MortScript V4.1
(c) Mirko Schenk
mort@sto-helit.de
http://www.sto-helit.de

1 Что такое MortScript? / Лицензия

MortScript – это "просто" интепретатор (типа Visual Basic), невидимая программа, которая работает самостоятельно. (За исключением регистрации файловых расширений в момент запуска MortScript.exe, но в этом нет необходимости, если Вы его инсталлировали – см. 3 Инсталляция)
Язык скриптов предназначен для пакетного управления, т.е. для запуска других приложений и удаленного управления ими, и для выполнения простейших системных операций, типа файловых операций, внесения изменений в реестр, и т.д. Доступны также соответствующие простые диалоги.
Для работы используется загрузка скриптовых файлов с расширениями .mscr или .mortrun, которые можно создавать с помощью любого текстового редактора (см. 4.1 Create and execute scripts). Для начинающих написание собственных скриптов может вызвать трудности.
Вы можете выполнять эти скрипты запустив их как и любое другое приложение в файловом проводнике (т.е. просто нажав на имя файла), или создав ярлык этого файла в вашем стартовом меню (\Windows\Start menu, или в любой другой папке) с помощью файлового проводника (или любого другого инструмента).
Нет никакой гарантии от нарушений в работе устройства, вызванных этой программой (даже авторский скрипт не может быть безупречным). Надо понимать, что чужие скрипты могут делать много рискованных вещей точно так же как любое "нормальное" приложение может прочитать или удалить файлы, или отправить данные в Интернет.
MortScript бесплатен в использовании, т.е. Вам не надо платить за него, но буду рад небольшой благодарности типа "спасибо!". См.также 11 Donations.
Исходник доступен по запросу (SubVersion repository), но в отличие от GPL, Вы не можете создавать ваш собственный дериватив (вариант программы) – это является нарушением “For that script, you've got to use the XxxScript interpreter from Yyy”.
Вы можете использовать MortScript с вашими собственными скриптами, даже в коммерческих целях (только учтите, что ваши коды будут в открытом доступе, т.к. они не оттранслированы в машинный код…). Также учтите, что работа скриптов зависит от типа устройства. Ссылка на мой сайт или упоминание моего имени является хорошей манерой (например, „MortScript is freeware, www.sto-helit.de“).

2 Функциональность

MortScript включает в том числе:
- Запуск, активация, скрытие и закрытие приложений
- Функции ожидания: с фиксированным временем, ожидание существующего или активного окна
- Имитация нажатия клавиш и щелчков мыши в окне
- Копирование, переименование, удаление файлов, создание ярлыков
- Создание и удаление папок (директорий)
- Поддержка ZIP архивов (без перезаписи вложенных файлов!)
- Чтение и запись текстовых файлов
- Просмотр и модификация реестра
- Интернет: Чтение текстовых файлов, загрузка, создание и закрытие соединений
- Управляющие структуры по условию, выбору и циклы For-, ForEach, While или Repeat
- Некоторые системные функции (например, поворот экрана, регулировка громкости, фоновой подсветки, мягкий перезапуск)
- Подпрограммы, локальные переменные, многоуровневые массивы…

3 Инсталляция

3.1 Различные варианты MortScript

MortScript доступен для PC, PocketPC, Смартфонов (с Windows Mobile) и PNA (Навигауионные системы, основанные на Windows Mobile). Функциональный диапазон зависит от возможностей устройств. Если функция не существует в каком-то варианте, то это оговаривается в ее описании в этом руководстве. Также возможна проверка используемого варианта MortScript (см. 9.27.2 Получение типа MortScript (MortScriptType)).
Инсталляционный файл содержит все варианты. Вы можете выбрать одну, наиболее подходящую для Вас.
Аббревиатуры систем:
PC = PC (Windows XP/Vista)
PPC = PocketPC
SP = Смартфон

PNA = Навигационное устройство

3.2 Установка на PC

Просто запустите MortScript-4.1-*system*.exe (например MortScript-4.1-PPC.exe) из директории “exe” архива на вашем настольном ПК и следуйте указаниям…
В настоящее время нет инсталлятора для настольного ПК, пожалуйста см. “Binaries” об этом…

3.3 CAB файл

Этот тип инсталляции доступен только на Windows Mobile, т.к. нет CAB файла для версии PC.
Скопируйте MortScript-4.0-*system*.cab из директории архива “cab” в архив устройства (используя “Browse” в ActiveSync или карту памяти) и откройте его с помощью файлового проводника в устройстве (или любого альтернативного файлового менеджера типа TotalCommander, Resco Explorer, и т.п.).

3.4 Двоичные файлы

Скопируйте файлы, содержащиеся в архивной поддиректории “bin” с именем, соответствующим типу устройства (например “bin/PPC”) в любое место на устройстве (например в “\Program files\MortScript”). Запустите MortScript.exe для регистрации файловых расширений .mscr и .mortrun.

4 Использование

4.1 Создание и исполнение скриптов

MortScript выполняет файлы с расширениями ”.mscr” и ”.mortrun”.
Это необходимо для обратной совместимости, т.к. раньше программа называлась "MortRunner".
Такие файлы могут быть созданы с помощью любого текстового редактора. Вы даже можете использовать PocketWord,, но используя опцию "Save as - Text" ("Сохранить как – Обычный текст (*.txt)") и впоследствии изменив расширение .txt на .mscr или .mortrun. Если Ваш редактор поддерживает несколько кодировок, то используйте “ANSI”. Начиная с V4.1 возможно также использование кодировки Unicode с соответствующими префиксами (см. 9.13.1 Reading a text file (ReadFile)) но ANSI рекомендуется для обратной соместимости.
Если файл открыть, например, нажав на его имя в файловом проводнике, строки файла начнут последовательно исполняться, также как и в пакетном файле.
Вы можете создать ярлык на открываемый файл в меню Пуск->Программы или в папке автозапуска. Вы можете сделать это в файловом проводнике путем «Копирования» файла с последующей «Вставкой ярлыка» в ”\Windows\Start Menu” или ”\Windows\StartUp” (могут иметь другие названия в зависимости от локализации устройства).

4.2 Параметры MortScript.exe

Параметром для MortScript.exe является выполняемый скриптовый файл, с указанием полного пути. Если путь содержит пробелы, то он должен быть заключен в кавычки (например, "Storage Card\myscript.mscr").
Вариант для PPC имеет дополнительный параметр /wait=*n*, где *n* – количество секунд, в течении которых MortScript ожидает начало работы указанного скрипта. Этот параметр полезен, т.к. карта памяти становится доступной только после выхода устройства из режима standby. Таким образом, если вы назначили скрипт на соответствующую клавишу и будите устройство с ее помощью, то скрипт может быстро не запуститься. По умолчанию MortScript ждет начала выполнения скрипта 5 секунд.
Кроме того все параметры в формате “name=value” устанавливаются как переменные для использования в скрипте.
Например, если в качестве параметра указать “test="This is a test"”, то команда “Message(test)” в скрипте выведет на экран сообщение “This is a test”.
Эти переменные являются глобальными, поэтому, если Вы используете в скрипте локальные переменные, то Вы должны исключить их из Global(...). См. также 7.5.2 Variable scope.

4.3 Многозадачный запуск скриптов и их прерывание

MortScript может быть запущен в многозадачном режиме (с несколькими скриптами одновременно), но только с одной задачей для каждого скрипта.
Если запускается второй экземпляр уже запущенного скрипта, то открывается диалоговое окно выбора скрипта (например, Choice, Message, ...) для активации. Если скрипт не выводит какое-либо окно, то опасаться нечего.
Если Вы хотите прекратить работу скрипта, то Вы можете использовать ScriptProcExists и KillScript. Смотрите также информацию в 9.21.7 Завершение работы запущенного скрипта (KillScript).

5 Дополнительные инструменты

5.1 Выполнение скрипта при вставленной или удаленной карте памяти

Autorun.exe позволяет использовать автозапуск Windows Mobile очень просто.
Если карта памяти вставлена или удалена, то Windows выполняет программу “autorun.exe” в папке “2577” (CE код для процессоров ARM) или “0” на карте памяти (Например, ”\Storage\2577\autorun.exe”).
Эта опция поддерживается не на всех устройствах. Например, я читал, что HP деактивировал эту опцию на iPAQ 2210. Это также не работает на PNAs and PCs, где autorun.exe используется как заглушка “dummy exe” (смотри ниже).
Если Вы скопируете autorun.exe, MortScript.exe также как и autorun.mscr и/или autoexit.mscr в эту папку, то скрипты autorun.mscr (после вставки) и autoexit.mscr (после удаления) будут выполняться (если соответствующие скрипты существуют).
Для обратной совместимости Вы можете также использовать autorun.mortrun and autoexit.mortrun. Если существуют и .mscr и .mortru, то используйте .mscr.

5.2 “Dummy exe” для скриптов

Если переименовать autorun.exe, то он будет выполняться подобно скрипту. Т.е., если autorun.exe, переименовать, например, в myscript.exe, то будет выполняться myscript.mscr, или , если такого нет, то myscript.mortrun.
Переименованный autorun.exe и скрипт должны находиться в одной директории.
Если MortScript.exe также располагается в этой директории, то запуск скрипта на выполнение не приведет к обычным для этого скрипта результатам (т.е. произойдет инсталляция MortScript – или вы получите сообщение об ошибке инсталляции…)
Это особенность управления для программ, которые могут запускать другие программы, но не позволяют выбирать файлы с расширением .mscr, вроде некоторых инструментов телефонных профилей для устройств PhoneEdition.

5.3 Поддержка скриптов для CAB инсталляций (setup.dll)

setup.dll доступен только для PocketPCs. Он позволяет выполнять скрипты автоматически после инсталляции или перед деинсталляцией при использовании CAB инсталляции. Таким образом сохраняются … при создании собственного setup DLL. Если CAB файлы для вас закрытая книга, то просто пропустите эту часть.
Для активации setup.dll необходимо определить setup.dll в CAB файле. Если Вы используете CAB wizard от Microsoft, то это можно сделать с помощью строки CESetupDLL = "setup.dll" в .inf файле, Другие инструменты для этого предлагают соответствующие меню или опции.
MortScript.exe и, если вы используете ZIP архивы – mortzip.dll должен быть инсталлирован в директории для инсталляции по умолчанию (%InstallDir%) для инсталляции CAB. Это же относится и к install.mscr (запускается после инсталляции) and uninstall.mscr (запускается перед деинсталляцией), в результате чего оба могут быть опущены (не указываться). Но, конечно, это необходимо только для их настройки, …

6 Важная общая информация

6.1 Глоссарий

Константа: Фиксированное значение, т.е. число типа 100 или строка типа "Test"
Переменная: строка, отождествляемая с соответствующим значением

Например, “x = 5” (“5” соответствует переменной“x”) или Message(x)” (Значение “5”, которое соответствует переменной “x”, будет отображено).
Выражение: Комбинация переменных, констант, функций (см.ниже) и операторов, результатом которой является единственное значение (например, “5*x” или ”"Script type: " & ScriptType()”)
Присвоение: Установка значения переменной, обычно с помощью “имя переменной = выражение“
Параметр: Результирующее выражение, которое передается командам или функциям
Команда: Инструкция, которая не возвращает значение, например, MouseClick or Message
Функция: Инструкция, которая возвращает значение, например, SubStr. Функции используется в выражениях, также могут быть использованы в присваивающих выражениях и параметрах.
Управляющая структура: Инструкции, которые могут модифицироваться в зависимости от выполнения условий, типа If, Choice, ...

6.2 Синтаксический стиль, используемый в этом руководстве

Стиль этого руководства (система обозначений) основан на расширенной нормальной форме Бэкуса (РНФБ):
жирный шрифт: Фиксированное значение, например, наименование команды или функции
наклонный шрифт: Имя переменной, обычно любое выражение
[...]: Необязательно, может быть опущено (обычно, в этом случае используются значения по умолчанию)
{...}: Возможны повторения или опущено
x|y|z: Либо x, либо y, либо z должно быть указано (обычно это фиксированные значения).
(...): Группирование (обычно необязательные "|" для ясности).Если символы выделены жирным шрифтом, то они должны быть выделены, например, скобками **((...))**.
Примеры использования синтаксиса
Command [**(** *Expression* {**,** Expression } **)**]
or
Variable = *Function***(** [*Expression* {**,** *Expression* }] **)**
whereby an single function call is just a special type of an expression. For more about expressions, see 7 Possible parameters and assignments.
При использовании одной или нескольких команд, которые не нуждаются в параметрах круглые скобки после нее (них) необязательны, т.е. их наличие зависит от Вас, например, „RedrawToday“ или „RedrawToday()” равнозначно.
Но это не приведет к вызову функции! (Т.к. в выражениях круглые скобки применяются по умолчанию при вызове функции, иначе, без скобок имя функции может быть принято за имя переменной.)

6.3 Пробелы, табуляторы, и разделители строк

Пробелы и табуляторы допускаются в любом месте до, после и между элементами (т.е. вокруг имен команд/функций, круглых скобок, значений, операторов…). Такие строки должны быть заключены в кавычки, иначе часть строки будет игнорирована.
Разделители строк внутри инструкции допустимы, если вы введете ”\” в конце строки, которая будет продолжена. Также это можно сделать внутри строки. Но все окружающие пробелы, табуляторы и разделители строк будут заменены одиночным пробелом.

Пример:
```
Message( "This is \
a test" )
```
будет отображено „This is a test“.

Line breaks in a string must be entered as ^NL^, see 7.3 Fixed strings.
Разделители строк в строке должны вводится как ^NL^, см. 7.3 Fixed strings.

6.4 Чувствительность к регистру символов

Команды и имена файлов не чувствительны к регистру символов, но не в заголовках окон. Но вы можете использовать часть заголовка окна. Т.е. Show("WORD") не будет выполнено, а Show("Word") активирует "Pocket Word" (или первое окно со словом "Word" в заголовке…).

6.5 Директории и файлы

Имена директорий и файлов всегда должны вводиться с указанием полного пути (например, "\path\to\file.ext" или "\some\directory"), т.к. в Windows Mobile нет понятия “текущая директория“ – и на настольном компьютере тоже может привести к неожиданному результату.
Если путь не указан, то MortScript добавит текущую директорию, из которой запущен скрипт, но только в случае, если MortScript модифицирует содержимое файла, но не для системных операций. Т.е. команды ReadFile, WriteFile, ReadIni, WriteIni, ForEach с IniSections или IniKeys, и CallScript(Function) будут работать с относительными путями, команды типа Copy, Move, Rename, all ZIP operations, нет.
Или, короче: Все то, что необходимо для конфигурирования вашего скрипта должно быть доступно в директории скрипта, все остальное должно приводиться с указанием полного пути.
Windows Mobile не поддерживает “.” и “..” в указании пути, даже что-то вроде “\some\path\..\file.txt” (что может быть результатом работы скрипта по комбинированию пути, или при использовании SystemPath и добавлении его к пути), т.е Вы не можете получить доступ к файлам родительской директории таким способом.

6.6 Комментарии

Строка комментариев должна начинаться с символа “#”. Пробелы перед “#” допускаются.
В INI файлах комментарии допускается начинать с символа “;” в начале строки (как и обычно в INI файлах…).

