвертирования данного значения в строковую метку времени.

4 - Количество файлов, которые совпадают при поиске по шаблону.

trigger

Тег (или выражение), который работает как триггер для пересчета информации о файле. Если триггер переключается, то производится обновление информации о файле. Дискретный или аналоговый тег, или дискретное или аналоговое выражение.

Примечание

Триггерный тег имеет смысл использовать, только если функция InfoFile() задействована в анимационной связи. Если функция используется в скрипте, то можно определять любое литеральное числовое значение, аналоговый тег, или числовое выражение.

Пример

Скрипт возвращает 1 если файл c:\data\log.txt

```
InfoFile(“c:\data\log.txt”,1,$minute)
```

Скрипт возвращает 14223 если файл c:\data\log.txt имеет размер 14223 байта.

```
InfoFile(“c:\data\log.txt”,2,$minute)
```

Скрипт возвращает 1138245266 если файл c:\data\log.txt был создан 26 января 2006 года в 11:14:26 AM.

```
InfoFile(“c:\data\log.txt”,3,$minute)
```

Скрипт возвращает 14 если в каталоге c:\data\ имеется 14 файлов с расширением txt.

```
InfoFile(“c:\data\*.txt”,4,$minute)
```

Получение информации о операционное среде Windows

В скрипте можно получать информацию о операционное среде Windows. Можно узнать:

- Свободные байты файла виртуальной памяти.
- Приблизительное количество Windows задач.

Функция InfoResources()

Возвращает свободные байты файла виртуальной памяти или приблизительное количество Windows задач.

Синтаксис

```
Result = InfoResources (infotype, trigger)
```

Параметры

infotype

Определяет тип информации, которую необходимо получить о операционное среде Windows. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение со следующим значением:

1 - Свободные байты файла виртуальной памяти.

2 - Приблизительное количество Windows задач. Может быть использовано для определения загрузки системы.

trigger

Тег (или выражение), который работает как триггер для пересчета информации о файле. Если триггер переключается, то производится обновление информации о файле. Дискретный или аналоговый тег, или дискретное или аналоговое выражение.

Примечание

Триггерный тег имеет смысл использовать, только если функция InfoResources() задействована в анимационной связи. Если функция используется в скрипте, то можно определять любое литеральное числовое значение, аналоговый тег, или числовое выражение.

Пример

Скрипт получает приблизительное количество Windows задач, и при использовании в анимационной связи, информация обновляется каждую минуту.

```
InfoResources(2,$second);
```

Получение информации связанной с InTouch

В скрипте можно получать информацию, связанную с InTouch, при помощи следующих функций:

Функция	Использование
InfoInTouchAppDir()	Получает информацию о каталоге приложения InTouch, которое разрабатывается.
InTouchVersion()	Получает информацию о версии InTouch.

Получение имени каталога приложения InTouch

В скрипте можно получить информацию о каталоге приложения InTouch, которое в данный момент работает при помощи функции InfoInTouchAppDir(). Данная функция полезна для определения в пути любого внешнего файла, которое используется в приложении InTouch.

Функция InfoInTouchAppDir()

Возвращает текущий каталог приложения InTouch.

Синтаксис

```
Result = InfoInTouchAppDir()
```

Возвращаемое значение

Строковый тег, который содержит каталог текущего работающего приложения InTouch.

Примечание

Имя каталога приложения может быть обрезано при передаче в строковый тег или отображении в анимационной связи, в связи с ограничением в 131 символ.

Пример

Скрипт может вернуть c:\documents and settings\user1\my documents\my intouch applications\packaging.

```
InfoInTouchAppDir();
```

Получение версии InTouch

В скрипте можно получить информацию о версии InTouch, которое в данный момент работает при помощи функции InTouchVersion ().

Функция InTouchVersion ()

Возвращает полный номер версии InTouch или только часть.

Синтаксис

```
Result = InTouchVersion (infotype)
```

Параметры

Infotype

Определяет, как будет возвращаться информация о версии. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение со следующим значением:

0 – функция возвращает номер версии полностью.

1- функция возвращает только основной номер версии.

2- функция возвращает только дополнительный номер версии.

3- функция возвращает только номер патча.

4- функция возвращает только номер билда.

Пример

Функция	Возможный результат
InTouchVersion(0)	9.5.0 1101.0377.0093.0031
InTouchVersion(1)	9
InTouchVersion(2)	5
InTouchVersion(3)	0
InTouchVersion(4)	1101

Скрипты связанные с системой безопасности

В скрипте можно добавлять и управлять системой безопасности внутри приложения InTouch при помощи различных встроенных функций и системных тегов.

Вход и выход из системы

Для входа и выхода пользователя в систему можно использовать следующие функции.

Функция	Использование
AttemptInvisibleLogon()	Вход пользователя в систему, путем передачи аутентификационных данных в параметрах.
LogonCurrentUser()	Вход в систему текущего пользователя Windows (если выбрана система безопасности “OS”)
PostLogonDialog()	Показать диалоговое окно входа пользователя в систему.
Logoff()	Выход пользователя из системы.

Установка и изменение пароля

Для изменения пароля можно использовать следующие функции и системные теги:

Функция	Использование
ChangePassword()	Вызывает диалоговое окно Change Password для текущего вошедшего в систему пользователя.
$ChangePassword	Вызывает диалоговое окно Change Password для текущего вошедшего в систему пользователя.
$PasswordEntered	Установка пароля.

Определение и конфигурирование пользователей

Для определения и конфигурирования пользователей можно использовать следующие функции и системные теги:

Функция	Использование
$ConfigureUsers	Вызывает диалоговое окно Configure Users.
$OperatorEntered	Устанавливает корректное имя пользователя
$OperatorDomainEntered	Устанавливает корректное имя пользователя домена (если выбрана система безопасности “OS”)

Управление системой безопасности и другой информацией

Для управления системой безопасности можно использовать следующие функции и системные теги:

Функция	Использование
$AccessLevel	Получение уровня доступа, вошедшего в систему пользователя.
AddPermission()	Назначение уровней доступа определенной группе пользователей (локально/домен)
GetAccountStatus()	Получение информации об учетной записи (окончание срока действия пароля, блокировки, флаги)
$InactivityTimeout	Отображение времени, которое осталось до того как пользователь автоматически выйдет из системы.
$InactivityWarning	Отображает время до предупреждения о истечении времени.
InvisibleVerifyCredentials()	Получение информации об InTouch уровне доступа пользователя операционной системы.
IsAssignedRole()	Просмотр, является ли текущий пользователь членом определенной роли.
QueryGroupMembership()	Просмотр, является ли текущий пользователь членом определенной роли.

Различные функции

Скрипты InTouch поддерживают воспроизведение звука, и таким образом можно связать человеко-машинное взаимодействие со звуками. Скрипты InTouch также поддерживают установку и получение свойств мастеров (Wizards).

Воспроизведение звуковых файлов из приложения InTouch

В скрипте можно привязывать события и условия с определенными звуками. Например, можно связать предупреждающее об аварии окно или критическое условие с предупреждающим сигналом.

