JSFX Programming Reference

JSFX. Справочник по программированию.

Introduction - Введение

This is a reference guide to programming audio-oriented effects for REAPER using JS. JS is a scripting language which is compiled on the fly and allows you to modify and/or generate audio and MIDI, as well as draw custom vector based UI and analysis displays.

Это справочник по программированию аудио-ориентированных эффектов для REAPER с помощью JS. JS это скриптовый язык, который компилируется “на лету” и позволяет изменять и/или генерировать аудио- и MIDI-сигналы, а также отображать пользовательский интерфейс и экраны анализаторов на основе векторной графики.

JS effects are simple text files, which when loaded in REAPER become full featured plug-ins. You can try loading existing JS effects and since they are distributed in source form, you can also edit existing effects to suit your needs (we recommend if editing an existing effect you save it as something with a new name--if you do not you may lose your changes when upgrading REAPER).

JS-эффекты являются простыми текстовыми файлами, которые при загрузке в REAPER становятся полнофункциональными плагинами. Вы можете загрузить существующие JS-эффекты и так как они распространяются в виде исходного кода, вы также можете редактировать существующие эффекты для удовлетворения ваших потребностей (при редактировании существующего эффекта, мы рекомендуем вам сохранить его под новым именем — иначе вы можете потерять свои изменения при обновлении REAPER).

This guide will offer an outline of the structure of the text file used by JS, the syntax for writing code, as well as a list of all functions and special variables available for use.

В этом руководстве будут предложены примерная структура текстового файла, используемого JS, синтаксис написания кода, а также список всех функций и специальных переменных, доступных для использования.

JSFX file structure (Структура файла JSFX)

JS Effects are text files that are composed of some description lines followed by one or more code sections.

JS-эффекты представляют собой текстовые файлы, которые состоят из нескольких строк описания, за которыми следуют один или более участков кода.

The description lines that can be specified are
Можно задать следующие строки описания:

- desc: **Effect Description**
  This line should be specified once and only once, and defines the name of the effect which will be displayed to the user. Ideally this line should be the first line of the file, so that it can be quickly identified as a JS file.

  Эта строка должна быть указана только один раз, и определяет имя эффекта, которое будет отображаться пользователю. В идеале эта строка должна быть первой строкой файла, чтобы он был быстро идентифицирован как файл JS.

- slider**1**:**5**<**0**,**10**,**1**>**slider description**
  You can specify multiple of these lines (from 1-64 currently) to specify parameters that the user can control using standard UI controls (typically a fader and text input, but this can vary, see below). These parameters are also automatable from REAPER.
  Можно указать несколько таких строк (от 1 до 64 в настоящее время), для определения параметров, которыми пользователь может управлять с помощью стандартных элементов управления пользовательского интерфейса (как правило, фейдер и текстовое поле ввода, но это может быть реализовано и по-другому, см. ниже). Эти параметры также могут быть автоматизированы из REAPER.

  In the above example, the first **1** specifies the first parameter, **5** is the default value of the parameter, **0** is the minimum value, **10** is the maximum value, and**1** is the change increment. **slider description** is what is displayed to the user.
  В приведенном выше примере первая цифра 1 указывает на то, что это параметр 1, цифра 5 является значением по умолчанию параметра, 0 является минимальным значением, 10 является максимальным значением, и 1 — приращение изменения. **Описание слайдера** — последовательность символов, которая отображается пользователю.

There are additional extended slider syntaxes. One is:
Также есть дополнительные расширенные синтаксисы слайдера. Одним из них является:

slider1:0<**0**,5,**1**{**zerolabel,onelabel,twolabel,threelabel,fourlabel,fivelabel**}>some setting
This will show this parameter with a list of options from "zerolabel" to "fivelabel". Note that these parameters should be set to start at 0 and have a change increment of 1, as shown above.
Это отобразит данный параметр со списком опций от "zerolabel" до "fivelabel". Обратите внимание, что начальное значение таких параметров должно быть задано равным 0 и иметь приращение изменения 1, как показано выше.

Another extended syntax is:
slider1:**/some_path**:**default_value**:slider description
In the above example, the **/some_path** specifies a subdirectory of the REAPER\Data path, which will be scanned for .wav, .txt, .ogg, or .raw files.**default_value** defines a default filename. If this is used, the script will generally use file_open(slider1) in the @serialize code section to read the contents of the selected file.
Еще один расширенный синтаксис:
slider1 **:/некий_путь: значение_по_умолчанию**: Описание слайдера
В приведенном выше примере, **/некий_путь** задает подкаталог пути REAPER\Data, который будет проверен на наличие WAV, TXT, OGG или RAW файлов. **Значение_по_умолчанию** определяет имя файла по умолчанию. Если используется имя по умолчанию, сценарий, как правило, будет использовать file_open(slider1) в разделе кода @serialize для чтения содержимого выбранного файла.