7 Поддерживаемые параметры и соглашения

7.1 Выражения

Все параметры функций и команд, управляющих структур и значения присваиваются в выражениях с использованием ”=”. (Исключение: Старый синтаксис, см. 10 Старые синтаксис и команды
Выражения состоят из следующих частей:
Фиксированная строка в кавычках, например, "Text"
Фиксированное число, например, 42
Переменные, например x
Функции, например, SubStr(*parameter*)
Операторы, например, +, -, &, ..., которые указывают какие значения (константы, содержимое переменных, результаты выполнения функций) и как будут скомбинированы.
Это все звучит несколько запутанно. Вот несколько пояснительных примеров:

Message("Hallo!")
- ➔ Команда “Message” вызывается с фиксированной строкой “Hallo!” в качестве параметра

Sleep(500)
- ➔ Команда “Sleep” вызывается с фиксированным числом “500” в качестве параметра

Sleep(pause * 100)
- ➔ В этом случае содержимое переменной “pause” объединяется с фиксированным числом “100” с помощью оператора ”*” (умножение).

Message("Battery level: " & BatteryPercentage() & "%")
- ➔ Конкатенация двух фиксированных строк с результатом функции “BatteryPercentage()” и передачи результата в качестве параметра команды “Message”.

message = "Battery level: " & BatteryPercentage() & "%"
- ➔ На выходе то же, но результат присваивается переменной “message”, а не передается в качестве параметра команде.

If (BatteryPercentage() > 20)
# инструкция (последовательность команд)

EndIf

➔ Выполнения последовательности команд при выполнении условия.

Более подробная информация о том, как пишутся константы и переменные и какие операторы при этом допустимы, в следующих частях руководства.

Допустимые функции описаны в in 9 Commands and functions.

7.2 Типы данных

MortScript не поддерживает “ввод с клавиатуры”, т.е. Вы не можете присвоить переменной “x” цифровое значение, если у нее строковый тип. Числа и строки хранятся отдельно но автоматически конвертируются при необходимости. Только в некоторых редких случаях тип текущих значений принимается во внимание, например в операциях сравнения.
Для цифровых операторов (“+” и подобных), значения, при необходимости, конвертируются в числа. Таким образом ““"5"+"10"” возвращает 15. Если строка не содержит корректного числа, то она преобразуется в “0” (zero).
В других случаях, когда используется оператор для работы с текстом (например, ”&” для конкатенации текста), числа конвертируются в строки, так „5 & 10“ возвращает „510“.
Аналогично обстоит дело и с параметрами. Обычно от типа параметра зависит использование его значения – как числа или как строки. Например, текст выводится в виде строки, а временной параметр должен быть числом.
Для параметров условных операторов и операторов “on/off” существует следующее правило: Если значение представлено в виде числа отличного от “0”, то это означает, что “условие выполнено” и, соответственно, “on”, в противном случае “не выполнено” / “off”.
Т.е. выражения типа 5, "10", 1=1 и т.п. соответствуют “true/on”, в то время как 0, "x", 2=1 соответствуют “false/off”.
Если строка содержит десятичные точки, то она преобразуется в плавающую величину, а если ожидается целое, то округляется. Например, “SleepMessage("4.5", "Waiting...", "Wait") выполняет ожидание длительностью 5 сек. Если величина с плавающей точкой преобразуется в строку (например, для Message), она по умолчанию форматируется в 6 десятичных разрядов. Используйте функцию Format() вместо преобразования по умолчанию для получения лучшего результата.

7.3 Фиксированные строки

Фиксированные строки должны окружаться кавычками (").
Для использования кавычек внутри уже закавыченной строки их необходимо ставить дважды, например.

Message("He said: ""This is a test""")

➔ Will show „He said: "This is a test"”.

Следующие комбинации символов заменяют специальные символы:
^CR^ -> Возврат каретки (CR)
^LF^ -> Перевод строки (LF)
^NL^ □ Windows-/DOS line bread in files (New line, consists of CR+LF)
^NL^ ->перевод строки в файлах Windows-/DOS (спецсимвол «новая строка» (NL), состоит из CR+LF)
^TAB^ -> Табуляция

В Windows новая строка в текстовых файлах комбинируется из спецсимволов возврата каретки и перевода строки. Но иногда, а также в файлах Unix используется только ^LF^.

7.4 Фиксированные числа

Числа записываются как обычно (например x = **5.4321**, Sleep(**20**), ...).
Если вы используете операции с плавающей точкой, то надо использовать десятичную точку, типа “1.” вместо “1”.

7.5 Переменные

Переменные это «место» для хранения значений, которые ассоциируются с ними.
Все части выражения, которые не являются ни константами (например, 123 или "string"), ни операторами (+, -, &, ...), ни функциями, сопровождаемыми скобками, интерпретируются как имена переменных.
Допустимыми символами для имен переменных являются буквы A-Z (без умляут, ударений и т.п.) цифры и символ подчеркивания (”_”). Имена переменных не чувствительны к регистру букв, т.е. MYVARIABLE и myvariable является именем одной переменной.
Имя переменной не должно начинаться с цифры, т.к. эти цифры могут быть интерпретированы как цифровые константы в выражении (и все, следующее за цифрой является оператором или ошибкой). Поэтому “9mod2”, также как и “9 mod 2”, не является переменной.
Присвоение значения переменной обычно выполняется с использованием символа ”=”, например
myvar = 5 * x + y
Для этого используются также некоторые команды и управляющие структуры, которые присваивают значения переменным, типа GetTime или ForEach.
Для использования переменной в выражении Вы должны указать ее имя, типа “x” в примере выше.
Если Вы используете старый синтаксис, то помните о более запутанных правилах использования переменных (%...% и т.п..).

7.5.1 Встроенные переменные

Некоторые переменные являются встроенными для более удобного написания инструкций. В отличие от других языков их можно модифицировать, но делать это не рекомендуется:

TRUE, ON, YES инициализируются как 1,
FALSE, OFF, NO инициализируются как 0,
CANCEL инициализируется как 2.
PI инициализируется как 3.14159265358979323846264338327 95 (π)
SQRT2 инициализируется как 1.41421356237309504880168872420 97 (square root of 2)
PHI инициализируется как 1.61803398874989484820458683436 56 (φ)
EULER инициализируется как 2.71828182845904523536028747135 27 (*e*)

7.5.2 Области действия переменных

Обычно все переменные являются глобальными, что означает, если Вы назначили переменную, то ее значение может быть получено и модифицировано во всех подпрограммах (см. 8.8 Sub routines (Sub / Call)) и скриптах, вызываемых с помощью CallScript (см. 8.9 Other script as subroutine (CallScript)).
Если Вы хотите использовать локальные переменные, то Вам надо использовать одну из двух команд “Local” или “Global”.
Если Вы используете Local() без параметров, то все используемые переменные будут определены как локальные начиная с этой команды до конца подпрограммы или скрипта. Это относится также и к главной программе (код, располагающийся перед первой подпрограммой)
Вы можете также перечислить переменные в Local(), только те переменные, которые будут локальными до тех пор пока еще есть глобальные.
Global() работает наоборот: Переменные передаваемые этой команде остаются глобальными до тех пор пока существуют локальные. Это вроде списка исключений для Local().
Пример:

```
Local()
x = "Test"
Call( "Sub1" )
Call( "Sub2" )
Message( x ) покажет „Test“, т.к. локальная переменная не изменялась в подпрограмме
Message( y ) ничего не покажет, т.к. это не локальная переменная (а глобальная, см.ниже)
Sub Sub1
x = 5 will set the global variable
Local( x )
y = "Hi!" global variable! (only x is local)
Message( x ) ничего не покажет (локальная переменная не инициализирована!)
EndSub
```

Sub Sub2
Global(x)
Message(x) отобразит глобальное значение „5“
Message(y) ничего не отобразит, т.к. y не является локальной переменной
EndSub

7.5.3 Массивы (Списки)

Массивы являются специальным типом переменных. Массив состоит из некоторого количества связанных переменных, которые называются элементами.
Элемент адресуется с помощью имени переменной и так называемого «индекса» в скобках). Таким образом array[1] идентифицирует элемент «1» в массиве “array”.
Также возможно применение строки в качестве индекса, например colors["blue"], причем имя переменной не чувствительно к регистру символов. Т.е., COLORS["BLUE"] тождественно элементу colors["blue"].
Вы можете использовать любое выражение в качестве индекса. Это главное преимущество массивов, т.к. обычно доступ к элементам осуществляется как к индексированным переменным (например, с помощью некоторой переменной-счетчика).
Некоторые инструкции (типа Choice или Split) могут содержать в качестве параметров массивы. Но они обращаются к элементам начиная с 1. Меньшие значения индексов (<= 0) игнорируются.
Вы можете использовать дополнительные размерности массивов с помощью дополнительных индексов, заключенных в скобки, например “colors[x][y]”.

Примеры:

```
array[ "1" & 1 ] = "eleven"
Message( array[ (2-1) & "1" ] )
list[1] = "a"
list[2] = "b"
list[5] = "f"
list["a"] = "A"
idx = Choice( "Selection", "Choose something", 0, 0, list )
```
➔ Только “a” и “b” будут показаны в диалоге выбора.

7.5.4 Ссылки ([*variable name*])

Ссылки позволяют получать доступ к переменным (или элементам массива) внутри выражения.
Полезно рассматривать это как смесь переменной в виде безымянного массива и Eval() (см. 9.2.2 Выражения в строке (Eval)).
Для ссылки на переменную используется выражение, которое вычисляет имя переменной (в частности элемент массива) в скобках.
Например, “["array[1]"]” ссылается на первый элемент массива “array”. Конечно, это не совсем грамотно, т.к. аналогично использованию “array[1]”, но более громоздко. Но в случае “[arrayName & "[" & elem & "]"]” сразу видны преимущества более громоздкой записи.
Вы можете использовать эту особенность почти всегда за исключением команд старого синтаксиса без скобок (см. 10 Старые синтаксис и команды).

Примеры:

```
[targetVar] = [sourceVar] * 10
Choice( "Selection", "Select:", [choiceArrayName] )
GetTime( [varHour], [varMinute], [varSecond] )
```

7.6 Операторы

7.6.1 Перечень допустимых операторов

Все операторы в порядке приоритета (понижение сверху вниз):

()	Скобки
NOT	Отрицание
^	Степень (x^y → x^y)
* / MOD	Умножение, деление, модуль (остаток от деления)
+ -	Сложение, вычитание
& \	Конкатенация строк
> >= < <= = <>	Числовое сравнение
gt ge lt le eq ne	Посимвольное сравнение
AND &&	Двоичное / логическое “and”
OR ||	Двоичное / логическое “or”

7.6.2 Логические и двоичные операторы

Для логических операторов (true или false, т.е. &&, || и NOT) есть следующее правило: если значение представляет собой действительное число, за исключением “0”, то это означает ”условие выполнено”, соответствующее “on”, в противном случае ”не выполнено”/ “off”.
Т.е. выражения типа 5, "10", 1=1, и т.п. соответствуют “true/on”, а 0, "x", 2=1 соответственно “false/off”.
„NOT 5“ возвращает „0“ (аналогично not 0 = true т.к. false = 0), „NOT (2-2)” соответствует „1“ (2-2 = 0 = false, поэтому результат true = 1).
Различие между AND и &&, и, соответственно, между OR и || в том, что для && и || каждое значение, которое не равно 0 является 1 (например 2 соответствует “true”, или 1). Если же Вы используете эти операторы только для объединения результатов сравнения или результатов проверочных функций, то они равнофункциональны, т.к. они возвращают либо 1 (true), либо 0 (false).
Двоичные операторы AND и OR кроме того могут использоваться для поразрядных операций, например в выражении ”(x AND 4) = 4” проверяется значение 3-го бита (4 = 100 в двоичной системе счисления) в значении переменной “x”.
Логические операторы && and || сначала … thought for “C hackers”, которые используют тот факт, что 1 AND 2 не равно 0 (двоичное 01 AND 10 лает в результате 0), но 1 = “true”.

7.6.3 Сравнение

Операторы цифрового и символьного сравнения имеют одинаковый приоритет, они стоят в разных позициях перечня операторов только для удобства просмотра этого перечня.
С тех пор как MortScript не поддерживает ввод с клавиатуры, оператор сам принимает решение о методе сравнения, - цифровое, или строковое. Таким образом "123" < "20" возвращает “false” (потому что 20 меньше чем 123), но 123 < 20 возвращает “true” (потому что символ “1” меньше чем “2”, также как “a” меньше “b”).
Если Вы не помните операторов символьного сравнения, то вспомните аббревиатуры от „greater than“, „greater/equal“, „less than“, ”(not) equals“, и т.п.

7.6.4 Конкатенация строк и путей

”\” является оператором для конкатенации (объединения) путей. Только единичный символ ”\” является указанием на конкатенацию.
А вот символ ”&” служит для простой конкатенации строк, применение которого может привести к созданию неправильного пути

Примеры:

"\My documents\" \ "\file.txt"
"\My documents" \ "file.txt"
"\My documents\" \ "file.txt"
➔ Результатом будет "\My documents\file.txt".
Сравните со следующим:
"\My documents\" & "\file.txt"
➔ "\My documents\\file.txt"
"\My documents" & "file.txt"
➔ "\My documentsfile.txt"

"\My documents\" & "file.txt"
➔ "\My documents\file.txt" – Только теперь результат правилен!

8 Управляющие структуры

8.1 Условия

В качестве условия может быть использовано любое выражение в скобках. Условие выполнено, если его результат (при необходимости преобразованный в число) не 0 (zero к тому же является встроенной переменной как и FALSE и NO).
Допустимые функции перечислены в соответствующей категории "9 Commands and functions " Читайте также часть 7, в частности 7.2 Типы данных для дополнительной информации.

Примеры:
If (**wndExists("Word")**)
EndIf
While (**x <> 5**)
EndWhile

8.2 Простая передача управления (If)

If(*expression* **)**
{ *instructions* }
{ **ElseIf(** *expression* **)**
{ *instructions* } }
[**Else**
{ *instructions* }]
EndIf

Выполняются строки между If и Else или EndIf, если условие выполнено, или строки между Else и EndIf (если Else присутствует), в противном случае.
Если используется ElseIf, то только строки между **первым** выполняющимся условием (начиная с if) и ближайшим ElseIf, Else, соответствующим EndIf, выполняются.. Блок Else выполняется в случае невыполнения условия.(в случае его существования).
Операторы If, Else, ElseIf, и EndIf должны быть разнесены по разным строкам.

8.3 Переход по значению (Switch)

Switch(*expression* **)**
Case(*value* {, *value* } **)**
{ *instructions* }
{ **Case(** *value* {, *value* } **)**
{ *instructions* } }
EndSwitch

В зависимости от результата выражения выполняется блок, который находится после строки "Case" с величиной, равной этому результату.
Одинаковые величины могут находится в разных блоках Case (например, „Case(1, 2)" and „Case(2, 3)"). Выполняется тот «подходящий» блок, который встретится раньше.
"Проскакивание" или "прерывание", как в C, недопустимы, но наиболее реальная эмуляция подобных вещей возможна с помощью множественных "Case" с одинаковыми значениями.
Вы можете использовать разные типы для каждой величины, результаты переключающих выражений будут преобразованы в соответствии с типом сравниваемой величины.
Однако применение чисел с плавающей точкой имеют определенные ограничения по двум причинам: во-первых, в связи с проблемами округления. Два на вид одинаковых числа могут отличаться, например, в 15-м разряде. Во-вторых, если Вы используете что-то вроде "Case(2)", то это может соответствовать и 1.5 и 2.4, т.к. соответствуют целому числу 2 при преобразовании и округлении. Поэтому необходимо использовать "Case(2**.**)" для сравнения чисел с плавающей точкой.