Функция PlaySound()

Проигрывает звук из WAV файла или стандартный звук Windows.

Синтаксис

```
PlaySound (soundname, flag)
```

Параметры

soundname

Имя звукового WAV файла. Литеральное строковое значение, строковый тег, или строковое выражение. Звуковой файл также может быть определен в Win.ini файле в разделе [Sounds], например MC= “C:\test.wav”.

flag

Определяет как будет проигрываться файл. Литеральное целочисленное значение, целочисленный тег, или целочисленное выражение со следующим значением:

0 – Проигрывать звук один раз, синхронно, то есть выполнение скрипта приостанавливается, пока звук не завершит проигрывание.

1 – Проигрывать звук один раз, асинхронно, то есть выполнение скрипта не приостанавливается, пока проигрывается звук.

9 – Проигрывать звук непрерывно (пока функция PlaySound() не будет вызвана снова).

Пример

Скрипт проигрывает звук Alert непрерывно. В файле Win.ini в разделе [Sounds], производится привязка звукового файла с именем: Alert = C:\alert.wav.

```
PlaySound (“Alert”,9);
```

Установка и получение свойств мастеров

Некоторые мастера, такие как Distributed Alarm Object и Windows Controls, содержат свойства, которые можно считывать и устанавливать. Данные свойства могут быть значениями в текстовом поле или содержать статус кнопки-флажка.

В скрипте можно получить доступ к данным свойствам при помощи функций:

Функция	Использование
SetPropertyD(), GetPropertyD()	Устанавливает и считывает свойства с дискретным типом данных.
SetPropertyI(), GetPropertyI()	Устанавливает и считывает свойства с целочисленным типом данных.
SetPropertyM(), GetPropertyM()	Устанавливает и считывает свойства с строковым типом данных.

Чтобы узнать список поддерживаемых мастером свойств, смотрите его описание.

Ниже приведено основное описание того, каким образом устанавливать и считывать свойства мастера.

Функция GetPropertyD()

Считывает свойства с дискретным типом данных в мастере и возвращает код успешности операции

Синтаксис

```
Result = GetPropertyD (controlname.property, dtag)
```

Параметры

controlname

Имя мастера, поддерживающего свойства. Литеральное строковое значение, строковый тег, или строковое выражение.

property

Свойство мастера, которое будет считано. Вместе с *controlname* может быть литеральным строковым значением, строковым тегом, или строковым выражением.

dtag

Дискретный тег, в который будет записано состояние свойство мастера.

Возвращаемое значение

Код ошибки. См. раздел Сообщения об Ошибках Windows компонентов, Глава 5 Мастера, руководства По разработке визуализации InTouch HMI.

Пример

При помощи следующего скрипта проверять свойство visibility, компонента *Checkbox1* и дискретного тега *dtag*.

```
result=GetPropertyD(“Checkbox1.visible”,dtag);
```

Скрипт устанавливает *dtag* в 1, если компонент *Checkbox1* видим, иначе *dtag* в 0.

Функция SetPropertyD()

Устанавливает свойство с дискретным типом данных в мастере и возвращает код успешности операции

Синтаксис

```
Result = SetPropertyD (controlname.property, Boolean)
```

Параметры

controlname

Имя мастера, поддерживающего свойства. Литеральное строковое значение, строковый тег, или строковое выражение.

property

Свойство мастера, которое будет установлено. Вместе с *controlname* может быть литеральным строковым значением, строковым тегом, или строковым выражением.

Boolean

Дискретный тег, значение из которого будет передано в свойство мастера. Литеральное дискретное значение, дискретный тег, дискретное выражение.

Возвращаемое значение

Код ошибки. См. раздел Сообщения об Ошибках Windows компонентов, Глава 5 Мастера, руководства По разработке визуализации InTouch HMI.

Пример

При помощи следующего скрипта можно управлять видимостью компонента *Checkbox1* .

```
result=SetPropertyD(“Checkbox1.visible”,dtag);
```

Если установить *dtag* в 0, и вызвать скрипт приведенный выше, то компонент *Checkbox1* станет невидимым.

Функция GetPropertyI()

Считывает свойства с целочисленным типом данных в мастере и возвращает код успешности операции

Синтаксис

```
Result = GetPropertyI (controlname.property, itag)
```

Параметры

controlname

Имя мастера, поддерживающего свойства. Литеральное строковое значение, строковый тег, или строковое выражение.

property

Свойство мастера, которое будет считано. Вместе с *controlname* может быть литеральным строковым значением, строковым тегом, или строковым выражением.

itag

Целочисленный тег, в который будет записано состояние свойство мастера.

Возвращаемое значение

Код ошибки. См. раздел Сообщения об Ошибках Windows компонентов, Глава 5 Мастера, руководства По разработке визуализации InTouch HMI.

Пример

При помощи следующего скрипта проверять выбранный элемент группы компонента *Radiobutton1*.

```
result=GetPropertyI("Radiobutton1.value",itag);
```

Скрипт устанавливает *itag* в 1(2,3…), если выбран первый (второй, третий…) элемент группы.

Функция SetPropertyI()

Устанавливает свойство с целочисленным типом данных в мастере и возвращает код успешности операции

Синтаксис

```
Result = SetPropertyI (controlname.property, integer)
```

Параметры

controlname

Имя мастера, поддерживающего свойства. Литеральное строковое значение, строковый тег, или строковое выражение.

property

Свойство мастера, которое будет установлено. Вместе с *controlname* может быть литеральным строковым значением, строковым тегом, или строковым выражением.

integer

Целочисленный тег, значение из которого будет передано в свойство мастера. Литеральное целочисленное значение, целочисленный тег, целочисленное выражение.

Возвращаемое значение

Код ошибки. См. раздел Сообщения об Ошибках Windows компонентов, Глава 5 Мастера, руководства По разработке визуализации InTouch HMI.

Пример

При помощи следующего скрипта управлять компонентом *Radiobutton1*, и установить второй элемент в группе.

```
result=SetPropertyI((“Radiobutton1.value”,2);
```

Функция GetPropertyM()

Считывает свойства с строковым типом данных в мастере и возвращает код успешности операции

Синтаксис

```
Result = GetPropertyM (controlname.property, mtag)
```

Параметры

controlname

Имя мастера, поддерживающего свойства. Литеральное строковое значение, строковый тег, или строковое выражение.

property

Свойство мастера, которое будет считано. Вместе с *controlname* может быть литеральным строковым значением, строковым тегом, или строковым выражением.

mtag

Строковый тег, в который будет записано состояние свойство мастера.

Возвращаемое значение

Код ошибки. См. раздел Сообщения об Ошибках Windows компонентов, Глава 5 Мастера, руководства По разработке визуализации InTouch HMI.

Пример

При помощи следующего скрипта можно проверять заголовок компонента *Checkbox1*.

```
result=GetPropertyM(“Checkbox1.caption”,mtag);
```

Скрипт записывает в mtag заголовок компонента Checkbox1.