- in_pin:**name_1**
  in_pin:**name_2**
  out_pin:**none**
  These optional lines export names for each of the JS pins (effect channels), for display in REAPER's plug-in pin connector dialog.
  Эти опциональные строки экспортируют имена для каждого из входов-выходов JS плагина (эффект-каналов) для отображения в диалоговом окне плагина “Plug-in pin connector” в Reaper-е.

If the only named in_pin or out_pin is labeled "none", REAPER will know that the effect has no audio inputs and/or outputs, which enables some processing optimizations. MIDI-only FX should specify in_pin:none and out_pin:none.
Если только именованный in_pin или out_pin указан со значением "none", REAPER будет знать, что этот JS плагин не имеет аудиовходов и/или выходов, что позволяет добиться некоторой оптимизации обработки. Для эффектов, работающих исключительно с MIDI, должны быть установлены значения in_pin: none и out_pin: none.

- filename:**0,filename.wav**
  These lines can be used to specify filenames which can be used by code later. These definitions include **0** (the index) and a filename. The indices must be listed in order without gaps -- i.e. the first should always be 0, the second (if any) always should be 1, and so on.
  Эти строки можно использовать для указания имен файлов, которые могут быть использованы в коде позже. Эти определения включают **0** (индекс) и имя файла. Индексы должны быть перечислены по порядку без пропусков — т.е. первый всегда должен быть 0, второй (если есть) всегда должен быть 1, и так далее.

  To use for generic data files, the files should be located in the REAPER\Data directory, and these can be opened with file_open(), passing the filename index.
  Для использования обычных файлов данных, эти файлы должны быть расположены в каталоге REAPER\Data, и их можно открыть с помощью file_open(), с указанием индекса имени файла.

  You may also specify a PNG file. If you specify a file ending in .png, it will be opened from the same directory as the effect, and you can use the filename index as a parameter to gfx_blit(). -- *REAPER 2.018+*
  Вы также можете указать файл PNG. Если указать файл с расширением .PNG, он будет открыт из той же папки, что и эффект, и вы сможете использовать индекс имени файла в качестве параметра для gfx_blit() — *REAPER 2.018+*.

- options:**option_dependent_syntax**
  This line can be used to specify JSFX options:
  - options:gmem=someUniquelyNamedSpace
    This option allows plugins to allocate their own global shared buffer, see gmem[].

- options:want_all_kb

  Enables the "Send all keyboard input to plug-in" option for new instances of the plug-in, see gfx_getchar().

- **@import** *filename*
  (requires REAPER v4.25+)
  You can specify a filename to import (this filename will be searched within the JS effect directory). Importing files via this directive will have any functionsdefined in their @init sections available to the local effect. Additionally, if the imported file implements other sections (such as @sample, etc), and the importing file does not implement those sections, the imported version of those sections will be used.

  Вы можете указать имя файла для импорта (это имя файла будет искаться в каталоге эффекта JS). Импорт файлов с помощью этой директивы будет иметь никаких функций, определенных в их сечений @init доступных для локального воздействия. Кроме того, если импортируемый файл выполняет другие разделы (например, @sample и т.д.), а также импорта файла не реализует те разделы, будет использоваться импортированный версия этих разделах.

  Note that files that are designed for import only (such as function libraries) should ideally be named xyz.jsfx-inc, as these will be ignored in the user FX list in REAPER.

  Обратите внимание, что файлы, которые предназначены только для импорта (такие, как библиотеки функций) в идеале должны быть названы xyz.jsfx-inc, так как они будут проигнорированы в списке пользовательских FX (эффектов?) в REAPER.