8.4 Переход с диалогом выбора (Choice, ChoiceDefault)

(**Choice(** *title*, *hint*, *value*, *value* {, *value* } **)**
| **Choice(** *title*, *hint*, *array* **)**
| **ChoiceDefault(** *title*, *hint*, *default*, *timeout*, *value*, *value*
{, *value* } **)**
| **ChoiceDefault(** *title*, *hint*, *default*, *timeout*, *array* **)**
)
Case(*value* {, *value* } **)**
{ *instructions* }
{ **Case(** *value* {, *value* } **)**
{ *instructions* } }
EndChoice

Предлагает выбор из указанных значений. В "Case" Вы используете номер входа (начиная с 1). При нажатии “Cancel” или невыборе ни одного из значений возвращается 0.
За исключением этого работает аналогично “Switch”.
Теоретически Вы можете использовать и Switch(Choice(...)) (Choice как функция, 9.20.6 Выбор из списка (Choice)), но Choice в качестве управляющей структуры выглядит лучше.
ChoiceDefault является разновидностью, позволяющей установить выбор по умолчанию, который будет использован по истечении timeout. Если пользователь выберет другой вход, то отсчет timeout начнется сначала (для возможности изменить выбор).
Параметр *default* используется как индекс (т.е. “2” для 2-го входа (выбора)).
0 может использоваться в качестве выбора не по умолчанию (т.е. "Cancel", если пользователь не сделал выбор).
Параметр *timeout* указывается в секундах. Если 0, то таймаут не используется.

Пример:

```
Choice( "Test","Select a number","One","Two","Three" )
Case( 1 )
Message( "One" )
Case( 2, 3 )
Message( "Two or three" )
Case( 3 )
Message( "Three" )
Case( 0 )
Message( "Cancel" )
Exit
EndChoice
```

См.также 9.20.9 Установка размера и шрифта для выбранной записи (SetChoiceEntryFormat)

8.5 Условный цикл (While)

While(*condition* **)**
{ *instructions* }
EndWhile

Выполняются строки между While и EndWhile до тех пор, пока условие (*condition*) выполняется
While и EndWhile должны находиться в разных строках.

8.6 Итерации для нескольких значений (ForEach)

ForEach *variable*{, *variable* } **in** *type* **(** *parameter* {, *parameter* } **)**
{ *instructions* }
EndForEach

Это очень мощный инструмент. Используемые переменные (*variable*) определяют *type* и *parameter* для каждой итерации. Это можно применить для составления списка значений (типа “values”) для ключей и значений в секциях внутри INI файлов (“iniKeys”).
Пожалуйста, учтите: Если элемент массива используется как итерационная переменная, то ее индекс будет вычисляться только при входе в цикл. Это означает, что, если “i” равно 1 при входе в цикл ForEach, то “array[i]” будет соответствовать “array[1]” при каждой итерации, но, по существу. “i” соответствует другому значению в теле цикла.
Это относится и к параметрам: Их значения вычисляются в момент входа в цикл.
Во всем остальном Вы можете использовать выражения для всех параметров, также как и в большинстве многих других команд.
В настоящее время существуют следующие варианты:

8.6.1 Циклы по значениям данных (по списку выражений, по содержимому массива, по разделяемым строкам, по символам строки)

ForEach *variable* **in values (** *value* {, *value* } **)**

Установка нового значения (*value*) переменной (*variable*) в каждой итерации

ForEach *variable* **in array (** *array variable* **)**

Присвоение значения элемента массива переменной. Учитываются только элементы, начиная с элемента с индексным номером, равным 1. Более низкие значения, буквенные индексы и значения после изменения шага индекса игнорируются.

ForEach *key, value* **in array (** *array variable* **)**
Циклы по всем элементам массива с первой переменной (*key*) в качестве индекса элемента и второй переменной (*value*) в качестве значения элемента

ForEach *variable* **in split (** *string, separator, trim?* **)**

Аналогично функции Split (см. 9.5.5 Разбиение строки на несколько переменных/массив элементов (Split)), но отдельные части строки (*string*) назначаются переменной (*variable*) по очереди.

ForEach *variable* **in charsOf (** *string* **)**

Присвоение переменной (*variable*) каждого символа строки (*string*) по очереди. С тех пор, как MortScript автоматически преобразовывает типы, это работает и для цифр – например “4” и “2” (для 42).

8.6.2 Цикл по значениям INI файла (секции, значения в секции)

ForEach *variable* **in iniSections (** *file name* **)**

Возвращает отдельную секцию ("[Section]", без кавычек) в указанном INI файле.

ForEach *key, value* **in iniKeys (** *file name, section* **)**

Присваивает содержимое секции (*section*) в INI файле указанным переменным (*value*).
„Key“ – это имя параметра INI файла, которое находится в строке файла перед ”=”, „value“

8.6.3 Цикл по данным системного реестра (подключам, значениям ключей)

ForEach *variable* **in regSubkeys (** *root, key* **)**

Возвращает все дерево указанного ключа. Параметры см. в 9.19.1 Считывание записей из реестра (RegRead).

ForEach *value, data* **in regValues (** *root, key* **)**

Присваивает значения ключей реестра указанным переменным.
„value“ – это имя переменной, которое получается из имени значения (параметра) ключа реестра, *data* – текущее значение этого параметра. Дополнительно см. 9.19.1 Считывание записей из реестра (RegRead).
Этот цикл не возвращает ключи со значениями по умолчанию (которые обычно обозначаются как “(Default)” or “@” в редакторах реестра).

8.6.4 Цикл по файлам и директориям

ForEach *variable* **in files** (*search expression*)
ForEach *variable* **in directories** (*search expression*)

Присваивает найденные файлы и директории переменным.
Выражение должно состоять из пути и части имени файла с метасимволом *, например "\Program Files\Mort*" or "\Program Files\Test*.exe".

8.7 Фиксированное число повторений (Repeat)

Repeat (*count* **)**
{ *instructions* }
EndRepeat

Повторяет команды, находящиеся между этими двумя командами заданное (*count*) число раз. *Count* не может быть меньше 1.

8.8 Простая итерация (For)

For *variable* **=** *start* **to** *end* **[step** *step* **]**
{ *instructions* }
Next

На первой итерации *variable* присваивается значение *start*, затем она увеличивается (или уменьшается, если *step* отрицателен) на *step* при каждой последующей итерации, до тех пор пока не превысит *end* (значение *variable* больше, чем *end*, если *step* положителен, и меньше, если *step* отрицателен).
Если *step* опущен, то он принимается равным 1 если *end* больше, чем *start* и -1 если *end* меньше, чем *start*.
Это работает и для целых чисел и для чисел с плавающей точкой. Для чисел с плавающей точкой используется точность в 6 цифр при сравнении с величиной *end* (см. также 9.4.8 Сравнение чисел с плавающей запятой (CompareFloat)).
Пожалуйста учтите, что любые выражения (для *start, end,* or *step*) вычисляются только для первой итерации. Например, если Вы используете переменную для *end* или *step* (“For i = 1 to end step x”), и изменяете эти переменные во время итерации, то используемая в операторе цикла переменная (“i”) будет скрытно увеличена и вновь проверена на соответствие значению этой переменной перед “For”.

8.9 Подпрограммы (Sub, Call/CallFunction)

Sub *subroutine name*
{ *instructions* }
EndSub
Call(*subroutine name* {**,** *parameter* } **)**
CallFunction(*subroutine name, variable* {**,** *parameter* } **)**

“Call” вызывает подпрограмму (*subroutine*), которая записана в скрипте начиная со строки, следующей за “Sub” с тем же параметром (*subroutine name*). При достижении конца подпрограммы (EndSub) происходит переход на строку, следующую за строкой с “Call”.
В отличие от большинства других команд для “Sub” имя подпрограммы должно быть указано в скобках и выражения (для формирования имени подпрограммы) не допускаются.

Для Call/CallFunction, пожалуйста, не забывайте использовать кавычки вокруг имени подпрограммы, т.к. в качестве этого параметра может использоваться вычисляемое выражение, как и для всех других параметров. Вы можете даже использовать что-то вроде “Call(variableContainingMySubroutine)” также как и “On ... gosub ...” (для тех, кто помнит команды языка программирования BASIC), но это является плохим стилем программирования может создать проблемы если Вы забудете как называется подпрограмма.
Для Call может быть также удобен старый синтаксис (“Call SubFunction”), но только в том случае, если Вы не используете параметры (*parameter*).
Если Вы указываете параметры, то подпрограмма будет иметь две локальных переменных (см. 7.5.2 Области действия переменных): argc, содержащую количество передаваемых в подпрограмму параметров, и argv. – массив всех передаваемых параметров.
Так, если Вы передаете два параметра, то argc равно 2, argv[1] – первый параметр, и argv[2] – второй параметр.
Если Вы используете CallFunction, в которой есть инструкция “Return(value)”, то полученное значение (value) будет присвоено *variable*. Return() не допускается использовать в подпрограмме во многих других языках, эта инструкция используется только для возврата полученного внутри функции значения! Если Return() не используется внутри функции, то переменная *variable* будет пуста (см. 9.2.4 Проверка существования переменной (IsEmpty))
Подпрограммы должны находиться после главной программы, MortScript выходит на первую “Sub” неожиданно.

8.10 Другие скрипты как подпрограммы (CallScript/CallScriptFunction)

CallScript(*MortScript file* {**,** *parameter* } **)**
CallScriptFunction(*MortScript file*, *variable* {**,** *parameter* } **)**

Выполняет другие скрипты как подпрограммы.
Это касается и переменных, т.е. это дает возможность изменять содержимое Local() или Global() переменных внутри вызывающего скрипта.
Обо всем, что касается параметров (argc и argv) и возвращаемого значения (Return()), смотри в описании подпрограмм, также смотри дополнительно 8.9 Подпрограммы (Sub, Call/CallFunction).
Также, пожалуйста, обратите внимание на 6.5 Директории и файлы.

Пример:
CallScript("subscript.mscr")

Для запуска других приложений или «внешнего» скрипта существуют специальные команды, см. 9.6 Выполнение приложений и открытие документов.

8.11 Установка возвращаемого значения (Return)

Return(value **)**

Возврат указанного значения в вызываемых CallFunction or CallScriptFunction.
В отличие от большинства других языков программирования эта инструкция не приводит к возврату в вызывающую программу, а только устанавливает возвращаемое значение! Допускается также использование массива переменных.
Смотри также 8.9 Подпрограммы (Sub, Call/CallFunction) и 8.10 Другие скрипты как подпрограммы (CallScript/CallScriptFunction).

8.12 Прерывание скрипта (Exit)

Exit

Завершает выполнение скрипта.

9 Команды и функции

Функции используются как *type* **= Function(** ... **)**.

Тип функции может быть либо строковый (*string*), либо булевый (*bool*), либо целый (*int*), либо плавающий (*float*), в зависимости от типа возвращаемого значения. *int* и *bool* означают целые числа (не дробные), но булева функция возвращает только TRUE (1) и FALSE (0).
Если возвращаемые значения могут быть различных типов, то тип функции *value*. (Обычно дополнительную информацию можно найти в описании функции).
Конечно, все функции могут быть также использованы в более сложных выражениях, чем простое присвоение типа “variable = ...” (смотри 7 Поддерживаемые параметры и соглашения)

9.1 Ошибки выполнения (ErrorLevel)

ErrorLevel (*error level* **)**

Указывает какие сообщения об ошибках отображать. Параметр *error level* является строковым (например, "syntax"), т.е. инструкция **not** “ErrorLevel(syntax)” – кроме "syntax", будет переменная, котрая содержит "syntax"...
По умолчанию „error“.
Это может быть изменено также и в процессе исполнения программы. Если *error level* равен "warn" or "error", то при наступлении ошибки (см.перечень ошибок ниже) выполнение программы будет прервано. Если же уровень равен "off" для "syntax", то ошибка будет игнорирована, что Вы можете проверить, например, с помощью “If (wndExists(...))”.

Допустимые уровни ошибок:
off No error messages

Скрипт может быть прерван без каких-либо сообщений

critical Critical messages

в настоящее время не используется, зарезервино на будущее

syntax Syntax errors

например неправильный параметр счетчика, или недопустимые команда или имя функции

error Other errors

например, не существующее окно, проблемы с записью или чтением ключей системного реестра, нового документа или с созданием новой директории,…

warn Warnings

например, если не удается удалить файл/директорию, не работает копирование/перемещение/переименование (такие файлы уже существуют)

Уровни включают все уровни, которые перечислены выше, т.е. при “error” сообщения с уровнями “syntax” и “critical” также будут отображаться.

9.2 Переменные

9.2.1 Присвоение значений (”=” и Set)

Variable = expression
Set(*variable, expression)*

Присваивает результат выражения переменной.
Set() может использоваться не всегда, лучше использовать *Variable = expression*.

Однако Set() имеет небольшую специфическую особенность. Если переменная заключена в символы % (%*Variable* %), то содержимое этой переменной используется как имя другой переменной. Если, например “varRef” содержит строку "var", и %varRef% выступает в роли используемой в выражении переменной, то результатом выражения будет “var”, а не “varRef”.
Если „varRef“ содержит строку заключенную в ”%”, то это будет работать как рекурсия до тех пор, пока будут существовать ссылки на переменные, заключенные в ”%” (или система рухнет в результате переполнения стека…).
Это обычно создает путаницу и частично устарело. Пожалуйста, по возможности используйте ссылки (см. 7.5.4 References ([variable name])).

9.2.2 Выражения в строке (Eval)

value = **Eval***(string)*

Вычисление выражения, содержащегося в строке, и возврат результата вычисления.

Например:
x = Eval("1+5*x")
возвращает x = 26, если x ранее было присвоено значение 5.

9.2.3 Удаление переменной или элемента массива (Clear)

Clear*(variable)*

Clear удаляет переменную или элемент массива. В отличие от обнуления переменной или элемента массива, при котором функция IsEmpty() возвращает TRUE (см.ниже), а элемент массива не работает в ForEach, Choice, и подобных операторах.

9.2.4 Проверка существования переменной (IsEmpty)

bool = **IsEmpty***(variable)*

Возвращает TRUE если переменная (или элемент массива) не определен как дальний (far) или удален с использованием Clear(), FALSE, - если переменная определена – даже если она содержит пустую строку.

9.2.5 Диапазон действия переменной (Local, Global)

Local*([variable {, variable }])*
Global*(variable {, variable })*

Local() определяет данную или все (не ссылочные) переменные как локальные, определенные в текущем блоке операторов (в подпрограмме, или в главной функции скрипта).
Global() работае с точностью до наоборот. Определяемые переменные будут доступны везде до тех пор, пока не будут переопределены как локальные во внутреннем блоке.
Подробнее см. «7.5.2 Variable scope»

9.3 Операции со строками

9.3.1 Получение длины строки (Length)

int = **Length***(string)*

Возвращает количество символов в строке.