Функция SetPropertyM()

Устанавливает свойство с целочисленным типом данных в мастере и возвращает код успешности операции

Синтаксис

```
Result = SetPropertyI (controlname.property, message)
```

Параметры

controlname

Имя мастера, поддерживающего свойства. Литеральное строковое значение, строковый тег, или строковое выражение.

property

Свойство мастера, которое будет установлено. Вместе с *controlname* может быть литеральным строковым значением, строковым тегом, или строковым выражением.

message

Строковый тег, значение из которого будет передано в свойство мастера. Литеральное строковое значение, строковый тег, строковое выражение.

Возвращаемое значение

Код ошибки. См. раздел Сообщения об Ошибках Windows компонентов, Глава 5 Мастера, руководства По разработке визуализации InTouch HMI.

Пример

При помощи следующего скрипта можно для компонента *Checkbox1* установить заголовок мастера динамически:

```
result=SetPropertyM(“Checkbox1.caption”,”Start Engine1”);
```

Скрипт устанавливает заголовок компонента Checkbox1 - ”Start Engine1”.

Глава 7

Скрипты с OLE объектами

Для того чтобы расширить возможности приложения InTouch HMI можно использовать OLE объекты. При помощи OLE объектов можно:

- Создавать диалоговые окна для интерфейса оператора.
- Доступ к функциям операционной системы, таким как Панель управления.
- Делать доступными данные из Manufacturing Execution Module для обработки внутри приложения InTouch HMI. См. документацию по Manufacturing Execution Module.

Создание, проверка и освобождение OLE объектов

Можно создавать и проверять OLE объекты для использования в скриптах InTouch. После использования OLE объекта можно освободить его, для освобождения памяти.

Использовать следующие функции для создания, проверки и освобождения OLE объекта:

- Функция OLE_CreateObject()
- Функция OLE_IsObjectValid()
- Функция OLE_ReleaseObject()

Функция OLE_CreateObject()

Перед тем как в скрипте обращаться к OLE объекту, необходимо создать его. После создания OLE объекта Вы получаете указатель, связанный с OLE объектом.

В скрипте можно создать OLE объект и назначить указатель при помощи функции OLE_CreateObject().

Синтаксис

`OLE_CreateObject` *`(%pointer, classname)`*

Параметры

`%pointer`

Выбранное имя указателя, который будет связан с OLE объектом. Может содержать буквенно-цифровые символы (A-Z, 0-9) и символ подчеркивания. Регистр символов не учитывается.

`classname`

Имя OLE класса. В имени класса регистр символов учитывается. Литеральное строковое значение, строковый тег, или строковое выражение.

Примечание

Если использовать то же имя объекта для создания другого OLE объекта, объект будет обновлен для связи с новым OLE классом. И освобожден от старого OLE класса.

Пример

Скрипт создает OLE объект, называющийся %WShell, который связан с классом Wscript.Shell

```
OLE_CreateObject(%WShell, “Wscript.Shell”);
```

Функция OLE_IsObjectValid()

В скрипте можно проверить достоверность/допустимость OLE объекта, при помощи функции OLE_IsObjectValid(). Это не является необходимым шагом при работе с OLE объектами, но рекомендуется убедится в том что при дальнейшей работе с OLE объектами не возникнет проблем.

Синтаксис

`Result = OLE_IsObjectValid(%pointer)`

Аргументы

%pointer

Имя указателя связанного OLE объектом, который будет проверяться.

`Result`

Дискретное значение, обозначающее следующее:

0 - OLE объект, который связан с указателем не достоверный/допустимый.

1- OLE объект, который связан с указателем достоверный/допустимый.

Пример

Скрипт создает OLE объект, основанный на классе Wscript.Shelli создает связанный с ним указатель %WS. *Isvalid* – это дискретный тег, который принимает значение TRUE, если объект создан успешно, иначе он принимает значение FALSE.

```
OLE_CreateObject(%WS, “Wscript.Shell”);
isvalid = OLE_IsObjectValid(%WS);
```

Функция OLE_ReleaseObject()

После того как в скрипте OLE объект был использован, можно освободить его и удалить его указатели для освобождения системных ресурсов. После того как OLE объект освобожден, нельзя использовать его указатель для доступа к свойствам и методам соответствующего OLE класса.

Синтаксис

`OLE_ReleaseObject` *`(%pointer)`*

Аргументы

`%pointer`

Имя указателя связанного OLE объектом. Может содержать буквенно-цифровые символы (A-Z, 0-9) и символ подчеркивания. Регистр символов не учитывается.

Пример

Скрипт освобождает OLE объект связанный с указателем %WShell и удаляет указатель %WShell.

```
OLE_ReleaseObject(%WShell);
```

Использование свойств и методов OLE объектов

В скрипте можно использовать указатели для чтения и записи значений из свойств OLE объектов. Указатели можно также использовать для вызова OLE методов. Свойства и методы доступны в зависимости от OLE объекта.

Доступ к свойствам OLE объекта

В скрипте можно подучить доступ к свойствам OLE объектов также как и в большинстве языков программирования. Свойства обычно определяются при помощи точки “.”оператора.

Примечание При использовании в скрипте свойств OLE объектов, необходимо убедится в том, что их ссылки не превышают 98 символов, включая “%”. Указатели OLE объектов должны быть как можно более короткими.

Считывание свойства OLE объекта

В скрипте можно считывать свойства OLE объекта, назначением свойству переменной.

Синтаксис

```
value = %pointer.property;
```

Аргументы

%pointer

Имя указателя связанного OLE объектом. Должен быть создан функцией OLE_CreateObject() или назначен другой указатель перед чтением свойства.

property

Имя свойства, которое будет считано.

value

Возвращаемое значение свойства OLE объекта.. Может быть присвоено тегу InTouch HMI или использовано для дальнейших вычислений. Но не может быть использовано напрямую в анимационной связи отображения.

Пример

Скрипт создает OLE объект, основанный на OLE классе *System.Random*, создает указатель *%SR* для связывания и присвоения свойства *.NextDouble* вещественному тегу *randtag*.

```
OLE_CreateObject(%SR,"System.Random");
randtag = %SR.NextDouble;
```

Запись в свойство OLE объекта

В скрипте можно записывать значение в свойство OLE объекта.

Синтаксис

```
%pointer.property = value;
```

Аргументы

%pointer

Имя указателя связанного OLE объектом. Должен быть создан функцией OLE_CreateObject() или назначен другой указатель перед записью значение в свойство.

property

Имя свойства, которое будет считано.

value

Значение, которое будет записано в свойство OLE объекта.. Литеральное значение, тег или выражение. Но не может быть использовано напрямую в анимационной связи ввода.

Вызов методов OLE объекта

В скрипте можно вызывать методов OLE объекта.

Синтаксис

```
%pointer.method(parameters)
```

Аргументы

%pointer

Имя указателя связанного OLE объектом. Должен быть создан функцией OLE_CreateObject() или назначен другой указатель перед вызовом метода.

method

Имя метода OLE объекта.

parameters

Список параметров, которые будут передаваться в метод. Параметры должны быть отделены запятой. Литеральные значения, теги или выражения.