  Following the description lines, there should be code sections. All of the code sections are optional (though an effect without any would likely have limited use). Code sections are declared by a single line, then followed by as much code as needed until the end of the file, or until the next code section. Each code section can only be defined once. The following code sections are currently used:
  После строк описания должны следовать разделы кода. Все разделы кода не являются обязательными (хотя эффект вообще без какого-либо раздела кода, скорее всего, имеет ограниченное применение). Разделы кода объявляются одной строкой, за ней расположено столько кода, сколько необходимо, до конца файла, или до следующего раздела. Каждый раздел кода может быть

определен только один раз. В настоящее время используются следующие разделы:

- **@init**

  The code in the @init section gets executed on effect load, on samplerate changes, and on start of playback. If you wish this code to not execute on start of playback, you can set ext_noinit to 1.0, and it will only execute on load or samplerate change (and not on playback start/stop).
  Код в разделе @init запускается на выполнение при загрузке эффекта, при изменении частоты дискретизации, а также в начале воспроизведения. Если вы не хотите запускать этот код в начале воспроизведения, вы можете установить ext_noinit в 1.0, и он будет выполняться только при загрузке или изменении частоты дискретизации (а не при запуске/остановке воспроизведения).

  All memory and variables are zero on load, and if no @serialize code section is defined, then all memory and variables will be re-zeroed before calling @init (on samplerate change, playback start/stop/etc).
  Вся память и переменные равны нулю при загрузке, а также если не определен блок @serialize, то вся память и переменные будут повторно обнулены перед вызовом @init (при смене частоты дискретизации, запуске\остановке воспроизведения и т.д.)

- **@slider**

  The code in the @slider section gets executed following an @init, or when a parameter (slider) changes. Ideally code in here should detect when a slider has changed, and adapt to the new parameters (ideally avoiding clicks or glitches). The parameters defined with sliderX: can be read using the variables sliderX.
  Код в разделе @slider запускается на выполнение следом за @init, или если изменился параметр (slider). В идеале код здесь должен определять, когда слайдер изменился, и адаптироваться к новым параметрам (в идеале избегая щелчков или сбоев). Параметры, определенные с помощью sliderX:, могут быть прочитаны с помощью переменных sliderX.

- **@block**

  The code in the @block section is executed before processing each sample block. Typically a block is whatever length as defined by the audio hardware, or anywhere from 128-2048 samples. In this code section the samplesblock variable will be valid (and set to the size of the upcoming block).

Код в разделе @block выполняется перед обработкой каждого блока семплов. Обычно блок имеет длину, определенную аудио-аппаратными средствами, или в пределах 128-2048 сэмплов. В этом разделе кода переменная samplesblock будет действительна (и установлена в размере предстоящего блока).

- **@sample**

  The code in the @sample section is executed for every PCM audio sample. This code can analyze, process, or synthesize, by reading, modifying, or writing to the variables spl0, spl1, ... spl63.

  Код в разделе @sample выполняется для каждого PCM-аудиосемпла. Этот код может анализировать, обрабатывать, или синтезировать с помощью чтения, изменения или записи переменных spl0, spl1, … spl63.

- **@serialize**

  The code in the @serialize section is executed when the plug-in needs to load or save some extended state. The sliderX parameters are saved automatically, but if there are internal state variables or memory that should be saved, they should be saved/restored here using file_var() or file_mem()(passing an argument of 0 for the file handle). (If the code needs to detect whether it is saving or loading, it can do so with file_avail() (file_avail(0) will return <0 if it is writing).

  Код в разделе @serialize выполняется, когда плагин должен загрузить или сохранить некоторое расширенное состояние. Параметры sliderX сохраняются автоматически, но если есть внутренние переменные состояния или памяти, которые должны быть сохранены, они должны быть сохранены/восстановлены здесь с помощью file_var() или file_mem() (передавая аргумент 0 для дескриптора файла). (Если код необходим для обнаружения будь то сохранения или загрузки, это можно сделать с file_avail() (функция file_avail(0) вернет <0, если она в этот момент записывает).

  Note when saving the state of variables or memory, they are stored in a more compact 32 bit representation, so a slight precision loss is possible. Note also that you should not clear any variables saved/loaded by @serialize in @init, as sometimes @init will be called following @serialize.

  Заметим, что при сохранении состояния переменных или памяти, они хранятся в более компактном 32-битном представлении, так что возможна небольшая потеря точности. Отметим также, что вы не должны очищать ни одну из переменных, сохраненную/загруженную посредством @serialize в @init, так как иногда @init будет вызываться следующим @serialize.

- **@gfx** [width] [height]
- **@gfx** [ширина] [высота]

  The @gfx section gets executed around 30 times a second when the plug-ins GUI is open. You can do whatever processing you like in this (Typically using gfx_*()). Note that this code runs in a separate thread from the audio processing, so you may have both running simultaneously which could leave certain variables/RAM in an unpredictable state.