Например:
x = Length("This is a test")
возвращает x = 14

9.3.2 Извлечение символьной подстроки из строки (SubStr)

string = **SubStr***(string, start [, length])*

Возвращает „length“ символов, начиная с символа номер „start“ от начала строки.
Если параметр length is опущен, или его величина больше количества символов от символа, определенного переменной start как начальный, до конца строки, то возвращается подстрока начиная с символа start и до конца строки.
Если строка короче, чем “start”, то возвращается пустая строка.
В параметре “start” можно использовать отрицательное значение. В этом случае отсчет идет от последнего символа в строке, т.е. при значении -2 отсчет будет идти от предпоследнего символа. При слишком большом значении абсолютной величины этого параметра (больше длины строки) отсчет будет вестись от начала строки.

Например:
x = SubStr("abcdef", 2, 3)
возвращает x = "bcd"
x = SubStr("asdf", -3)
возвращает x = "sdf"

9.3.3 Деление строки на части (Part)

string = **Part***(string, separator, index [, trim?])*

Делит строку в соответствии с указанным разделителем (параметр separator) и возвращает часть, определенную параметром index (например вторую, если index=2). Можно использовать также и отрицательное значение index, т.е. при -1 - возвращается последняя часть, при -2 – предпоследняя, и т.д.
Если параметр "trim?"=TRUE или опущен, то пробелы, окружающие часть строки будут опущены.

См. также раздел 9.5.3. «Деление строки на несколько переменных/элементов массива (Split)».

Например:
x = Part("a | b | c", "|", 2)
возвращает x = "b" (2 part, пробелы удалены)
x = Part("a\ b \ c.def", "\", -1, 0)
возвращает x = " c.def" (последняя часть, пробелы не удалены)
x = Part("one, two, three", ",", 4)
возвращает x = "" (пустая строка, т.к. 4-я часть не существует)

9.3.4 Поиск подстроки внутри строки (Find)

string = **Find***(string to check, string to search [, start])*

Возвращает позицию первого символа найденной подстроки.
Если указан параметр start, то поиск начинается с указанной позиции.
Если искомая подстрока внутри строки отсутствует, то возвращается 0. Необходимо отметить, что недопустимо использовать функции типа SubStr с параметром start, равным 0.

Например:
x = Find("abcdefcd", "cd", 5)
возвращает x = 7
x = Find("abcdef", "CD")
возвращает x = 0 (функция чувствительна к регистру!).

9.3.5 Поиск последнего вхождения символа (ReverseFind)

int = **ReverseFind***(string, character)*

Возвращает позицию последнего найденного символа в строке. В отличие от Find допускается только один символ.

Если символ не найден, то возвращается 0.

Например:
x = ReverseFind("abcba", "b")
возвращает x = 4

9.3.6 Замена подстроки внутри строки (Replace)

string = **Replace***(source, old, new)*

Замена подстроки “old” внутри строки “source” на подстроку “new”.

Например:
x = Replace("My old string", "old", "new")
возвращает x = "My new string"

9.3.7 Изменение регистра символов (ToUpper/ToLower)

string = **ToUpper***(string)*
string = **ToLower***(string)*

Возвращает строку с символами в верхнем регистре (ToUpper), либо в нижнем (ToLower).
Если переменная является параметром, то ее содержимое не модифицируется (в отличие от старых команд MakeUpper/MakeLower).
В зависимости от вашей системы и ее локализации “спецсимволы” типа “а” или “е” не конвертируются!

Например:
x = ToUpper("Abcba")
возвращает x = "ABCBA"
x = ToLower("AbcBA")
возвращает x = "abcba"

9.3.8 Части имени файла (FilePath, FileBase, FileExt)

string = **FilePath***(file with path)*
string = **FileBase***(file with path)*
string = **FileExt***(file with path)*

Эти функции позволяют разделить имя файла и путь к файлу.
FilePath возвращает путь без имени файла ("\My documents" для "\My documents\test.txt”)
FileBase возвращает имя файла без пути и расширения ("test" для "\My documents\test.txt”)
FileExt возвращает расширение файла ("txt" для "\My documents\test.txt”)
Может быть использовано совместно с 9.18.4 Получение системных путей (SystemPath).

9.4 Математические функции

9.4.1 Форматирование вывода (Format)

string = **Forma***t(value, decimals)*

Возвращает value как строку с указанным количеством знаков после десятичного разделителя. Значение может быть при необходимости округлено.

Например:
x = Format(123.456789, 2)

возвращает x = "123.46"
x = Format(12, 2)
возвращает x = "12.00"

9.4.2 Конвертирование в/из 16-ричных чисел (NumberToHex, HexToNumber)

string = **NumberToHex***(int value)*
int = **HexToNumber***(string)*

NumberToHex конвертирует целое значение (величина с плавающей запятой предварительно округляется) в строку шестнадцатеричной величины. Строка форматируется побайтно, т.е. каждая цифра преобразуется в символ (например "0100" для
256 или "0a" для 11).
HexToNumber работает наоборот. Конвертирует строку с шестнадцатеричным значением в целое значение. Работает до первого не шестнадцатеричного символа, например HexToNumber("axe") возвратит 10, т.к. первый символ "a" является 16-ричным символом, а "x" – нет. Работает также с символами в верхнем регистре ("ADE").

Учтите, что эти функции работают надежно только с величинами от 0 до 2 147 483 647 (7fffffff). Числа от 2 147 483 648 (80000000) до 4 294 967 295 (ffffffff) конвертируются правильно из десятичных в 16-ричные, но возвращают отрицательные значения при конвертации из 16-ричных в десятичные. Так при конвертации значений от -1 до 2 147 483 647 результат будет от ffffffff (-1) до 80000001 (-2 147 483 647). Это результат особенности внутреннего хранения значений (первый бит – «отрицательный» флаг).

Значения, выходящие за допустимый диапазон приводят к ошибкам или непредсказуемым результатам.

9.4.3 Округление (Round, Floor, Ceil)

int = **Round***(value)*
int = **Floor***(value)*
int = **Ceil***(value)*

Эти функции округляют value до целого значения.
Floor округляет вниз (2,9 -> 2), Ceil округляет вверх (2,1 -> 3), а Round округляет стандартным образом (2,49 -> 2,5 - > 3).

9.4.4 Генератор случайных чисел (Rand)

int = **Rand***(max)*
float = **Rand***()*

Если указан параметр max, то возвращается целое число в диапазоне от 0 до (max-1).
При отсутствии параметра возвращается число в диапазоне от 0 до 0,999…

9.4.5 Тригонометрические функции (Sin, Cos, Tan)

float = **Sin***(radians)*
float = ***Cos***(radians)
float = ***Tan***(radians)

Учтите, что эти функции используют в качестве параметра радианы! Если вам надо использовать градусы (типа 45^{o}), то воспользуйтесь формулой "degree * PI / 180".

9.4.6 Логарифмы (Log, Log10)

float = **Log***(value)*
float = **Log10***(value)*

Log рассчитывает натуральный логарифм по e (переменная MortScript EULERT), Log10 – десятичный логарифм (по 10).

9.4.7 Квадратный корень (Sqrt)

float = **Sqrt***(value)*

Рассчитывает квадратный корень из value. Корень из двух может быть получен из внутренней переменной SQRT2.

9.4.8 Сравнение чисел с плавающей запятой (CompareFloat)

int = **CompareFloat***(value1, value2, precision)*

Сравнение двух чисел с плавающей запятой не всегда может быть корректно из-за ошибок округления. Ноль не бывает в машинном представлении реальным нулем, но что-то около $1*10^{-20}$.
Небольшая помощь при использовании этой функции – округление с указанной точностью (precision) перед сравнением.
Возвращает -1, если value1 < value2, 1 если value1 > value2, and 0 если значения равны.

9.5 Массивы

9.5.1 Получение старшего индекса в массиве (MaxIndex)

int = **MaxIndex***(array)*

Возвращает максимальное числовое значение индекса массива начиная с 1.

Например:
array[1]="a"
array["2"]="b"
array[3]="c"
array[5]="e"
array["x"]="X"
max = MaxIndex[array]
возвращает 3.

MortScript перечисляет строку индексов с цифровым содержимым как цифровые индексы (точнее, говоря по другому, округляет значения, сохраняя их как строки), поэтому "2" учитывается пока существует "x"
Т.к. 4 отсутствует, то 5 игнорируется. То же самое было бы, если бы элемент array[4] существовал, но был бы очищен с помощью Clear().

9.5.2 Получение количества элементов (ElementCount)

int = **ElementCount***(array)*

В отличие от MaxIndex эта функция возвращает количество элементов массива включая и очищенные с помощью оператора Clear() (который только удаляет значение элемента, но не удаляет сам элемент.
For the array in the MaxIndex example, ElementCount would return 5.
Для массива, приведенного в примере для MaxIndex, ElementCount возвратит значение 5.

9.5.3 Создание массива из списка переменных (Array)

array = **Array***(value {, value })*

Возвращает массив с заданными значениями, проиндексированными начиная с 1.
Учтите, что MortScript не позволяет создавать массив типа “Array("a", "b", "c")[1]”, вы можете только создать массив переменных и, затем, получать доступ к элементам через эти переменные.

Например:
days = Array("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")
day = days[FormatTime("w")+1]

9.5.4 Создание массива с указанием индексов (Map)

array = **Map***(index, value {, index, value })*

Возвращает массив с заданными ключами и значениями.
Вы должны задать пару значений для каждого параметра, где первое значение является индексом элемента массива, а второе – величиной (значением) того же элемента.
Учтите, что MortScript не позволяет создавать массив типа “Array("a", "b", "c")[1]”, вы можете только создать массив переменных и, затем, получать доступ к элементам через эти переменные.

Например:
months = Map("01", "Jan", "02", "Feb", "03", "Mar", etc.)
month = months[FormatTime("m")]

9.5.5 Разбиение строки на несколько переменных/массив элементов (Split)

Split*(string, separator, trim?, variable {, variable })*
array = **Split***(string, separator [, trim?])*

Разделяет строку на части указанным разделителем. Если разделитель состоит из нескольких символов, то они должны все присутствовать в разделяемой строке.
Если вы используете команду с присвоением полученных частей нескольким переменным, то, если переменных больше получаемых частей, то лишние переменные получают пустые значения, если же частей больше, чем переменных, то лишние части будут проигнорированы.
Если в функции используется только одна переменная, то соответствующая переменная будет являться названием массива с индексом от 1 до n, где n – количество получаемых частей.
Если параметр "trim?" равен TRUE (или опущен), то пробелы, окружающие получаемые части будут удалены.

Например:
Split("a | b | c", "|", 1, a,b,c,d)
возвращает a="a", b="b", c="c", d=""
Split("a\ b \c.def", "\", 0, a, b)
возвращает a="a", b=" b "
Split("one, two, three", ",", 1, list)
возвращает list[1]="one", list[2]="two", list[3]="three"
list = Split("one, two, three", ",")
возвращает list[1]="one", list[2]="two", list[3]="three"

9.6 Выполнение приложений и открытие документов

9.6.1 Открытие приложений/документов и продолжение выполнения скрипта (Run)

Run(a*pplication* [, *parameter*] **)**
Запуск приложения. Скрипт продолжает выполняться до тех пор, пока приложение загружено и выполняется.
Ссылки (*.lnk), параметры и документы также допустимы.
Должен указываться полный путь к приложению.

Примеры:

Run("\Windows\StartMenu\Messages.lnk")
Run("\Windows\PWord.exe", "\My documents\doc.psw")

9.6.2 Открытие приложения/документа и ожидание завершения его работы (RunWait)

RunWait(*application* [, *parameter*] **)**

Аналогично Run, но ожидает завершения запущенной программы
Файлы .lnk здесь не работают.
Учтите, что нежелательно запускать вторую копию уже запущенной программы. Это обусловлено тем, что Windows характеризуется восстановительным режимом работы. Программа запускается второй раз. Второй экземпляр программы отслеживает уже работающий процесс и, обнаружив его, закрывается сам. Таким образом скрипт продолжает работать и после того, как второй процесс закроется (выгрузится из памяти), а старый процесс будет продолжать работать.

9.6.3 Запуск других скриптов как подпрогрвмм (CallScript, CallScriptFunction)

CallScript(*MortScript file* {, *parameters* } **)**
CallScriptFunction(*MortScript file, variable,* {, *parameters* } **)**

Запуск других скриптов как подпрограмм
См.также 8.9 Other script as subroutine (CallScript)

9.6.4 Create new document / element (New)

New(*menu entry* **)**

Создает новый документ (соответствующий запущенному приложению или ему подобному).
Вход в меню осуществляется также как и при входе в меню “New” экрана Today.
Возможны отличия, обусловленные системной локализацией.

К сожалению эта полезная функция сложна при использовании в Windows Mobile 5 и более новых версиях. В этом случае Вы можете либо попытаться использовать ее, либо обратить внимание на ветку реестра
HKEY_LOCAL_MACHINE\Software\Microsoft\Shell\Extensions\NewMenu.

Не доступна на PC

Пример:
New("Appointment")

9.6.5 Запуск приложения в указанное время (RunAt)

RunAt*(Unix timestamp, application [, parameter])*
RunAt*(year, month, day, hour, minute*
, applikation [, parameter])

Запускает приложение в указанное время. Для этого программа становится в очередь. Мобильник (PPC) при необходимости будет «разбужен».
“Unix timestamp” – это время в секундах, отсчитанное от 01/01/1970. Этот вариант интересен для комбинации с TimeStamp(), т.к. TimeStamp()+86400 для каждых суток (24 часа* 60 минут * 60 секунд = 86400).
На многих устройствах MortScripts не может быть запущен непосредственно. В этом случае Вы можете запустить MortScripts.exe из скрипта как параметр, т.е.
RunAt(starttime, SystemPath("ScriptExe") \ "MortScript.exe", \
""""" & SystemPath("ScriptPath") \ "notify.mscr" & """")
Другая проблема: На многих устройствах с WM5 происходит «пробуждение» и запуск программы без включения дисплея и устройство сразу после запуска программы опять «засыпает». Часто помогает ко-

манда ToggleDisplay(ON) в начале скрипта, если же нет, то надо либо обновить систему, либо подправить системный регистр.
Не работает на PC.

9.6.6 Запуск приложений при каждом включении устройства (RunOnPowerOn)

RunOnPowerOn*(application [, parameter])*

Запуск приложения каждый раз при включении устройства. Для этого программа ставится в очередь на исполнение.
Тщательно обдумайте необходимость использования этой команды, в случае удаления или перемещения указанной программы Вас замучают сообщения об ошибке.
Пожалуйста учтите советы и замечания, относящиеся к команде RunAt в плане запуска скриптов или работе в среде WM5.
Not available for: PC
Не работает на PC.