Пример

Скрипт создает OLE объект, основанный на OLE классе *Shell.Application*, создает указатель *%sa* к OLE объекту и вызывает метод *.MinimizeAll*. Данный метод сворачивает все окна.

```
OLE_CreateObject(%SA,"Shell.Application");
%SA.MinimizeAll();
```

Примечание Не разрешены дополнительные параметры в скриптах OLE в InTouch HMI. Все параметры должны быть определены.

Назначение нескольких указателей одному OLE объекту

В скрипте можно назначить несколько указателей одному OLE объекту.

Синтаксис

```
%newpointer = %pointer
```

Аргументы

%pointer

Имя указателя уже связанного с OLE объектом.

%newpointer

Имя метода OLE объекта.

parameters

Новое имя указателя, который будет связан тем же OLE объектом. Может содержать буквенно-цифровые символы (A-Z, 0-9) и символ подчеркивания. Регистр символов не учитывается.

Пример

Скрипт создает OLE объект, основанный на OLE классе *Wscript.Shell.*, создает указатель *%WS*. Указатель *%WS2* при присвоении указывает на тот же OLE объект. Может быть использовано для чтения и записи в свойства и вызов методов одного и того же OLE объекта.

```
OLE_CreateObject(%WS,"Wscript.Shell");
%WS2=%WS;
```

Примечание Можно использовать строковые теги при подключении с указателями. Если присвоить тег указателю, он получает ID значение. Можно использовать для создания еще указателей на тот же OLE объект.

Выявление OLE ошибок

В скрипте, можно использовать OLE функции для выявления неисправностей и ошибок OLE.

Функция	Использование
OLE_GetLastObjectError()	Получение номера последней OLE ошибки
OLE_GetLastObjectErrorMessage()	Получить информацию по последней OLE ошибке
OLE_ResetObjectError()	Сброс последней ошибки
OLE_ShowMessageOnObjectError()	Показать или спрятать диалоговое окно сообщения OLE ошибки
OLE_IncrementOnObjectError()	Считать количество OLE ошибок в теге InTouch HMI.

Функция OLE_GetLastObjectError()

Функция возвращает номер последней OLE ошибки.

Синтаксис

```
errnum = OLE_GetLastObjectError();
```

Аргументы

errnum

Номер последней OLE ошибки.

Функция OLE_GetLastObjectErrorMessage()

Функция возвращает сообщение последней OLE ошибки.

Синтаксис

```
errmsg = OLE_GetLastObjectErrorMessage();
```

Аргументы

errmsg

Сообщение последней OLE ошибки.

Функция OLE_ResetObjectError()

В скрипте, можно использовать функцию OLE_ResetObjectError() для сбрасывания последней OLE ошибки. Таким образом, номер последней OLE ошибки устанавливается в 0, а сообщение последней OLE ошибки становится пустым.

Это может быть использовано для идентифицирования любых ошибок при работе нескольких OLE функций.

Синтаксис

```
OLE_ResetObjectError()
```

Функция OLE_ShowMessageOnObjectError()

По умолчанию при возникновении OLE ошибки, возникает окно с сообщением об ошибке.

В скрипте, можно определить будет или нет отображаться окно с сообщением об ошибке, при помощи функции OLE_ShowMessageOnObjectError().

Синтаксис

```
OLE_ShowMessageOnObjectError(Boolean)
```

Аргументы

Boolean

Значение, определяющее будет ли отображаться окно с сообщением об OLE ошибке. Литеральное дискретное значение, дискретный тег или дискретное выражение со следующим значением.

0 – при возникновении OLE ошибки, окно с сообщением об ошибке не возникает.

1 – при возникновении OLE ошибки, возникает окно с сообщением об ошибке.

Пример

Скрипт подавляет все окна с сообщениями об OLE ошибках. При возникновении OLE ошибки, не окно с сообщением об ошибке не отображается.

```
OLE_ShowMessageOnObjectError(0)
```

Функция OLE_IncrementOnObjectError()

В скрипте, можно использовать функцию OLE_IncrementOnObjectError() чтобы назначить целочисленный тег в качестве счетчика OLE ошибок.

Синтаксис

`OLE_IncrementOnObjectError(integertag)`

Параметры

integertag

Тег, который выступает в качестве счетчика.

Примечание

Если отображается диалоговое окно сообщения об OLE ошибке, то тег счетчик инкрементируется только после того как будет закрыто окно сообщения об OLE ошибке.

Пример

Скрипт назначает тег *errorcount* в качестве счетчика ошибок, скрывает диалоговое окно с сообщением об ошибке и пытается создать OLE объект, основанный на неправильном имени OLE класса. Что вызывает ошибку, значение тега *errorcount* инкрементируется на 1.

```
errorcount = 0;
OLE_IncrementOnObjectError(errorcount);
OLE_ShowMessageOnObjectError(0);
OLE_CreateObject(%WS,”InVaLiD.cLaSs.nAmE”);
```

Что можно сделать с OLE

Можно использовать следующие скрипты для того, чтобы получить представление о той эффективной функциональности, которую можно добавить в приложение при использовании OLE объектов.

Генерирование случайных чисел

В скрипте можно использовать следующие команды для генерирования случайных чисел от 0 до 255:

```
OLE_CreateObject(%SR,”System.Random”);
randtag = (%SR.NextDouble)*255;
```

Создание диалогового окна интерфейса пользователя

В скрипте можно использовать следующие команды создания диалогового окна интерфейса пользователя.

```
dim DlgBody as message;
dim DlgTitle as message;
dim Style as integer;
dim Result as integer;
DlgBody = “Do you want to open the valve ‘MR-3-FF’?”;
DlgTitle = “Confirm Opening Valve MR-3-FF”;
Style = 48;
OLE_CreateObject(%WS,"Wscript.Shell");
result = %WS.Popup(DlgBody,1,DlgTitle,Style);
```

Данный пример создает следующее окно.

Тег Style определяет, какой значок и какие кнопки появляются в диалоговом окне.

Значок	Стиль	Значение
(Нет значка)	Без значка	0
	Значок ошибки	16
	Значок вопроса	32
	Значок предупреждения	48
	Значок информационный	64

Для использования определенной кнопки, добавить одно из следующих значений к значению Style.

Значение	Стиль
0	Только кнопка OK
1	Кнопки OK и Cancel
2	Кнопки Abort, Retry, Ignore.
3	Кнопки Yes, No, Cancel
4	Кнопки Yes, No
5	Кнопки Retry, Cancel
6	Кнопки Cancel, Try Again, Continue

Результирующий тег содержит номер кнопки, которую нажал пользователь. Что может быть использовано для разветвления алгоритма скрипта InTouch. Возможны следующие коды:

Результат	Значение
1	Нажата кнопка OK
2	Нажата кнопка Cancel
3	Нажата кнопка Abort
4	Нажата кнопка Retry
5	Нажата кнопка Ignore
6	Нажата кнопка Yes
7	Нажата кнопка No
10	Нажата кнопка Try Again
11	Нажата кнопка Continue

Открытие Windows панели свойств даты и времени

В скрипте следующая команда открывает Windows панель свойств даты и времени:

```
OLE_CreateObject(%WP,"Shell.Application");
%WP.SetTime();
```

Можно делать аналогичные задачи при помощи вызова различных методов и передачи их в связанный OLE объект.