  Код в разделе @gfx запускается на выполнение около 30 раз в секунду, когда GUI плагина открыт. Вы можете делать все, что обработка вам нравится в этом (обычно, используя gfx_*()). Обратите внимание, что этот код выполняется в отдельном от обработки звука потоке, так что у вас оба потока, работая одновременно, могут оставить некоторые переменные/RAM в непредсказуемом состоянии.

  The @gfx section has two optional parameters, which can specify the desired width/height of the graphics area. Set either of these to 0 (or omit them) to specify that the code doesn't care what size it gets. Note that these are simply hints to request this size -- you may not always get the specified size. Your code in this section should use the gfx_w, gfx_h variables to actually determine drawing dimensions.

  Раздел @gfx имеет два необязательных параметра, которыми можно указать желаемую ширину/высоту графической области. Установите любой из них в 0 (или опустите их), чтобы указать, что коду “все равно”, какой размер будет. Обратите внимание, что они всего лишь подсказки для запроса этого размера — вы не всегда можете получить указанный размер. Ваш код в этом разделе должен использовать специальные переменные gfx_w и gfx_h, чтобы реально определить размеры рисуемой области.

Basic code reference Основной код (описание)

The core of JSFX is custom code written in a simple language (called EEL2), which has many similarities to C. Code is written in one or more of the numerous code sections. Some basic features of this language are:

Ядро JSFX — это сделанный на заказ код, написанный на простом языке (назван EEL2), который имеет много общего с языком Си. Код пишется в одном или нескольких разделах. Несколько основных особенностей этого языка:

- Variables do not need to be declared, are by default global to the effect, and are all double-precision floating point.
- Переменные можно не объявлять, они являются по умолчанию глобальными для эффекта, и все двойной точности с плавающей точкой.

- Basic operations including addition (+), subtraction (-), multiplication (*), division (/), and exponential (^)
- Основные операции включают в себя сложение (+), вычитание (-), умножение (*), деление (/) и возведение в степень (^)

- Bitwise operations including OR (|), AND (&), XOR (~), shift-left (<<), and shift-right-sign-extend (>>). These all convert to integer for calculation.
- Битовые операции включают в себя ИЛИ (OR или |), И (AND или &), исключающее ИЛИ (XOR или ~), сдвиг-влево (shift-left или <<), и сдвиг-вправо-с расширением знака (shift-right-sign-extend или >>). Все они для вычислений преобразуют свои операнды в целое.

- Parentheses "(" and ")" can be used to clarify precedence, contain parameters for functions, and collect multiple statements into a single statement.
- Скобки "(" и ")" могут быть использованы для уточнения приоритета, содержат параметры для функций, и сбора нескольких выражений в одно.

- A semicolon ";" is used to separate statements from eachother (including within parentheses).
- Точка с запятой ";" используется для разделения операторов друг от друга (в том числе в скобках).

- A virtual local address space of about 8 million words, which can be accessed via brackets "[" and "]".
- Виртуальное локальное адресное пространство имеет размер около 8 млн.

слов, которые могут быть доступны через скобки "[" и "]".

- A shared global address space of about 1 million words, accessed via gmem[]. These words are shared between all JSFX plug-in instances.
- Общее глобальное адресное пространство имеет размер около 1 млн. слов, доступных через gmem[]. Эти слова являются общими для всех экземпляров JSFX-плагина.

- Shared global named variables, accessible via the "_global." prefix. These variables are shared between all JSFX plug-in instances.
- Общие глобальные именованные переменные доступны через префикс"_global." и являются общими для всех экземпляров JSFX-плагина.

- User definable functions, which can define private variables, parameters, and also can optionally access namespaced instance variables.
- Функции, определяемые пользователем, могут задавать private-переменные, параметры, а также опционально могут иметь доступ к экземплярам переменных в пространстве имен.

- Numbers are in normal decimal, however if you prefix an '$x' to them, they will be hexadecimal (i.e. $x90, $xDEADBEEF, etc). -- *(REAPER v4.25+ can also take traditional C syntax, i.e. 0x90)*
- Числа записываются в обычном десятичном виде, однако, если вы перед ними используете префикс '$x', то они будут шестнадцатеричными (т.е. $x90, $xDEADBEEF, и т.д.) — *(REAPER v4.25+ также принимает традиционный синтаксис языка Си, т.е. 0x90)*.

- You may specify the ASCII value of a character using $'c' (where c is the character).
- Вы можете указать значение ASCII символа с помощью $'c' (где c — символ).