9.6.7 Удаление приложения из „Notification Queue“

RemoveNotifications*(application [, parameter])*

Удаление программы из очереди на исполнение, т.е. теперь она не будет автоматически запускаться в указанное время (RunAt) или по событию (типа RunOnPowerOn).
В случае нескольких вхождений программы в очередь на исполнение (например, после нескольких команд “RunAt”, все они будут удалены.
Если указан параметр, то только программы с соответствующим параметром будут удалены. Иначе говоря все вхождения программы будут удалены, вне зависимости от того какой параметр объявлен. Для удаления только вхождений без параметра используйте пустую строку ("") в качестве параметра.
Не работает на PC.

9.7 Окна приложений

9.7.1 Заголовки окон – как MortScript обнаруживает окна

Многие команды и функции в этом разделе работают с заголовками окон.
MortScript оперирует с окнами, которые в заголовке содержат указанный текст. При этом имеет значение регистр, т.е. “Word” не соответствует “WORD”. В случае нескольких окон MortScript пытается найти лучшее соответствие.относительно предыдущего окна (проверка начинается с самого верхнего окна):

- Это окно является главным? (не дочернее, или дочернее от настольного компьютера)
- Заданный текст найден в начале заголовка?
- Окно видимое? (т.е не является фоновым, не отображаемым в списке задач)

9.7.2 Показ и активация окна (Show)

Show*(window title)*

Активация окна с указанным заголовком.

9.7.3 Минимизация/скрытие окна (Minimize)

Minimize*(window title)*

Минимизирует (или, на устройствах с Windows Mobil, скорее скрывает) окно с указанным заголовком.

9.7.4 Закрытие окна / завершение приложения (Close)

Close*(window title)*

Закрывает окно с указанным заголовком. Если это главное окно приложения, то приложение обычно также закрывается. Однако для некоторых программ это не так.

9.7.5 Получение заголовка активного окна (ActiveWindow)

string = *ActiveWindow()*

Возвращает заголовок активного окна.

9.7.6 Проверка активности окна (WndActive)

bool = WndActive(window title)

Возвращает TRUE, если окно с указанным заголовком активно, в противном случае FALSE.
Указанный текст может находиться в любом месте заголовка окна, сравнивается только регистр. Например WndActive("top") возвращает TRUE, если активен экран Today (это заголовок «Рабочего стола»), и FALSE, если активно окно с заголовком "Top program".

9.7.7 Проверка существования окна (WndExists)

bool = **WndExists(** *window title* **)**

Схожа с WndActive(), также возвращает TRUE, если окно существует, но не активно.

9.7.8 Ожидание, пока окно существует (WaitFor)

WaitFor(*window title, seconds* **)**

Ожидание (не более указанного времени), если указанное окно существует.

9.7.9 Ожидание активизации окна (WaitForActive)

WaitForActive(*window title, seconds* **)**

Ожидание (не более указанного времени), если указанное окно активно.

9.7.10 Получение заголовка окна / содержимого элемента (WindowText)

string = **WindowText(** *x, y* **)**

Получение текстовых элементов окна, которые находятся в указанном месте. В большинстве диалоговых окон таким способом вы можете получить метки, метки кнопок или содержимое редактируемых боксов.
Это не работает как положено в приложениях с искаженными (трансформированными) элементами (например во многих играх) или в т.н. косых боксах. В этих случаях обычно возвращается либо пустая строка, либо название приложения.

9.7.11 Получение координат окна (GetWindowPos, WndLeft, -Right, -Top, -Bottom)

GetWindowPos(*window name, left, top, right, bottom* **)**
int = **WndLeft(** *window name* **)**
int = **WndRight(** *window name* **)**
int = **WndTop(** *window name* **)**
int = **WndBottom(** *window name* **)**

Получение позиции окна с указанным именем.
GetWindowPos присваивает значения переменным указанным как left, top, right, and bottom.
Функция возвращает все координаты, характеризующие расположение окна.

9.7.12 Передача окну специальных команд (SendOK, SendCancel, SendYes, SendNo)

SendCommand(*[window title,] command id* **)**
SendOK *[* **(** *window title* **)** *]*
SendCancel *[* **(** *window title* **)** *]*
SendYes *[* **(** *window title* **)** *]*
SendNo *[* **(** *window title* **)** *]*

Эта команда эмулирует нажатие соответствующей клавиши.
Если заголовок окна (window title) не указан, то команда применяется к активному окну.
Не гарантируется работа со всеми программами, т.к. не все программы используют стандартные сигналы.

9.7.13 Посылка расширенных команд/сообщений (SendCommand, SendMessage, PostMessage)

SendCommand(*[window title,] command id* **)**
PostMessage(*[window title,] message id, wparam, lparam* **)**
SendMessage(*[window title,] message id, wparam, lparam* **)**
int = **SendMessage(** *[window title,] message id, wparam, lparam* **)**

SendCommand позволяет Вам послать идентификатор любой допустимой команды (command id) (обычно применяется для кнопок и пунктов меню), но для этого требуется хорошее знание программы, т.к. command id свои для каждой программы и могут отличаться в разных версиях одной программы.
То же самое относится и к SendMessage и PostMessage, содержимое которых еще более не стандартно. Вам необходимо хорошее знание идей программиста, создавшего программу, для их использования, иначе можно создать большие проблемы вплоть до краха программы при попытке послать непредусмотренный для программы набор данных. SendMessage позволяет послать программе управляющее сообщение, а также вернуть поученное значение. С помощью PostMessage сообщение добавляется в очередь сообщений, Ваш скрипт приступает к выполнению следующего действия не дожидаясь управляющего сообщения. MortScript поддерживает только цифровые параметры.

9.8 Keystrokes

9.8.1 Посылка строки (SendKeys)

SendKeys(*[window title,] string* **)**

Посылает сигнал, имитирующий ввод текста нажатием клавиш для указанного или активного в данный момент (если не указан заголовок окна) окна.
Примеры:
SendKeys("My window", "Hi, how are you?")
SendKeys("Some text")

9.8.2 Посылка спецсимволов (например клавиш перемещения курсора) (Send...)

Send*Special***(** *key name [, state]]* **)**

Эмулирует нажатие кнопок.
Доступна эмуляция следующих кнопок: Alt, Ctrl, Shift, CR, Win, Context, Tab, ESC, Space, Up, Down, Left, Right, Home, End, PageUp, PageDown, Delete, Backspace, Insert, Snapshot, F1 - F12, LeftSoft, RightSoft.
Наименования кнопок должны соответствовать определенным строкам. Регистр не важен (т.е. “Esc” и “ESC” равнозначны). Не все кнопки доступны для эмуляции на всех системах одновременно, например, программные кнопки не могут эмулироваться на настольных компьютерах.
Возможно использование цифровых кодов кнопок. Значения цифровых кодов сильно зависят от аппаратных модификаций и используются в основном продвинутыми хакерами.

Если параметр state опущен, то имитируется кратковременное нажатие кнопки с отпусканием. Вы можете использовать значения этого параметра “down” и “up” для нажатия и последующего, через какое-то время, отпускания кнопки. (удерживание для Alt, Ctrl, Shift, and Win). Не забывайте дать команду “up”, иначе могут возникнуть неприятные ситуации в дальнейшем.

Send*Special [* **(** *window title [, Ctrl?, Shift? [, Alt?]]* **)** *]*

Активирует указанное окно и передает ему указанные спецсимволы (Special). Если окно не указано, то используется активное в данный момент окно.
Ctrl?, Shift?, and Alt? переключают соответствующие клавиши. Если значение параметра TRUE, то клавиша нажимается одновременно со спецсимволом.

Используемые спецсимволы:
CR.....Перевод каретки
Tab.....Табуляция
Esc.....Отмена
Space.....Пробел
Backspace.....Удаление символа слева от курсора (”<-”)
Delete.....Удаление символа справа от курсора („Del“)
Insert.....„Ins.” («Вставка», Обычно используется для переключения между режимами замены и вставки символов при редактировании)
Up/Down/Left/Right.....Перемещение курсора
Home.....„Home“, перемещение курсора в начало строки
End.....„Ende, перемещение курсора в конец строки
PageUp/PageDown.....Страница вверх / вниз („Page ↑” / „Page ↓”)
LeftSoft/RightSoft....„Программируемые кнопки“ на Смартфонах и PPCs начиная с WM5
Win.....клавиша „Windows“ на Smartphones and PPCs since WM5 («Пуск»)
Context.....„Context menu“ на PC (настольных ПК) и Смартфонах/PPC начиная с WM5

Примеры:
SendCR("ERROR")
SendDown
SendHome("",0,1) (выделение первого символа в строке)

9.8.3 Копирование содержимого экрана в буфер обмена (Snapshot)

Snapshot *[* **(** *window title* **)** *]*

Активация указанного окна (если параметр указан) и копирование содержимого экрана в буфер обмена (Соответствует функции клавиши “Print screen” на настольных ПК, может не работать в отдельных программах).

9.8.4 Имитация нажатия Ctrl+клавиша (SendCtrlKey)

SendCtrlKey (*[window title,] key* **)**

Имитация нажатия комбинации Ctrl+клавиша.
Например, SendCtrlKey("v") имитирует нажатие комбинации клавиш Ctrl+V (вставка в буфер обмена) для текущего окна.
Регистр значения не имеет, "v" и "V" воспринимаются одинаково.
Имя клавиши должно соответствовать только одному буквенно-цифровому символу. Разумеется, строка с именем клавиши может являться результатом выражения (значением переменной).

9.9 Щелчки мышью / нажатие

9.9.1 Одиночный щелчок (MouseClick)

MouseClick(*[window title,] x, y* **)**

RightMouseClick(*[window title,] x, y* **)**
MiddleMouseClick(*[window title,] x, y* **)**

Эмулирует щелчок мышью в указанной точке.
Если указано окно, то позиция точки указывается относительно его верхнего левого угла. Если окно имеет выделенный участок (например текстовый бокс), то может быть указана любая точка внутри этого участка.
Если окно не указано, то левым верхним углом является точка с координатами 0,0.
Команды Right... и Middle... эмулируют щелчок соответственно правой и средней кнопкой мыши и доступны только на настольном компьютере.

9.9.2 Двойной щелчок (MouseDblClick)

MouseDblClick(*[window title,] x, y* **)**
RightMouseDblClick(*[window title,] x, y* **)**
MiddleMouseDblClick(*[window title,] x, y* **)**

Аналогично MouseClick, но эмулируется двойной щелчок.
Команды Right... и Middle... эмулируют двойной щелчок соответственно правой и средней кнопкой мыши и доступны только на настольном компьютере.

9.9.3 Раздельное нажатие/отпускание кнопки мыши (MouseDown/MouseUp)

MouseDown(*[window title,] x, y* **)**
MouseUp(*[window title,] x, y* **)**
RightMouseDown(*[window title,] x, y* **)**
RightMouseUp(*[window title,] x, y* **)**
MiddleMouseDown(*[window title,] x, y* **)**
MiddleMouseUp(*[window title,] x, y* **)**

Эмулирует нажатие и, соответственно, отпускание кнопки мыши. Параметры такие же как и в MouseClick. Эти две команды используются в паре. Т.е. Вы можете эмулировать «Нажатие с удержанием» или «Схватить и перетащить» (MouseUp в другой позиции).
Команды Right... и Middle... эмулируют щелчок соответственно правой и средней кнопкой мыши и доступны только на настольном компьютере.

9.10 Ожидание

9.10.1 Фиксированная задержка в миллисекундах (Sleep)

Sleep(*milliseconds* **)**

Ожидание указанное время.

9.10.2 Показ сообщения в течении указанного времени / до выполнения условия (SleepMessage)

SleepMessage(*seconds, message [, title [, OK allowed? [, condition]]]* **)**

Показывает сообщение в течении указанного времени.
If OK allowed? is TRUE, the dialog can be dismissed with a button, if not, this is not possible.
Если параметр OK allowed? равен TRUE, то сообщение может быть закрыто нажатием кнопки OK, если не TRUE, то эта возможность недоступна.
Если указано условие, то его выполнение проверяется каждую секунду и при выполнении диалог закрывается.

См.также 9.20.10 Set font for big messages (SetMessageFont)

Пример:

SleepMessage(10, "Waiting for PocketWord", "Wait...", 0, wndExists("Word"))

9.10.3 Ожидание для окна (WaitFor / WaitForActive)

Смотри 9.7.8 Wait until a window exists (WaitFor) и 9.6.8 Wait until a window becomes active (WaitForActive)

9.11 Время

9.11.1 Временная метка (TimeStamp)

int = **TimeStamp()**

Возвращает текущее время в секундах, прошедшее с 01/01/1970.

9.11.2 Формат отображения времени (FormatTime)

string = **FormatTime(** *format [, timestamp]* **)**

Возвращает временную метку или текущее время, если иное не указано, форматированные в соответствии со строкой format.
Эти символы заменяются соответствующими значениями:
H Часы (00-23)
h Часы (01-12)
a am/pm
A AM/PM
i Минуты (00-59)
s Секунды (00-59)
d Дни (01-31)
m Месяцы (01-12)
Y Года (4 цифры)
y Year (2 цифры)
w Номер дня недели (0=Воскресенье … 6=Суббота)
u Метка времени (Unix timestamp)
{MM}Название месяца (например,. “January”)
{M} Сокращенное название месяца (например, “Jan”)
{WW}День недели (например, “Monday”)
{W} Сокращенное наименование дня недели (например, “Mon”)
Все остальные символы в строке выводятся без изменений
Учтите, что все возвращаемые значения являются строковыми. Это позволяет использовать лидирующие нули, например 02 для февраля, что удобно для использования их при создании имен файлов. Однако, это может явиться проблемой при использовании массивов. Вам либо прийдется переопределить элементы массива в строковые (“Month["01"] = "First"”), либо конвертировать строковые переменные в числовые, например, используя “FormatTime("m")*1”.

Примеры:
x = FormatTime("h:i:s a")
x = FormatTime("m/d/Y", TimeStamp() + 86400)

9.11.3 Возвращает текущее время в нескольких переменных (GetTime)

GetTime(*variable, variable, variable* **)**

Возвращает текущее время в три переменные – час, минута, секунда.

GetTime(*variable, variable, variable, variable, variable, variable* **)**

Подобно предыдущей команде, но в 4-я, 5-я и 6-я переменные соответствуют дню месяца, месяцу и году (4 цифры).

Учтите, что все возвращаемые значения являются строковыми. Это позволяет использовать лидирующие нули, например 02 для февраля, что удобно для использования их при создании имен файлов.

9.12 Копирование, переименование, перемещение, и удаление файлов

9.12.1 Копирование одного файла (Copy)

Copy(*source file, target file [, overwrite?]* **)**

Копирование файла.
Параметр target должен содержать путь и имя файла
Если параметр overwrite? равен FALSE или отсутствует, то уже существующий файл не будет перезаписан.

Пример:
Copy("\My documents\test.txt", "\Storage\text.txt")

9.12.2 Копирование нескольких файлов (XCopy)

XCopy(*source files, target directory [, overwrite?*
[, subdirs?]] **)**

Копирование файлов в директорию-приемник (target directory).
Имя файлов-источников может содержать маски (* и ?, например "\My documents*.psw", но не "\My **.psw").
Источник-директория должна уже существовать.
Если параметр overwrite? равен FALSE или отсутствует, то уже существующие файлы не будут перезаписаны.
Если параметр subdirs? равен TRUE, то все файлы-источники, соответствующие маскам и содержащиеся в поддиректориях, также будут скопированы. Поддиректории-приемники при необходимости будут созданы.