Метод	Открывает
TrayProperties()	Свойства трея
FileRun()	Диалоговое окно File Run
FindFiles()	Диалоговое окно Find Files
FindComputer()	Диалоговое окно Find Computer
ShutdownWindows()	Панель Shutdown Windows

Чтение и запись значений в реестр Windows

В скрипте можно использовать OLE для чтения и записи значений в реестр Windows при помощи:

- Создания OLE объекта на основе Windows класса Wscript.Shell.
- Использования методов RegRead() и RegWrite() объекта OLE.

Например, эти команды считывают установленную версию InTouch HMI напрямую из ключа реестра и сохранить в теге rkey.

```
OLE_CreateObject(%WS,"Wscript.Shell");
rkey = %WS.RegRead("HKLM\SOFTWARE\Wonderware\InTouch\
Installation\Version");
```

Эти команды записывают значение 1 в ключ реестра, который определяет, скрыты ли для текущего пользователя расширения файлов.

```
OLE_CreateObject(%WS,"Wscript.Shell");
%WS.RegWrite("HKCU\Software\Microsoft\Windows\Current
Version\Explorer\Advanced\HideFileExt",1,"REG_DWORD");
```

Сворачивание окон

В скрипте можно использовать следующие команды для сворачивания всех окон на рабочем столе:

```
OLE_CreateObject(%WA,"Shell.Application");
%WA.MinimizeAll();
```

Можно делать аналогичные задачи при помощи вызова следующих методов.

Метод	Открывает
TileHorizontally()	Расположить окна горизонтально
TileVertically()	Расположить окна вертикально
CascadeWindows()	Расположить окна каскадно
UndoMinimizeALL()	Восстановить все окна

Глава 8

Скрипты с ActiveX компонентами

Можно использовать ActiveX компоненты, для чтения и записи в теги. В скрипте можно ссылаться на ActiveX компоненты.

Можно также создать скрипты, которые выполняется, когда происходит событие ActiveX компонента. Эти скрипты могут быть использованы повторно и импортированы в другое приложение.

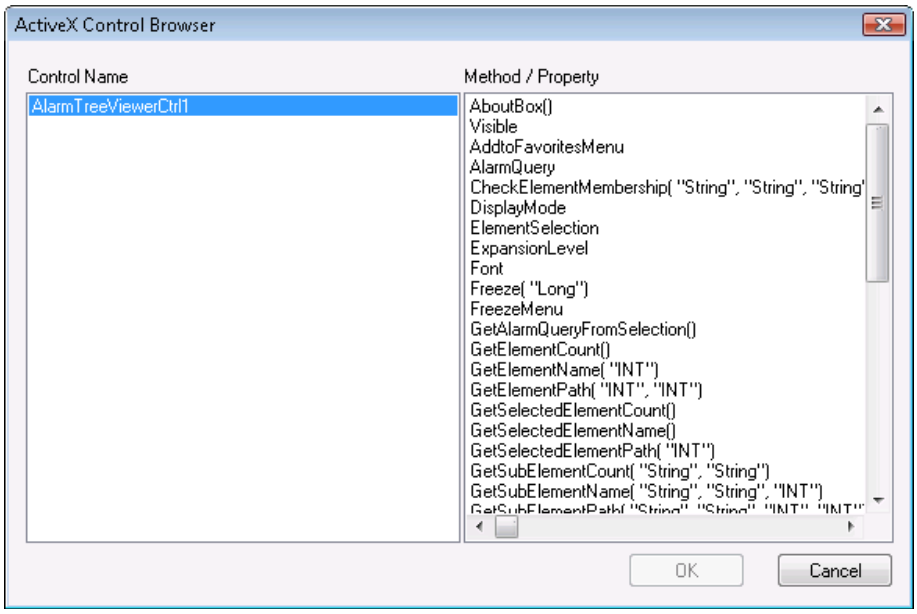
Вызов методов ActiveX компонента

В скрипте можно вызывать методы ActiveX компонента для выполнения операций поддерживаемых ActiveX компонента. Скрипты также могут быть выполнены по событию ActiveX компонента.

Примечание Для того чтобы вызвать метод ActiveX, когда произойдет ActiveX событие, необходимо сделать предварительную работу. См. раздел Конфигурирование скриптов ActiveX событий.

Для вызова метода ActiveX компонента

1. В диалоговом окне редактора скриптов, в меню Insert выберите ActiveX. Появится диалоговое окно ActiveX Control Browser.



2. Выбрать слева в панели имя ActiveX компонента. Справа в панели отобразятся имена свойств и методов поддерживаемых ActiveX компонентом.
3. Выбрать имя метода, который необходимо использовать, из панели справа. Метод и параметры вставятся в скрипт.
4. Сконфигурировать параметры метода внутри скобок.
5. Нажать ОК.

Доступ к свойствам ActiveX компонента из InTouch HMI

В скрипте можно читать и записывать свойства ActiveX компонентов для обмена данными между ActiveX и тегами InTouch. Можно также иметь доступ к свойствам ActiveX компонентов для установки дискретных значений.

Конфигурирование ActiveX компонента для чтения и записи данных

В скрипте можно считывать и записывать данные в свойства ActiveX элементов управления.

Сделать это можно двумя способами:

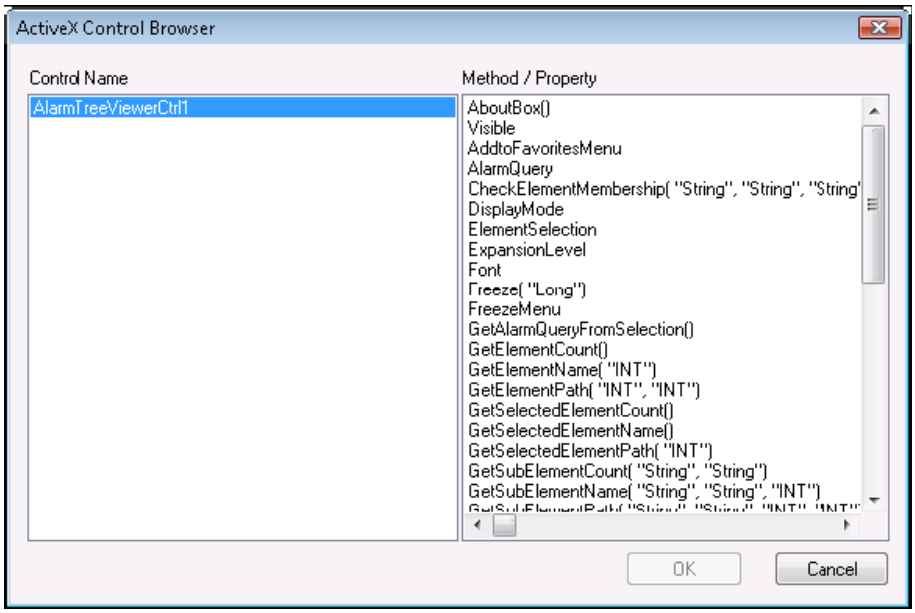
- Использовать свойство ActiveX компонента в скрипте или скрипте события ActiveX. Значение свойства считывается или записывается каждый раз, когда выполняется скрипт.
- Привязать свойство ActiveX компонента напрямую к тегу InTouch HMI. Значение свойства считывается или записывается каждый интервал обновления.