- If you wish to generate a mask of 1 bits in integer, you can use $~X, for example $~7 is 127, $~8 is 255, $~16 is 65535, etc. -- *REAPER 4.25+*.
- Если вы хотите создать маску 1 бита в целом числе, вы можете использовать $~X, например $~7 это 127, $~8 это 255, $~16 это 65535 и т.д. — *REAPER 4.25 +*.

- Comments can be specified using:
  - // comments to end of line
  - /* comments block of code that span lines or be part of a line */

- Комментарии могут быть указаны с помощью:
    - // Комментарий в одну строку
    - /* Блок комментариев в коде, который может содержать несколько строк или быть частью строки */

Operator reference
Основные операторы

Listed from highest precedence to lowest (but one should use parentheses whenever there is doubt!):
Показаны от наивысшего приоритета к низшему (но надо использовать скобки, когда есть сомнения!):

- **[ ]**

```
z=x[y];
x[y]=z;
```

You may use brackets to index into memory that is local to your effect. Your effect has approximately 8 million (8,388,608) slots of memory and you may access them either with fixed offsets (i.e. 16811[0]) or with variables (myBuffer[5]). The sum of the value to the left of the brackets and the value within the brackets is used to index memory. If a value in the brackets is omitted then only the value to the left of the brackets is used.
Вы можете использовать скобки для индексного доступа к локальной памяти вашего эффекта. Ваш эффект имеет около 8 млн. (8388608) слотов памяти, и вы можете получить к ним доступ либо по фиксированным смещениям (например, 16811[0]) или с помощью переменных (myBuffer[5]). Сумма значения слева от скобок и значения в скобках используется для индексации памяти. Если значение в скобках опущено, то используется только значение слева от скобок.

```
z=gmem[y];
gmem[y]=z;
```

If 'gmem' is specified as the left parameter to the brackets, then the global shared buffer is used, which is approximately 1 million (1,048,576) slots that are shared across all instances of all JSFX effects.
Если 'gmem' указан в качестве левого параметра скобок, то используется глобальный общий буфер — это примерно 1 миллион (1048576) слотов, которые являются общими для всех экземпляров всех JSFX-эффектов.

- **!value** -- returns the logical NOT of the parameter (if the parameter is 0.0, returns 1.0, otherwise returns 0.0).

- **!value** — возвращает логическое НЕ параметра (если параметр равен 0.0, то возвращает 1.0, в противном случае возвращает 0.0).
- **-value** -- returns value with a reversed sign (-1 * value).
- **-value** — возвращает value с обратным знаком (-1 * value).
- **+value** -- returns value без изменений.
- **+value** — возвращает value без изменений.

- **base ^ exponent** -- returns the first parameter raised to the power of the second parameter. This is also available the function pow(x,y)
- **base ^ exponent** — возвращает первый параметр, возведенный в степень второго параметра. Также доступна функция pow(x,y).

- **numerator % denominator** -- divides two values as integers and returns the remainder.
- **числитель % знаменатель** — делит два значения как целые числа и возвращает остаток.

- **value << shift_amt** -- converts both values to 32 bit integers, bitwise left shifts the first value by the second. Note that shifts by more than 32 or less than 0 produce undefined results. -- *REAPER 4.111+*
- **value << shift_amt** — преобразует оба значения в 32-разрядные целые числа, побитово сдвигает влево первое значение на количество бит, указанных во втором. Обратите внимание, что сдвиги более чем на 32 или меньше 0 приводят к неопределенным результатам. — *REAPER 4.111+*

- **value >> shift_amt** -- converts both values to 32 bit integers, bitwise right shifts the first value by the second, with sign-extension (negative values of y produce non-positive results). Note that shifts by more than 32 or less than 0 produce undefined results. -- *REAPER 4.111+*
- **value >> shift_amt** — преобразует оба значения в 32-разрядные целые числа, побитово сдвигает value вправо на количество бит, указанных в shift_amt, с расширением знака (отрицательные значения y приводят к неположительным результатам). Обратите внимание, что сдвиги более чем на 32 или меньше 0 проиводят к неопределенным результатам. — *REAPER 4.111+*

- **value / divisor** -- divides two values and returns the quotient.
- **значение / делитель** — делит два значения и возвращает частное.
- **value * another_value** -- multiplies two values and returns the product.
- **значение * другое_значение** — умножает два значения и возвращает

произведение.