Примеры:
XCopy("\My documents*.txt", "\Storage")
XCopy("\My documents*.txt", "\Storage", TRUE, TRUE) (будет скопирован также
“\My documents\texts\x.txt” в “\Storage\texts\x.txt”)

9.12.3 Переименование или перемещение одного файла (Rename)

Rename(*source file, target file [, overwrite?]* **)**

Переименование или перемещение файла
Необходимо также указывать путь для файла-приемника
Если параметр overwrite? равен FALSE или отсутствует, то уже существующий файл перезаписан не будет.

9.12.4 Перемещение нескольких файлов (Move)

Move(*source files, target directory [, overwrite?*
[, subdirs?]] **)**

Перемещает файлы в директорию-приемник.
Имя файлов-источников может содержать маски (* и ?, например "\My documents*.psw", но не "\My **.psw").
The target must be an existing directory.
Источник-директория должна уже существовать.
Если параметр overwrite? равен FALSE или отсутствует, то уже существующие файлы не будут перезаписаны.

Если параметр subdirs? равен TRUE, то все файлы-источники, соответствующие маскам и содержащиеся в поддиректориях, также будут перемещены. Поддиректории-приемники при необходимости будут созданы. Поддиректории-источники не будут перемещены, если они после перемещения оказываются пустыми.

9.12.5 Удаление файла(ов) (Delete)

Delete(*files* **)**

Удаление файла(ов).
Имена файлов могут содержать маски (* и ?, например "\My documents*.psw", но не "\My **.psw").

9.12.6 Удаление файлов и поддиректорий (DelTree)

DelTree(*files* **)**

Удаляет файл(ы), включая все поддиректории.
Если (под)директория уже пустая, то она не будет удалена.
Имена файлов могут содержать маски (* и ?, например "\My documents*.psw", но не "\My **.psw"), которые могут использоваться также и для поддиректорий.
Если фильтр по маске не указан (например, DelTree("\Temp")) и указанный путь существует, то это соответствует маске *.*.

Пожалуйста будьте внимательны и аккуратны!

9.12.7 Создание ярлыка/ссылки (CreateShortcut)

CreateShortcut(*shortcut file, target file [, overwrite?]* **)**

Создает ярлык (ссылку) на файл. Он может быть, например, использован в меню автозапуска.
Если параметр overwrite? равен FALSE или отсутствует, то существующий ярлык перезаписан не будет.

Пример:
CreateShortcut("\Windows\Start Menu\Test.lnk","\Storage\Test.exe")

9.13 Чтение и запись текстовых файлов

9.13.1 Чтение текстового файла (ReadFile)

string = **ReadFile(** *file name [, length [, codepage]]* **)**

Считывание содержимого файла в переменную. Размер файла ограничен 512кБ или доступной памятью.
Если параметр length равен 0, то по умолчанию используется максимум – 512кБ.

Возможные значения параметра codepage могут принимать либо номер кодовой страницы (если Вы ее знаете), например 1252 (Western Europe), 437 (American DOS), либо любую из нижеприведенных строк:

- "latin1" (Western Europe)
- "jis" (Japanese)
- "wansung" (Korean)
- "johab" (Korean)
- "chinesesimp" (Chinese simplified)
- "chinesetrad" (Chinese traditional)
- "hebrew"
- "arabic"
- "greek"
- "turkish"
- "baltic"

- "latin2" (mostly Eastern Europe)
- "cyrillic"
- "thai"
- "utf8" (спецкодировка только для не-ASCII символов)
- "unicode" (2 байта на каждый символ, более точный UTF-16 little endian)
- "utf8-prefix" (похож на "utf8", но файл начинается с 16-ричных значений EF BB BF, которые являются индикатором для некоторых редакторов/программ)
- "unicode-prefix" (похож на "utf8-prefix", но с префиксом FF FE и unicode)

По умолчанию используется системная кодовая страница, которая зависит от локализации Windows. Файлы в UTF8 или Unicode также распознаются если они начинаются с соответствующих префиксов (см. кодирующий "-prefix" выше).
Вы можете разделить файл на строки используя ForEach (содержимое "^LF^", TRUE)
См.также возможности ForEach для INI-файлов и ReadINI/WriteINI!

9.13.2 Запись в текстовый файл (WriteFile)

WriteFile(*file name, contents [, append? [, codepage]]* **)**

Записывает contents в файл.
Если параметр append? равен FALSE или отсутствует, то, если файл уже существует, он будет перезаписан, иначе содержимое contents будет добавлено в конец существующего файла.
Допустимые кодировки (codepage) см. выше в описании ReadFile. Варианты с "-prefix" допустимы только если параметр append? равен FALSE!

9.13.3 Чтение значений в INI файле (IniRead)

string = **IniRead(** *file name, section, entry* **)**

Чтение записей из INI-файла. Имя секции (section) должно писаться без кавычек

Пример:
x = IniRead("\My documents\test.ini", "Settings", "Test")

9.13.4 Запись значений в INI файл (IniWrite)

IniWrite(*file name, section, entry, value* **)**

Создает запись в INI-файле. Имя секции (section) должно писаться без кавычек
Если MortScript загружен, то он производит анализ и вписывает запись в файл. Это может быть удобнее, чем делать то же самое с помощью нескольких команд (ReadFile, ForEach with split, WriteFile), особенно прибольшом количестве вносимых изменений.

Пример:
IniWrite("\My documents\test.ini", "Settings", "Test", "x")

9.13.5 Доступ к последовательному порту (SetComInfo)

SetComInfo(*port, timeout [, baud rate [, parity [, bits*
[, stop bits [, control]]]]] **)**

Эта команда активирует COM порт.
Она должна использоваться перед ReadFile or WriteFile. Когда Вы вызываете эти функции с параметром „COM1:" вместо имени файла, то доступ к порту предварительно должен быть инициализирован с указанными параметрами.
Используйте ReadFile с максимальным параметром размера (size), иначе возможны длительные задержки вплоть до таймаута. Например data = ReadFile("COM1:", 100).

Parameters:
Port.....Указание порта как имени файла DOS, т.е. "COM1:". Пожалуйста, обращайте внимание на верхний регистр букв и двоеточие!
Timeout.....Время в миллисекундах после которого система деактивирует порт
Baud rate…….Скорость передачи данных. Обычно это 9600, 14400, или 56700, если вы опустили параметр, то по умолчанию используется скорость 9600.
Parity.....Бит контроля. Возможные значения "none", "even", "odd", "mark", и "space". В большинстве случаев и по умолчанию используется "none", иногда even" или "odd".
Bits.....Количество бит в передаваемом байте. В настоящее время почти всегда и по умолчанию используется 8. В редких случаях 7. Меньше практически никогда не используется.
Stop bits.....Количество стоповых бит (между байтами). Возможные значения 1 (по умолчанию), 1.5 (ограничитель строки), и 2.
Control.....Type of flow control. Available are "None", "RTS/CTS" (default), and "XON/XOFF".
Тип потокового управления. Допустимы "None", "RTS/CTS" (по умолчанию), и "XON/XOFF".

Совет: В зависимости от системы, драйверов и устройства не все параметры могут работать корректно. Особенно значение параметра timeout, который даже может быть проигнорирован.

9.14 Системная файловая информация

9.14.1 Проверка существования файла или директории (FileExists/DirExists)

bool = **FileExists(** *file name* **)**
bool = **DirExists(** *directory name* **)**

Возвращает TRUE, если файл или директория существует, FALSE в противном случае.
Также возвращает FALSE, если параметр не соответствующего типа. Например, “FileExists("\Windows")” возвратит FALSE, т.к. это директория, а не файл.

9.14.2 Определение размера свободного дискового пространства (FreeDiskSpace)

int = **FreeDiskSpace(** *directory* **)**

Возвращает свободное дисковое пространство в указанной директории в байтах (максимум 2147483147 = ~2GB).
На устройствах с Windows Mobile надо указать директорию (например “\Storage“ for the storage card), на настольном компьютере букву диска (“D:\...”).

9.14.3 Определение размера дискового пространства (TotalDiskSpace)

int = **TotalDiskSpace(** *directory* **)**

Возвращает размер дискового пространства в указанной директории в байтах (максимум 2147483147 = ~2GB).
На устройствах с Windows Mobile надо указать директорию (например “\Storage“ for the storage card), на настольном компьютере букву диска (“D:\...”).
Конечно, 2GB во многих случаях недостаточно, но использование больших значений невозможно из-за того, что они требуют 32 бита, а такие числа MortScript воспринимает как отрицательные.

9.14.4 Получение размера файла (FileSize)

int = **FileSize(** *file name* **)**

Возвращает размер файла в байтах

9.14.5 Получение времени создания файла (FileCreateTime)

int = **FileCreateTime(** *file name* **)**

Возвращает время создания файла в формате unix, или 0, если файл не существует.
См.также 9.11 Time о том как сравнивать полученные данные и их форматировании.

9.14.6 Получение времени последнего изменения файла (FileModifyTime)

int = **FileModifyTime(** *file name* **)**

Возвращает время последнего изменения файла в формате unix, или 0, если файл не существует.
См.также 9.11 Time о том как сравнивать полученные данные и их форматировании.

9.14.7 Получение атрибутов файла (FileAttribs)

bool = **FileAttribute(** *file name, attribute* **)**

Возвращает текущие значения атрибутов указанного файла. TRUE = атрибут установлен, FALSE = не установлен
Allowed values for “attribute”:
Допустимые значения параметра “attribute”:

- directory (указанный файл является директорией?)
- hidden (скрытый файл?)
- readonly (защита от записи?)
- system (системный файл?)
- archive (не архивирован?)

Эти значения строковые (т.е. либо заключены в кавычки, либо являться значением переменной или выражения).

9.14.8 Установка атрибутов файла (SetFileAttribute, SetFileAttribs)

SetFileAttribute(*file name, attribute, set?* **)**

Установка (set?=TRUE) и соответственно сброс (set?=FALSE) указанного атрибута файла. Все остальные атрибуты остаются без изменений.
Допустимые значения параметра “attribute”:

- directory (указанный файл является директорией?)
- hidden (скрытый файл?)
- readonly (защита от записи?)
- system (системный файл?)
- archive (не архивирован?)

Эти значения строковые (т.е. либо заключены в кавычки, либо являться значением переменной или выражения).

Примеры:
SetFileAttribute("\Test.txt", "hidden", TRUE)
➔ устанавливает атрибут «скрытый»
SetFileAttribute("\Test.txt", "readonly", FALSE)
➔ снятие защиты от записи

SetFileAttribs(*file name, read only? [, hidden? [, archive?]]* **)**

Установка указанных атрибутов. TRUE (или любое другое цифровое значение за исключением 0/FALSE) устанавливает атрибут, FALSE – сбрасывает. Пустая строка ("") оставляет атрибут без изменений.

Примеры:

SetFileAttribs("\Test.txt", "", TRUE)

➔ устанавливает атрибут скрытого файла, атрибут «только чтение» (и другие атрибуты) остаются без изменений.

SetFileAttribs("\Test.txt", FALSE)

➔ удаляет атрибут «только чтение», все остальные атрибуты остаются без изменений.

9.14.9 Получение номера версии (FileVersion / GetVersion)

string = **FileVersion(** *file name* **)**
GetVersion(*file name, variable, variable, variable, variable* **)**

Возвращает номер версии ресурса либо в строке ("a.b.c.d"), либо в отдельных переменных (целочисленные значения).
Эта информация может отсутствовать или быть не точной в некоторых файлах. Если информация есть, то она присутствует в 4-х уровнях, обычно главный номер, номер подверсии, номер патча, номер билда.
При использовании функции FileVersion эти части разделяются точками, (типа "3.1.2.0"), в команде GetVersion для каждой части предназначена отдельная переменная.

9.15 ZIP архивы

9.15.1 Важные замечания

Эта функция, которая теперь стала доступной, не может перезаписывать файлы уже содержащиеся в архиве. Если уже существующий в архиве файл вновь добавляется в архив, то в архиве появляются два одинаковых файла с разным временем их создания. Не все архиваторы имеют эту особенность. Поэтому рекомендуется на всякий случай в такой ситуации заново создавать архив.
Другая проблема заключается в кодировке имен файлов в ZIP архивах. Этот момент не стандартизирован, а Unicode не поддерживается.
MortScript использует, как и большинство Windows/DOS программ, кодовую страницу DOS – 437. Это может приводить к проблемам, если в ваших файлах содержатся специальные символы или символы иностранных языков (например, Кириллические или Греческие символы). Кроме того, ZIP функции на Java используют UTF8.

9.15.2 Сжатие одного файла (ZipFile)

ZipFile(*source file, ZIP file, file name in archive [, rate]* **)**

Добавляет указанный файл в архив. Для файла-источника (сжимаемого файла) и для ZIP файла должны быть указаны полные пути. В противном случае имя файла в архиве обычно прописывается с относительным путем либо вообще без пути.
The compression rate ranges from 1=no compression to 9=best, if omitted, it defaults to 8.
Уровень сжатия (rate) можно менять от1до 8, если параметр опущен, то по умолчанию сжатие происходит с уровнем 8.

Пример:

```
ZipFile( "\Storage\Test\manual.psw", "\Storage\mans.zip", \
      "test\testman.psw" )
```

9.15.3 Сжатие нескольких файлов (ZipFiles)

ZipFiles(*source files, ZIP file* [, *subdirectories?*
[, *path in archive* [, *rate*]]])

Добавляет несколько файлов в архив. Указание файлов-источников аналогично командам XCopy или Move с использованием пути и символов маскирования в имени файлов (например, "\My documents*.psw").
Если параметр *subdirectories?* равен TRUE, то фильтр имен файлов используется также и для поддиректорий, т.е. "\My documents*.psw" включает также и "\My documents\Word\x.psw".

Если параметр *path in archive опущен,* то в архиве будут отсутствовать пути к сжатым файлам, т.е. файл "\My documents\Word\x.psw" в архиве сохраняется как "Word\x.psw", "\My documents\x.psw" как "x.psw", и т.д. Если . *path in archive* = "docs", то в архиве файлу "docs\Word\x.psw" будет соответствовать файл "docs\x.psw".

Примеры:
ZipFiles("\Storage\Test*.psw", "\Storage\mans.zip", TRUE, "test")

- ➔ Сжимает все файлы, удовлетворяющие маске *.psw из директории \Storage\Test и ее поддиректорий в директорию “test” в архивном файле archive \Storage\mans.zip

ZipFiles("\Storage\Test*.jpg", "\Storage\jpgs.zip")

- ➔ Сжимает все файлы, удовлетворяющие маске *.jpg из \Storage\Test в корневую директорию в архивном файле \Storage\jpgs.zip. Поддиректории в директории-источнике будут проигнорированы.