Конфигурирование скриптов для чтения и записи свойств ActiveX компонентов

В окне редактора скрипта можно сконфигурировать свойства ActiveX компонентов для чтения и записи значений из тегов или выражений InTouch HMI.

Для считывания данных и записи данных в свойство ActiveX компонента

1. Открыть окно редактора скрипта, в меню Insert выбрать ActiveX. Появится диалоговое окно ActiveX Control Browser.



2. Выбрать слева в панели имя ActiveX компонента. Справа в панели отобразятся имена свойств и методов, поддерживаемых ActiveX компонентом.
3. Выбрать имя свойства, которое необходимо использовать, из панели справа. Свойство вставится в скрипт на место курсора.
4. Назначить свойству тег или использовать в соответствии с Вашими требованиями.
5. Нажать ОК.

Пример

Данный скрипт считывает свойство ToProperty компонента ActiveX с именем AlarmViewerCtrl1, в тег topri.

```
topri = #AlarmViewerCtrl1.ToPriority;
```

Данный скрипт записывает значение MS Comic в свойство Font компонента ActiveX с именем AlarmViewerCtrl1. Данный пример динамически меняет шрифт ActiveX компонента AlarmViewerCtrl1.

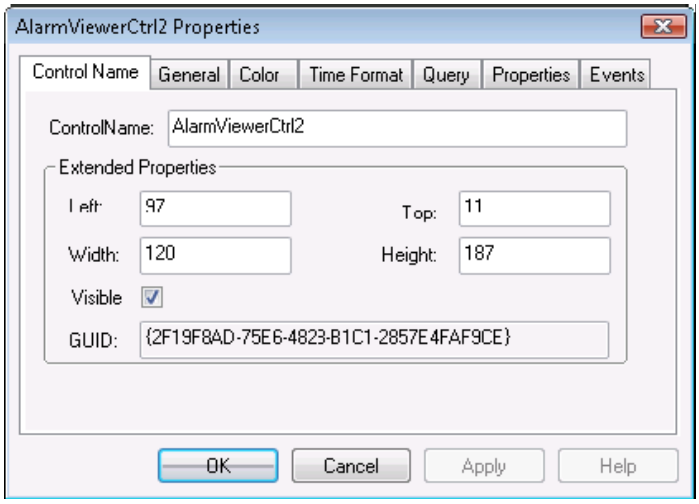
```
#AlarmViewerCtrl1.Font = “MS Comic”;
```

Подключение свойств ActiveX компонента к тегам

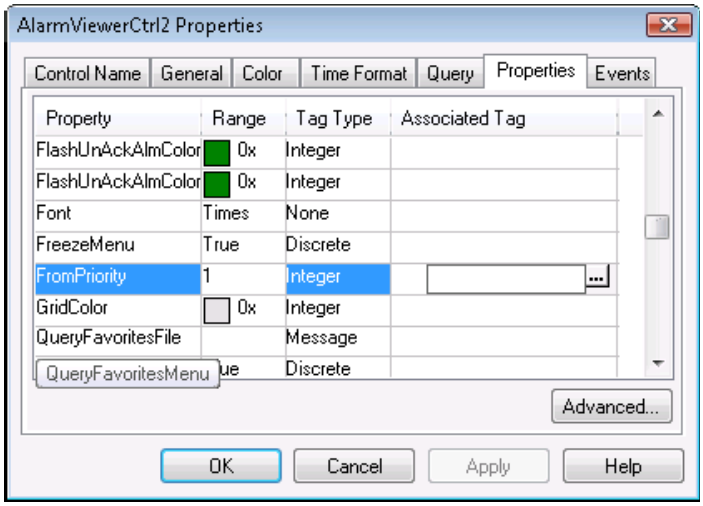
Можно подключить свойства ActiveX компонента к тегам InTouch HMI.

Для подключения свойств ActiveX компонента к тегам

1. Двойное нажатие мышкой на ActiveX элемент. Откроется диалоговое окно свойств.



2. Выбрать закладку свойств.
3. Выбрать свойство в списке.



4. Назначить тег. Сделать это можно следующим образом:
   - Ввести напрямую имя тега в колонке Associated Tag.
   - Нажать кнопку с ... в колонке Associated Tag между квадратными скобочками. Появится диалоговое окно Select Tag. Выбрать тег и нажать тег.
5. Нажать OK.

Создание многократно используемых скриптов событий ActiveX

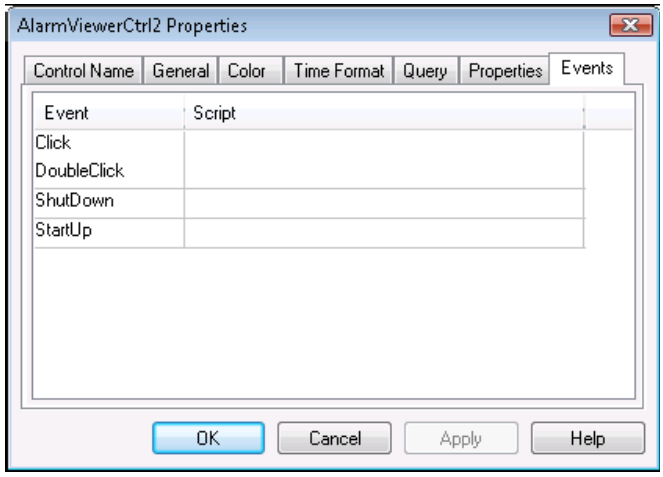
ActiveX компоненты поддерживают обработку событий, таких как однократное нажатие на компонент, с которым можно связать определенное действие. Эти действия хранятся в скриптах событий ActiveX.

Создание скриптов событий ActiveX

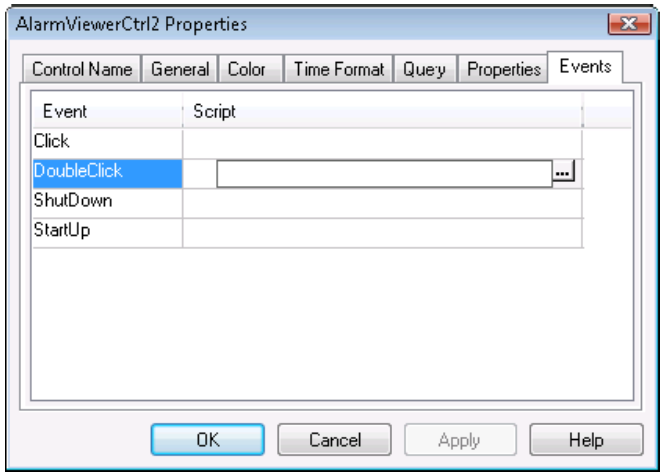
Можно создать использующийся многократно скрипт события, который выполняется каждый раз когда происходит определенное событие ActiveX компонента.

Для создания скрипта события ActiveX

1. Двойное нажатие мышкой на ActiveX компонент. Откроется диалоговое окно свойств.
2. Нажать на закладку Events (События)



3. Выбрать необходимое событие, к которому необходимо привязать скрипт, и нажать в соответствующем пустом поле колонки Script.



4. Ввести новое имя и нажать OK. Когда появится сообщение, нажать OK. Появится диалоговое окно ActiveX Events Scripts.
5. Написать скрипт.

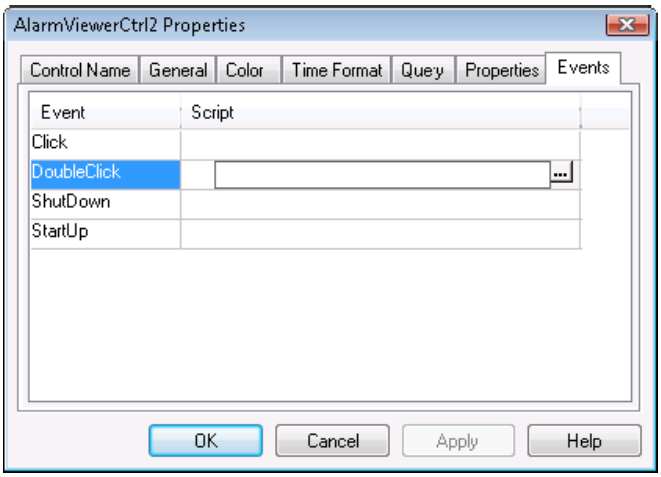
Повторное использование скриптов ActiveX событий

Можно использовать повторно скрипты событий ActiveX, если они были созданы тем же ActiveX компонентом и событием.

Например, в приложении имеется несколько ActiveX компонентов AlarmViewer, и они могут использовать совместно скрипт событий для события Double Click (двойное нажатие).

Для того чтобы использовать скрипт событий повторно:

1. Двойное нажатие на ActiveX компонент. Появится диалоговое окно со свойствами.
2. Выбрать закладку Events (События)
3. Выбрать необходимое событие, к которому необходимо привязать скрипт, и нажать в соответствующем пустом поле колонки Script.



4. Нажать на кнопку с точками. Появится диалоговое окно Choose ActiveX Script (Выбрать ActiveX скрипт)
5. Выбрать ActiveX скрипт и нажать OK.
6. Нажать OK.

Создание самоссылающихся скриптов ActiveX событий

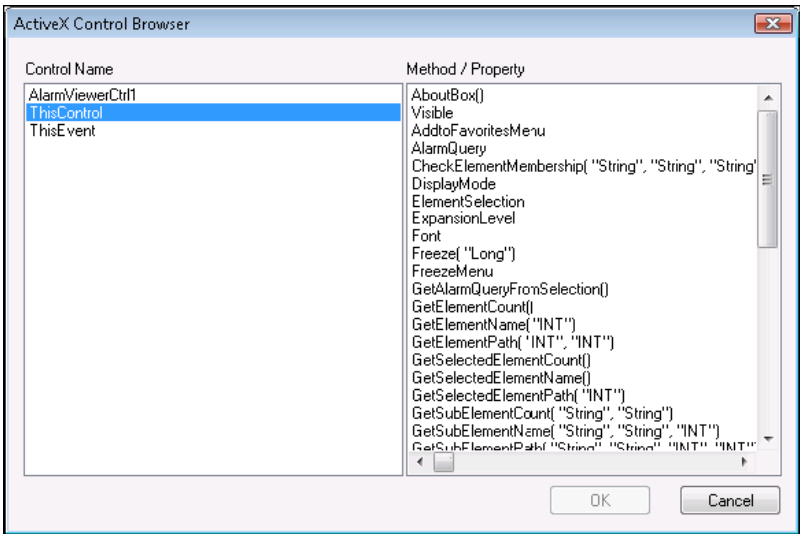
При использовании ActiveX скриптов событий, можно сконфигурировать их для ссылки на себя, вместо абсолютного имени ActiveX компонента. Это полезно, при создании ActiveX скрипта события, который будет использоваться повторно. Скрипты ActiveX событий могут одно из двух:

- Ссылаться на определенный ActiveX компонент, который генерирует событие (ThisControl)
- Ссылаться на определенное событие, вызывающее скрипт (ThisEvent).

Ссылка на определенное событие позволяет ActiveX компоненту передавать другие параметры в скрипт ActiveX компонента.

Для создания самоссылающегося скрипта ActiveX события

1. Создать скрипт ActiveX события для определенного ActiveX компонента. Смотрите раздел Создание скриптов ActiveX событий.
2. Диалоговом окне ActiveX Event Scripts, выбрать Insert и затем ActiveX. Появится диалоговое окно ActiveX Control Browser.



3. В левой панели, сделать одно из следующих:
   - Нажать ThisControl, для просмотра свойств и методов, которые можно использовать в связи с данным компонентом (и любого другого компонента, для которого будет использован повторно данный скрипт)
   - Нажать ThisEvent для просмотра свойств и методов ActiveX компонента, которые можно использовать в связи с самоссылающимся событием.

4. В правой панели выбрать свойство или метод и нажать OK. ActiveX компонент копируется в окно скрипта.
5. Конфигурировать скрипт
6. Нажать OK.

Пример:

```
ThisEvent.ClicknRow;
```

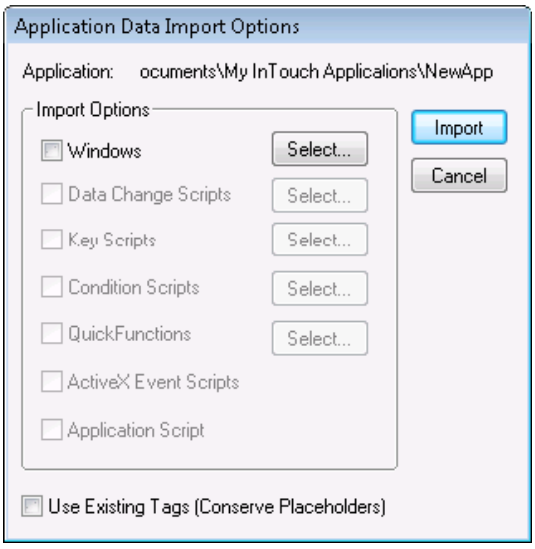
Данный скрипт возвращает номер строки нажатой для ActiveX компонента, вызывающего скрипт.

Импортирование скриптов ActiveX событий

Можно импортировать скрипты ActiveX событий из других приложений InTouch HMI, чтобы использовать их в текущем разрабатываемом приложении.

Для импортирования скриптов ActiveX событий из других приложений

1. В меню File, выбрать Import. Появится диалоговое окно Import from directory.
2. Отрыть каталог с приложением InTouch HMI, которое содержит скрипты ActiveX событий, которые необходимо импортировать.
3. Нажать OK. Появится диалоговое окно Application Data Import Options.



4. Выбрать ActiveX Event Scripts и нажать Import. Все скрипты ActiveX событий, импортируются в текущее приложение InTouch HMI.