9.15.4 Извлечение файла из архива (UnzipFile)

UnzipFile(*ZIP file, file name in archive, target file* **)**

Извлекает указанный файл из архива.
Для извлекаемого файла (target file) должен быть указан полный путь, путь, указанный в архиве игнорируется

Пример:
UnzipFile("\Storage\mans.zip", "test\test.psw", \
"\Storage\test.psw")

- ➔ Разархивирует файл „test\test.psw“ из архив ”\Storage\mans.zip“ в файл ”\Storage\test.psw“

9.15.5 Извлечение содержимого архива (UnzipAll)

UnzipAll(*ZIP file, target directory* **)**

Извлекает все файлы, содержащиеся в архиве, в указанную директорию. Используются пути, содержащиеся в архиве, а, при необходимости, создаются.

9.15.6 Извлечение с использованием пути, содержащегося в архиве (UnzipPath)

UnzipPath(*ZIP file, path in archive, target directory* **)**

Поддиректории в этом пути извлекаются также. Указанный путь не создается в принимающей директории, за исключением поддиректорий. Принимающая директория должна уже существовать.

Пример:
UnzipPath("\Storage\mans.zip", "test", "\Storage\test-unzip")

- ➔ Извлечение всех файлов, содержащихся в директории “test” и ее поддиректориях из архива “\Storage\mans.zip” to “\Storage\test-unzip”. Т.е., “test\sub\x.psw” будет извлечен в “\Storage\test-unzip\sub\x.psw”.

9.16 Соединения

9.16.1 Создание соединения (Connect)

Connect
Connect(*connection name* **)**
Connect(*title, message* **)**

Подключение к Интернет.
Подключение без параметров пытается использовать соединение по умолчанию На некоторых устройствах это работает не надежно.

Соединение с указанием названия соединения (connection name) использует указанное соединение. Имя соединения легко находится и настраивается в системных настройках и обычно инициализируется Вашим оператором связи. В название соединения обычно входят слова “Internet”, “The Internet” или подобные.
Если в качестве параметра указаны (заголовок) title и (запрос) message, то просматриваются все доступные соединения (может предоставляться Выбор), и выбирается одно из из используемых.
Не доступно на PC, PNA (навигаторах).

9.16.2 Закрытие соединения (CloseConnection/Disconnect)

CloseConnection
Disconnect

Закрывает соединение, установленное с помощью Connect. Это только сигнал для операционной системы о том, что MortScript больше не использует данное соединение. В зависимости от скорости реакции системы соединение может еще некоторое время оставаться активным.
Contrary to this, Disconnect terminates all connections, including ActiveSync. Sadly, since WM5 AKU3, this doesn't work anymore. Currently, there's no known way to hang up a connection from a program.
Напротив Disconnect закрывает все соединения, включая ActiveSync. К сожалению, начиная с WM5 AKU3, эта команда больше не работает. К сожалению нет возможности разрывать соединение из программы.
Не доступно на PC, PNA (навигаторах).

9.16.3 Проверка соединения (Connected/InternetConnected)

bool = **Connected()**
bool = **InternetConnected(** [*URL*] **)**

Проверка соединения, по крайней мере “RAS connection” (“Remote Access”). Во всяком случае для соединений с использованием ActiveSync, для других соединений на многих устройствах, но не на всех…
Возвращает TRUE, если соединение существует, FALSE в противном случае.
InternetConnected проверяет соединение с Internet. К сожалению большинство устройств возвращает “true” для активно работающего соединения (т.е. функция возвращает TRUE) и проверяет соединение, только при доступности сервера. Выполняя эту функцию вы можете перейти на URL, используемый в качестве параметра для тестирования соединения (например на "http://www.google.com").
Не доступно на PC, PNA (навигаторах).

9.17 Доступ в Интернет

9.17.1 Установка прокси

SetProxy(*proxy* **)**

Указание proxy для доступа по http. При использовании Windows Mobile не просто указать proxy в системных настройках…
Строка параметра должна выглядеть примерно как "proxy.foo.bar:8080".
Not available for: PNA
Не доступно на PNA (навигаторах).

9.17.2 Загрузка (Download)

Download(*URL, target file* **)**

Подобно команде Copy, но использует URL ("http://..." or "ftp://...") как источник и показывает окно прогресса, т.к. обычно занимает длительное время…

Пример:
Download("http://www.sto-helit.de/test.txt", \

"\Storage\text.txt")
Не доступно на Смартфонах, PNA (навигаторах).

9.17.3 Другие возможности

Все файловые операции для чтения одного файла работают также с файлом-источником. Это относится к командам Copy, ReadFile, IniRead и некоторым вариантам ForEach.

9.18 Директории

9.18.1 Создание директории (MkDir)

MkDir(*directory* **)**

Создание директории.
Невозможно одновременное создание (одной командой) директории и поддиректории!
Т.е. MkDir("\My documents\Some\Path") вызовет ошибку, если поддиректория "Some" еще не существует.

9.18.2 Удаление директории (RmDir)

RmDir(*directory* **)**

Удаление директории.
Не работает, если директория не пустая.

9.18.3 Переход в директорию (ChDir)

ChDir(*directory* **)**
Меняет текущую директорию.
Работает только на настольных компьютерах, в Windows Mobile нет понятия «текушая директория».

9.18.4 Получение системных путей (SystemPath)

x = **SystemPath(** *type* **)**

Возвращает путь к системным директориям
Параметр type является строкой, например "StartMenu".
Допустимые значения:

- ProgramsMenu..... “Programs” в меню start
- StartMenu..... Меню start, не работает на смартфонах
- Startup..... Папка автозапуска (содержит ярлыки для программ, запускающихся после soft reset и при загрузке Windows)
- Documents.....”\My documents” или другая соответствующая папка, не работает на устройствах с PPC2002
- ProgramFiles..... ”\Program files” или другая соответствующая папка, не работает на устройствах с PPC2002
- AppData.....”\Application data” или другая соответствующая папка
- ScriptExe..... Путь к MortScript.exe (без имени файла), не работает на смартфонах
- ScriptPath..... Путь к текущему скрипту (без имени файла)
- ScriptName..... Имя текущего скрипта (без пути и расширения)
- ScriptExt..... Расширение текущего скрипта (".mscr" или ".mortrun").

Т.е. вы можете запустить текущий скрипт на выполнение с помощью команды Run (SystemPath("ScriptExe") \ "MortScript.exe", \ SystemPath("ScriptPath") \ SystemPath("ScriptName") & \ SystemPath("ScriptExt"))

9.19 Реестр

9.19.1 Считывание записей из реестра (RegRead)

value = **RegRead(** *root, key, value name* **)**

Считывает указанные значения из реестра.
Для *root* допустимы следующие значения:
HKCU.....HKEY_CURRENT_USER
HKLM..... HKEY_LOCAL_MACHINE
HKCR.....HKEY_CLASSES_ROOT
HKUS..... HKEY_USERS
Поддерживаются только эти аббревиатуры из 4-х букв! Не забудьте про кавычки, если не хотите использовать переменные типа HKCU.
Если *value name* является пустой строкой (""), то используется значение по умолчанию. (В редакторах реестра обычно отображается как ”<Default>” or ”@”).
The value's data type is automatically regarded. DWords are returned as integer number, string values as strings, binary data as a string containing the data as hex dump (e.g. "010ACF"), and “MultiString”s as array with string elements.
Тип данных определяется автоматически. DWords возвращается как целое число, строковое значение как строка, двоичные данные как строка, содержащая восьмеричные числа, а “MultiString” как массив строк.

9.19.2 Запись в реестр (RegWriteString/RegWriteDWord/RegWriteBinary/RegWriteMultiString)

RegWriteString(*root, key, value name, value* **)**
RegWriteDWord(*root, key, value name, value* **)**
RegWriteBinary(*root, key, value name, value* **)**
RegWriteMultiString(*root, key, value name, array* **)**

Запись в реестр.
О допустимых значениях для *root* см. в 9.19.1 Считывание записей из реестра (RegRead)
Если *value name* является пустой строкой (""), то используется значение по умолчанию.
RegWriteString записывает строковое значение (цифровые значения автоматически конвертируются).
RegWriteDWord записывает цифровое значение (строковое значение автоматически преобразовывается, некорректная строка преобразуется в “0”).
RegWriteBinary записывает двоичные данные. Данные должны быть в виде строки, содержащей 16-ричные числа (типа "010A"), пробелы другие символы недопустимы!
RegWriteMultiString записывает последовательность строковых значений (составляющую массив). Все элементы нумеруются начиная с 1, при необходимости элементы конвертируются в строки.

Примеры:
```
RegWriteDWord( "HKCU", "Software\Microsoft\Inbox\Settings", \
"SMSDeliveryNotify",1 )
(Установка текста SMS во многих устройствах)
RegWriteString( "HKCU", "Software\Mort\MortPlayer\Skins", \
"Skin", "Night" )
RegWriteBinary( "HKCU", "Software\Mort\Dummy", "", "C000" )
RegWriteMultiString( "HKCU", "Software\Mort\Dummy", "Days",
Array( "Mon", "Tue", "Wed" ) )
```

9.19.3 Проверка существования значения (RegValueExists)

bool = **RegValueExists(** *root, key, value name* **)**

Возвращает TRUE, если искомое значение (value name) существует, FALSE, если нет.
Значения для *root см. в* RegRead.

9.19.4 Проверка существования ключа (registry path) (RegKeyExists)

bool = **RegKeyExists(** *root, key* **)**

Возвращает TRUE, если указанный ключ («поддиректория» в реестре) существует, FALSE, если нет.
Значения для *root см. в* RegRead.

9.19.5 Удаление значения из реестра (RegDelete)

RegDelete(*root, key, value name* **)**

Удаляет значение из реестра
Значения для *root см. в* RegRead.

9.19.6 Удаление ключа реестра (registry path) (RegDeleteKey)

RegDeleteKey(*root, key, values?, sub keys?* **)**

Удаление ключа («поддиректории» в реестре).
Если *values?* равно TRUE, то все содержащиеся внутри ключа значения удаляются также.
values? используется также для ключей, содержащихся внутри ключа, если *sub keys?* is TRUE.
Т.е. команда RegDeleteKey("HKCU", "\Software\Something", FALSE, TRUE) удалит только пустые подключи (т.к. записи в подключах не должны удаляться).
Если ключ не может быть удален, то будет показано сообщение об ошибке.
Путь не должен содержать ошибки, т.к. в противном случае может произойти удаление целой секции реестра. Команду надо использовать осторожно, особенно при использовании в ней переменных или выражений.

9.20 Диалоги

9.20.1 Ввод текста (Input)

string = **Input(** *message* [, *title* [, *numeric?* [, *multiline?* [, *default*]]]] **)**

Открывает простой диалог, который позволяет ввести любой текст, который будет возвращен функции.
Если *numeric?* равно TRUE, то можно вводить только цифры («-« или «.» также не допустимы!).
Если *multiline?* равно TRUE, то вводимый текст будет отображаться построчно. На многих устройствах параметр *numeric?* с этой опцией будет проигнорирован.
Если вы укажите параметр *default*, то в окне ввода можно будет редактировать вводимый текст.
Учтите, что возвращаемое функцией значение будет строковым, даже если *numeric?* равно TRUE.
См.также 9.20.10 Set font for big messages (SetMessageFont)

9.20.2 Сообщение (Message)

Message(*text* [, *title*] **)**

Отображает текст в окне

9.20.3 Отображение больших сообщений с помощью скроллбара (BigMessage)

BigMessage(text [, *title*] **)**

Подобна команде Message, но не использует системную функцию вывода сообщения, которая меняет размеры выводимого текста (за исключением Смартфонов). Вместо этого используются фиксированный размер диалогового окна, в котором можно прокручивать текст используя скроллбар.
См.также 9.20.10 Set font for big messages (SetMessageFont)

9.20.4 Сообщение в течении указанного времени / до выполнения условия (SleepMessage)

SleepMessage(*seconds, message* [, *title* [, *OK allowed?* [, *condition*]]] **)**

См. 9.10.2 Показ сообщения в течении указанного времени / до выполнения условия (SleepMessage) и 9.20.10 Set font for big messages (SetMessageFont)

9.20.5 Простые вопросительные диалоги (Question)

int = **Question(** *question* [, *title* [, *type*]] **)**

Вывод простых вопросительных диалогов. Используются системные диалоги с надписями на кнопках, которые зависят от локализации Windows.

Допустимые значения параметра *type*:
"YesNo".....Выводит “Yes” и “No” (по умолчанию)
"YesNoCancel".....Выводит “Yes”, “No”, и “Cancel”
"OkCancel".....Выводит “OK” и “Cancel”
"RetryCancel".....Выводит “Retry” и “Cancel”

Параметр *type* должен быть строковым и заключен в кавычки, или, если в качестве параметра *type* используется соответствующая переменная, то она также должна быть строковой.
Возвращаемые значения:
“Yes”, “OK”, “Retry”: 1 (встроенная переменная “YES”)
“No”: 0 (встроенная переменная “NO”)
“Cancel”: 2 (встроенная переменная “CANCEL”)

Имейте в виду, что “Cancel” соответствует оператору условия If либо While, поэтому вы либо используете “If (Question(...., "OkCancel") = CANCEL)” либо “Switch(Question(....))” для соответствующего управления.

9.20.6 Выбор из списка (Choice)

int = **Choice(** *title, hint, default, timeout, value, value* {, *value*} **)**
int = **Choice(** *title, hint, default, timeout, array* **)**

Работает подобно 8.4 ChoiceDefault, но возвращает выбранную запись вместо запуска управляющей структуры.
Удобно использовать Choice как функцию, если возвращаемое значение требуется позже или в другом месте.
См.также 9.20.9 Установка размера и шрифта для выбранной записи (SetChoiceEntryFormat)

9.20.7 Выбор директории (SelectDirectory)

string = **SelectDirectory(** *title, message* [, *default*] **)**

Показывает диалоговое окно для выбора из существующих директорий. Если указан параметр *default,* то окно открывается в том каталоге, в котором это окно было закрыто последний раз

9.20.8 Получение имени файла (SelectFile)

string = **SelectFile(** *title, save?,* [*filter* [,*info* [,*default*]]] **)**

Показывает диалоговое окно выбора файла.
Если *save?* равно TRUE, то пользователь может ввести новое имя файла, иначе он сможет выбирать только из существующих файлов.
Параметр *filter* позволяет отображать в окне только файлы удовлетворяющие маске, например "*.txt" или "prefs.*".
Параметр *info* по разному работает на настольных компьютерах и на других устройствах. На настольных ПК используется стандартный системный диалог. Который не позволяет добавлять тексты (Работает хо-

рошо, но этот способ слишком сложен для скриптовых языков, который должен быть простым и легким), информация в которых содержит описание типа файлов (обычно используется что-то вроде “All files (*.*)”). Поэтому для Windows Mobile, для которого это диалоговое окно слишком объемно, используется собственное диалоговое окно, которое показывает информацию в верхней части окна. Если вы хотите использовать скрипт на обоих платформах, то вы должны использовать текст, понятный одновременно для той и другой, типа “Select text file”.
Если указан параметр *default*, то можно указать path/file (если *save?* равен FALSE, то только существующий).

9.20.9 Установка размера и шрифта для выбранной записи (SetChoiceEntryFormat)

SetChoiceEntryFormat(*entry size* [, *font size*, *font name*] **)**

Это изменяет высоту каждой записи и, если указано, шрифт выпадающего списка. Это относится ко всем диалоговым окнам выбора после использования команды, включая и функцию и управляющую структуру.
Параметр *size* указывается в пикселах. В мобильных устройствах с VGA значение будет удвоено, также как и для размера шрифта.
Параметр *font size* указывается в пикселах, *font name* типа “Courier”, “Tahoma” и т.д. Имейте в виду, что не все устройства содержат все шрифты, “Tahoma” (для мобильных устройств) или “Arial” (для настольных ПК) являются самыми распространенными.

См.также 8.4 Branching with selection dialog (Choice, ChoiceDefault) and 9.20.6 Selection from a list (Choice)

9.20.10 Установка шрифта для больших сообщений (SetMessageFont)

SetMessageFont(*font size*, *font name* **)**

Это меняет шрифт окна сообщения, создаваемого MortScript, т.е. командами BigMessage, SleepMessage, и Input.
Параметры аналогичны параметрам шрифта команды SetChoiceEntryFormat, описанной выше.
См. также 9.20.3 Отображение больших сообщений с помощью скроллбара (BigMessage), 9.20.1 Ввод текста (Input) и 9.20.4 Сообщение в течении указанного времени / до выполнения условия (SleepMessage).

9.21 Процессы (запущенные приложения)

9.21.1 Проверка поддержки управления процессами (SupportsProcHandling)

bool = **SupportsProcHandling()**

Возвращает TRUE, если работа функций типа Kill и ProcExists поддерживается на устройстве.
Эта функция предназначена для навигаторов (PNA), но поддерживается и на других устройствах.
Функции MortScript Kill или ProcExists нуждаются в системной библиотеке (toolhelp.dll), которая включена не во все устройства с обрезанной Windows CE. Т.е. если вы попытаетесь использовать эти функции на устройстве, на котором отсутствует эта библиотека, то получите сообщение об ошибке. Используя предварительно эту функцию вы можете проверить наличие библиотеки и получить об этом информацию с помощью несложного скрипта, формирующего то или иное сообщение в зависимости от значения, возвращаемого данной функцией.

9.21.2 Проверка существования процесса (ProcExists)

bool = **ProcExists(** *process name* **)**

Возвращает TRUE, если указанный процесс (process name) запущен, FALSE в противном случае.
“process name” – это имя EXE-файла. Обычно лучше писать имя файла без указания пути, типа "solitaire.exe", т.к. это быстрее и реже приводит к ошибкам (неправильный путь, опечатка ит.д.). Но при этом

также можно проверить с указанием полного пути. На Windows для настольных ПК проверка с указанием полного пути работает не для всех программ, например для системных утилит и сервисов невозможен запрос с указанием системного пути

9.21.3 Проверка существования процесса скрипта (ScriptProcExists)

bool = **ScriptProcExists(** *script name* **)**

Возвращает TRUE, если указанный скрипт запущен, FALSE в противном случае.
"ProcExists" не работает с MortScripts, т.к. имя процесса всегда "MortScript.exe".
Имя скрипта может быть либо без пути (например, "myscript.mscr") или с указанием полного пути (например, "\My documents\myscript.mscr"), в версии для настольных ПК необходимо также указывать также букву диска.
См. также информацию в 9.21.7 Завершение работы запущенного скрипта (KillScript).

9.21.4 Имя процесса активного окна (ActiveProcess)

string = **ActiveProcess(** [*full path?*] **)**

Возвращает имя программы, открывшей текущее активное окно.
Если "full path?" равен TRUE, то имя программы возвращается с указанием полного пути, в противном случае путь не возвращается.
На настольных ПК не всегда можно получить полный путь.

9.21.5 Имя процесса указанного окна (WindowProcess)

string = **WindowProcess(** window title [**,** full path?] **)**

Возвращает имя программы, открывшей указанное окно.
Если "full path?" равен TRUE, то имя программы возвращается с указанием полного пути, в противном случае путь не возвращается.
На настольных ПК не всегда можно получить полный путь.

9.21.6 Завершение процесса (Kill)

Kill(*process name* **)**

Завершает работу приложения. В параметре должно быть указано либо имя exe-файла без пути (например, solitare.exe), либо с указанием пути к этому файлу. Также как и в ProcExists, можно не указывать путь.
См. 9.21.2 Проверка существования процесса (ProcExists) для получения большей информации.
WARNING: This command kills the process regardless of any losses!
ВНИМАНИЕ: Эта команда уничтожает процесс без сохранения данных.
Поэтому есть вероятность потерять данные, нарушить работу системы, или получить сообщение об ошибке.
Поэтому по возможности используйте команду Close, которая позволяет завершить приложение корректно (сохранить/закрыть файлы и т.д.).

9.21.7 Завершение работы запущенного скрипта (KillScript)

KillScript(*script name* **)**

Завершает указанный скрипт. KillScript ожидает до 3-х секунд завершения выполнения текущей команды скрипта для избежания проблем, связанных с неправильным завершением операций. Если этого не происходит, то процесс завершается аналогично команде Kill.
В параметре «script name» может быть указано либо только имя скрипта (например, "myscript.mscr") либо с указанием полного пути (например, "\My documents\myscript.mscr"), в версии для настольного ПК необходимо также указывать букву диска.

Если имя скрипта указано с путем, то возможен запуск скрипта с таким же именем, но по другому пути. В этом случае лучше указывать полные пути к скриптам (например, используя 9.18.4 Получение системных путей (SystemPath)).
Не забывайте о скрипте, запущенном дважды. Если Вы хотите запустить или остановить задачу, работающую в фоновом режиме, то хорошим стилем будет сначала с помощью дополнительного скрипта проверить существование процесса.
Не ИСПОЛЬЗУЙТЕ команды RunWait или CallScript, т.к. скрипт, содержащий их останется активным до тех пор, пока фоновая задача не завершится, и запущенный повторно прервет выполнение фонового скрипта.!

Example:
```
backScript = SystemPath( "ScriptPath" ) \ "background.mscr"
If ( ScriptProcExists( backScript ) )
If ( Question( "Stop background process?" ) = YES )
KillScript( "background.mscr" )
EndIf
Else
Run( backScript )
EndIf
```

9.22 Сигналы

9.22.1 Изменение громкости системного звука (SetVolume)

SetVolume(*value* **)**

Устанавливает громкость системного звука. Возможные значения от 0 (выкл.) до 255 (макс.громкость). Некоторые устройства, например Loox720, циклически меняют громкость (обычно с 4-мя или 16-ю шагами), но большинство устройств поддерживают все 16 уровней.
Недоступно на: PC.

9.22.2 Воспроизведение WAV файла (PlaySound)

PlaySound(*WAV file* **)**
Воспроизведение указанного файла. Скрипт приостанавливается на время звучания.

9.22.3 Вибрация (Vibrate)

Vibrate(*milliseconds*)

Включает вибрацию с указанной продолжительностью.
На PC (настольном компьютере) включает звук из системного динамика.
На PPC вместо вибратора может быть доступен светодиод, но это не является стандартом.
На большинстве устройств MortScript работает со светодиодом, к которому происходило обращение последний раз. Но это может и не работать.

9.23 Дисплей / экран

9.23.1 Получение цвета точки на экране (ColorAt)

int = **ColorAt(** *x, y* **)**

Получение цвета точки экрана в указанной позиции. В некоторых устройствах полоса с заголовком экрана игнорируется, т.е. возвращается цвет заднего фона экрана today.

9.23.2 Создание цветового кода RGB (RGB)

int = **RGB(** *red, green, blue* **)**

Конвертирование десятичных значений комбинаций красного, зеленого и голубого цветов (от 0 до 255 каждое значение) в формат, используемый функцией ColorAt(...), используемой для сравнения.

9.23.3 Получение красной/зеленой/голубой частей цветового кода (Red, Green, Blue)

int = **Red(** *color* **)**
int = **Green(** *color* **)**
int = **Blue(** *color* **)**

Эти функции дублируют RGB. Они извлекают красную, зеленую, и, соответственно голубую часть цветового кода (от 0 до 255 для каждой части), возвращаемого ColorAt или RGB.

9.23.4 Поворот экрана (Rotate)

Rotate(*orientation* **)**

Поворот экрана.
Доступные значения: 0=по умолчанию (portrait), 90=поворот вправо, 180= разворот, 270=поворот влево.
Не работает на: Smartphone, PC, PPC/PNA с WM2003 и более устаревших.

9.23.5 Установка яркости подсветки (SetBacklight)

SetBacklight(*battery*, *external* **)**

Установка яркости подсветки в соответствии с указанными значениями
Battery при работе от аккумулятора, *external* при работе от внешнего источника питания.
Допустимые значения находятся между 0 и 100.
Эта команда работает не на всех устройствах!
Также значения максимальной яркости различны для каждого устройства. В настоящее время мне известно об устройствах с максимальными уровнями яркости при цифровых значениях 10, 60 и 100, некоторые устройства даже отрабатывают регулировку яркости наоборот (например, 10=выкл., 0- вкл.), или используют уникальные значения для регулировки (например, 0=вкл., 1=выкл., 10=следующий уровень яркости).
Не работает на: PC, PNA, Smartphone

9.23.6 Вкл./Выкл. дисплея (ToggleDisplay)

ToggleDisplay(*on?* **)**

Включение дисплея (*on?* = TRUE) или выключение (*on?* = FALSE).
Не работает на: PC, PNA, Smartphone

9.23.7 Получение размера экрана (ScreenWidth, ScreenHeight)

int = **ScreenWidth()**
int = **ScreenHeight()**

Возвращает ширину и высоту экрана в пикселах.

9.23.8 Получение информации о свойствах экрана (ориентация/разрешение) (Screen)

bool = **Screen(** *type* **)**

Возвращает TRUE, если параметр *type* соответствует реальному свойству, FALSE, если не соответствует.
Допустимые значения параметра *type*:
"landscape" (ландшафтная ориентация экрана?)

"portrait" (портретная ориентация?)
"square" (квадратный экран?)
"vga" (VGA разрешение – не важно – “по умолчанию” or “реальное VGA”)
"qvga" (QVGA разрешение)
Не работает на: PC

9.23.9 Обновление экрана today (RedrawToday)

RedrawToday

Обновляет экран Today. Используется, если после внесения изменений в реестр…
Не работает на: PC

9.23.10 Показ/скрытие курсора ожидания (ShowWaitCursor/HideWaitCursor)

ShowWaitCursor
HideWaitCursor
Показ (ShowWaitCursor) или скрытие (HideWaitCursor) песочных часов (или вращающегося диска в Windows
Mobile).

9.24 Буфер обмена

9.24.1 Копирование текста в буфер обмена (SetClipText)

SetClipText(*text* **)**

Копирование указанного текста в буфер обмена.

9.24.2 Получение текста из буфера обмена (ClipText)

string = **ClipText()**

Возвращает текст из буфера обмена.
Необходимым условием является нахождение в буфере обмена именно текстовых данных, что в свою очередь зависит от приложения, поместившего перед этим свои данные в буфер обмена.

9.25 Memory

9.25.1 Свободный объем основной памяти (FreeMemory)

int = **FreeMemory()**

Возвращает объем доступной основной памяти в килобайтах.
Устройства с системой более старой, чем Windows Mobile 5 динамически разделяют память устройства между памятью программ (FreeMemory()) и “RAM disk” в главной директории (FreeDiskSpace("\")).

9.25.2 Размер основной памяти (TotalMemory)

int = **TotalMemory()**

Возвращает полный размер основной памяти в килобайтах.

9.26 Питание

9.26.1 Проверка подключения внешнего источника питания (ExternalPowered)

bool = **ExternalPowered()**

Возвращает TRUE, если устройство питается от внешнего источника питания, FALSE, если только от аккумулятора.
В случае PC всегда возвращает TRUE, даже если ноутбук питается от аккумулятора.

9.26.2 Текущее состояние аккумулятора (BatteryPercentage, BackupBatteryPercentage)

int = **BatteryPercentage()**
int = **BackupBatteryPercentage()**

Возвращает текущее состояние аккумулятора в процентах. Если устройство подключено к внешнему источнику питания, то это значение может быть некорректно на некоторых устройствах.
Немного о резервных аккумуляторах. Необходимо понимать, что они есть не во всех устройствах (дешевые устройства с незаменяемым аккумулятором или устройства с WM5 и более новые, которые сохраняют данные без резервного аккумулятора), в некоторых устройствах вместо резервного аккумулятора используется конденсатор, который не поддерживает ответ на данный запрос. В этих устройствах возвращаемое значение зависит от их особенностей.
В случае PC всегда возвращает 100, даже если ноутбук питается от аккумулятора.

9.26.3 Выключение устройства (PowerOff)

PowerOff

Выключение устройства (перевод в режим standby). После включения устройства скрипт продолжает свою работу. Имейте в виду, что сразу после включения устройства обычно нет прямого доступа к карте памяти. Поэтому нужно использовать Sleep и/или While(not FileExists(...)) для избежания ошибок…
Не работает на: PC

9.26.4 Отключение автоматического выключения устройства (IdleTimerReset)

IdleTimerReset

Сбрасывает неработающий системный таймер. Таким образом Вы можете предотвратить (при периодическом выполнении команды) или отложить автоматическое выключение устройства.
К сожалению система использует другой таймер для выключения подсветки, который невозможно опросить или модифицировать (по крайней мере он не документирован). Поэтому нет возможности отменить эту опцию устройства.
Также, похоже, на некоторых устройствах IdleTimerReset не эффективна.
Не работает на: PC

9.27 Система

9.27.1 Получение версии системы (SystemVersion)

value = **SystemVersion(** [*element*] **)**

Возвращает версию системы. Если параметр отсутствует или указан неправильно, то возвращаемая строка содержит информацию в формате major.minor.build, например 5.1.2600 for XP with SP2 или 5.1.195 for WM5.
При указании параметра можно получить отдельные части. Доступные значения параметра:
"major".....возвращает основной номер версии (целое число)
"minor"..... возвращает вспомогательный номер версии (целое число)
"build".....возвращает номер издания (целое число)
"platform"..... возвращает платформу, либо "Win95" (включает также 98 и Me), "WinNT" (включает XP and Vista), и "WinCE" (включает Smartphone / PPC / Windows Mobile).

9.27.2 Получение типа MortScript (MortScriptType)

string = **MortScriptType()**

Возвращает строку с типом MortScript, используемым для выполнения скриптов. В настоящее время возвращаются следующие значения:
"PPC".....PocketPC
"PC"..... PC (Настольный)
"SP".....Smartphone
"PNA".....PocketNavigation (обрезанная Windows Mobile для навигационных устройств)

9.27.3 Получение текущей версии MortScript (MortScriptVersion, GetMortScriptVersion)

string = **MortScriptVersion()**
GetMortScriptVersion(*variable*, *variable*, *variable*, *variable* **)**

Получение номера версии MortScript в виде строки ("a.b.c.d") или внутри простых переменных (целочисленные значения).
При использовании функции MortScriptVersion все части строки разделяются точками, (например, "4.1.0.0"), команда GetMortScriptVersion помещает каждую часть в отдельную переменную.

9.27.4 Перезагрузка устройства (Reset)

Reset

Выполняет мягкую перезагрузку. Пожалуйста используйте эту команду только в скриптах для собственного употребления или после предупреждения/вопроса.
Не работает на: PC