Глава 9

Поиск и выявление ошибок в скриптах

Для поиска и выявления ошибок в скриптах можно использовать Log Viewer для отображения текущих значений тегов.

Сохранение сообщений в Log Viewer

Можно использовать Archestra Log Viewer для помощи при поиске ошибок в скриптах. Archestra Log Viewer располагается в ArchestrA System Management Console (SMC) и устанавливается при установке InTouch HMI.

Один из способов отладки скриптов:

1. Установить контрольные точки в скриптах для записи значений в Log Viewer.
2. Открыть Log Viewer для просмотра значений.

Другим путем является создание скрипта по нажатию кнопки, который будет сохранять значения тегов в Log Viewer.

Для установки контрольных точек в скриптах

1. Открыть скрипт, в котором, возможно, происходят ошибки
2. Определить строку, в которой необходимо установить контрольную точку.
3. Вставить один из следующих фрагментов кода в строку:

   · LogMessage(messagetag);

   В данном скрипте, messagetag – это имя строкового тега, значение которого требуется сохранять в Log Viewer.

   · LogMessage(StringFromIntg(inttag,10));

   В данном скрипте, inttag – это имя целочисленного тега, значение которого требуется сохранять в Log Viewer.

   · LogMessage(Text(realtag,"#.#######"));

   В данном скрипте, realtag – это имя вещественного тега, значение которого требуется сохранять в Log Viewer.

   · LogMessage(DText(disctag,"TRUE","FALSE"));

   В данном скрипте, disctag – это имя дискретного тега, значение которого требуется сохранять в Log Viewer.

   · LogMessage("DEBUG tag:"+ind.name+"value:"+Text(ind,"#.####"));

   Сохранения другой информации, такой как идентификатор и/или тег. Где ind – может быть аналоговым или индиректным тегом.

Функция LogMessage()

Запись определенного пользователем сообщения в Archestra Log Viewer.

Категория

Разное

Синтаксис

```
LogMessage(“Message_Tag”);
```

Параметр

Message_Tag

Строка, которая будет записываться в Log Viewer. Фактическая строка или строковый тег.

Примечание

Мощное средство для поиска и выявления ошибок в скриптах InTouch HMI. Если правильно расположить функции LogMessage() в скрипте, можно определить порядок выполнения скрипта, производительность скрипта, и идентифицировать значения тегов до того как они изменились и после того как они были обработаны скриптом. Каждое сообщение регистрируется в Log Viewer с точной меткой даты и времени.

Примеры

```
LogMessage("Report Script is Running");
```

Данная инструкция будет регистрировать в Log Viewer следующее.

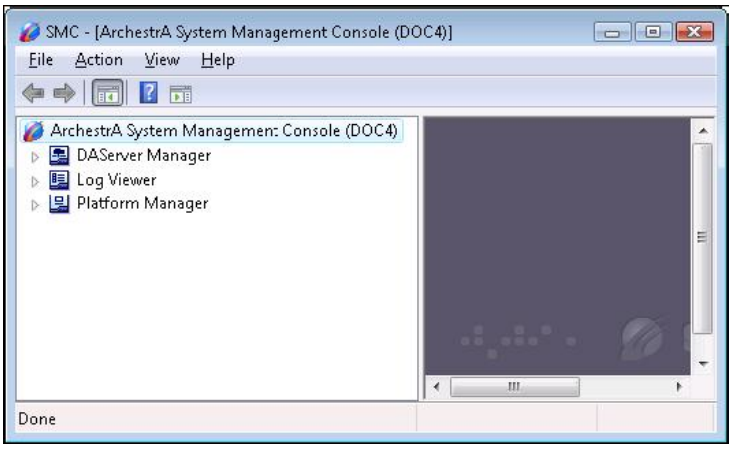
94/01/14 15:21:14 WWSCRIPT Message:Report Script is Running.

```
LogMessage(“The Value of MyTag is “ + Text(MyTag,“#”));
MyTag = MyTag + 10;
LogMessage(“The Value of MyTag is “ + Text(MyTag,“#”));
```

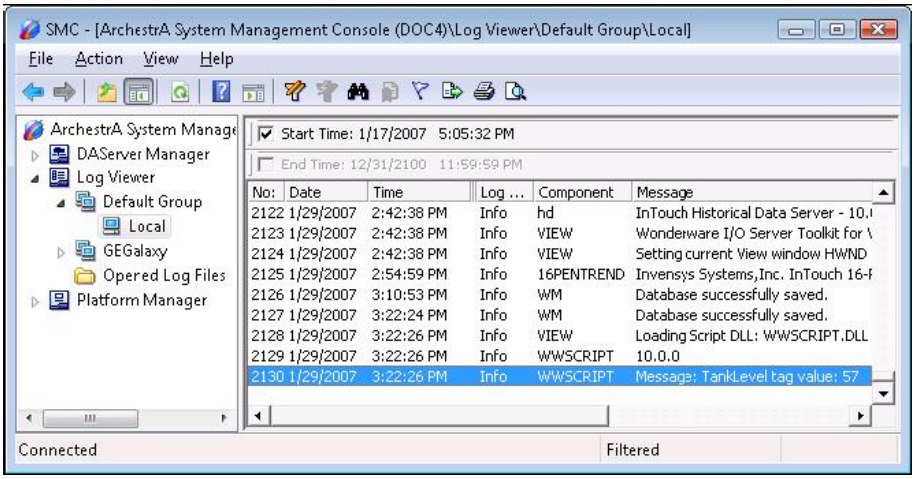
Просмотр сообщений в Log Viewer

Для просмотра сообщений в Log Viewer

1. Нажать Start, выбрать Programs, затем Wonderware, и нажать ArchestrA System Management Console. Появится окно с ArchestrA System Management Console.



2. В левой панели раскрыть Log Viewer, а затем Default Group, и выбрать Local. В панели справа отобразятся сообщения.



3. Найти сообщения записанные при помощи функции LogMessage().

Примечание Если производится отладка скрипта приложения InTouch HMI на удаленном узле, необходимо добавить имя узла сети в группу Node Group в Log Viewer и просмотреть сообщения для данного узла.

InTouch HMI Руководство по разработке скриптов и логике

Статистика изменений

Апрель 2008	Версия 1.0	Первая версия

KLINKMANN

www.klinkmann.com

Helsinki
ph. +358 9 540 4940
automation@klinkmann.fi

Санкт-Петербург
тел. +7 812 327 3752
klinkmann@klinkmann.spb.ru

Москва
тел. +7 495 641 16 16
moscow@klinkmann.spb.ru

Екатеринбург
тел. +7 343 376 53 93
yekaterinburg@klinkmann.spb.ru

Самара
тел. +7 846 993 49 33
samara@klinkmann.spb.ru

Київ
тел. +38044 495-33-40
klinkmann@klinkmann.kiev.ua

Мінск
тел. +375 17 2000876
minsk@klinkmann.com

Rīga
tel. +371 738 1617
klinkmann@klinkmann.lv

Tallinn
tel. + 372 6 684 500
klinkmann.est@klinkmann.ee

Vilnius
tel. +370 5 215 1646
post@klinkmann.lt