- **value - another_value** -- subtracts two values and returns the difference.
- **значение * другое_значение** — вычитает два значения и возвращает разницу.
- **value + another_value** -- adds two values and returns the sum.
- **значение + другое_значение** — складывает два значения и возвращает сумму.
- **a | b** -- converts both values to integer, and returns bitwise **OR** of values.
- **a | b** — преобразует оба значения в целое, и возвращает побитовое ИЛИ значений.
- **a & b** -- converts both values to integer, and returns bitwise AND of values.
- **a & b** — преобразует оба значения в целое, и возвращает побитовое **И** значений.
- **a ~ b** -- converts both values to 32 bit integers, bitwise XOR the values. — *REAPER 4.25+*
- **a ~ b** — преобразует оба значения в 32 разрядные целые числа побитового **XOR** значения. — *REAPER 4.25+*.
- **value1 == value2** -- compares two values, returns 1 if difference is less than 0.00001, 0 if not.
- **значение1 == значение2** — сравнивает два значения, возвращает 1, если разница меньше 0.00001, 0 — если нет.
- **value1 === value2** -- compares two values, returns 1 if exactly equal, 0 if not. -- *REAPER 4.53+*
- **значение1 === значение2** — сравнивает два значения, возвращает 1, если в точности равны, 0 — если нет. — *REAPER 4.53+*
- **value1 != value2** -- compares two values, returns 0 if difference is less than 0.00001, 1 if not.
- **значение1 = значение2** — сравнивает два значения, возвращает 0, если разница меньше 0.00001, 1 — если нет.
- **value1 !== value2** -- compares two values, returns 0 if exactly equal, 1 if not. -- REAPER 4.53+
- **значение1 == значение2** — сравнивает два значения, возвращает 0, если в точности равна, 1 - если нет. — *REAPER 4.53+*
- **value1 < value2** -- compares two values, returns 1 if first parameter is less than second.
- **значение1 < значение2** — сравнивает два значения, возвращает 1, если первый параметр меньше второго.

- **value1 > value2** -- compares two values, returns 1 if first parameter is greater than second.
- **значение1> значение2** — сравнивает два значения, возвращает 1, если

первый параметр больше второго.

- **value1 <= value2** -- compares two values, returns 1 if first is less than or equal to second.
- **значение1 <= значение2** — сравнивает два значения, возвращает 1, если первый параметр меньше или равен второму.
- **value1 >= value2** -- compares two values, returns 1 if first is greater than or equal to second.
- **значение1 >= значение2** — сравнивает два значения, возвращает 1, если первое больше или равно второму.
- **y || z** -- returns logical OR of values. If 'y' is nonzero, 'z' is not evaluated.
- **y || z** — возвращает логическое **OR** значений 'y' и 'z'. Если 'y' не равно нулю, 'z' не вычисляется.
- **y && z** -- returns logical AND of values. If 'y' is zero, 'z' is not evaluated.
- **y && z** — возвращает логическое **AND** значений 'y' и 'z'. Если 'y' равна нулю, 'z' не вычисляется.
- **y ? z** -- how conditional branching is done -- similar to C's if/else.
- **y ? z** — как делать условное ветвление — по аналогии с if/else в языке Си.
- **y ? z : x** --
  If y is non-zero, executes and returns z, otherwise executes and returns x (or 0.0 if ': *x*' is not specified).
  Note that the expressions used can contain multiple statements within parentheses, such as:

```
x % 5 ? (
  f += 1;
  x *= 1.5;
) : (
  f=max(3,f);
  x=0;
);
```

- **y ? z : x** — Если ‘y’ отличен от нуля, выполняется и возвращается ‘z’, в противном случае выполняется и возвращается ‘x’ (или 0.0, если ': x' не указано).

Обратите внимание, что используемые выражения могут содержать несколько выражений в круглых скобках, например:

```
 x % 5 ? (
      f += 1;
      x *= 1.5;
    ) : (
      f=max(3,f);
      x=0;
    );
```

- **y = z** -- assigns the value of 'z' to 'y'. 'z' can be a variable or an expression.
- **y = z** — присваивает значение 'z' переменной 'y'. 'z' может быть переменной или выражением.
- **y *= z** -- multiplies two values and stores the product back into 'y'.
- **y *= z** — умножает два значения и сохраняет результат обратно в 'y'.
- **y /= divisor** -- divides two values and stores the quotient back into 'y'.
- **y /= divisor** — делит два значения и сохраняет частное обратно в 'y'.
- **y %= divisor** -- divides two values as integers and stores the remainder back into 'y'.
- **y %= divisor** — делит два значения как целые числа и сохраняет остаток обратно в 'y'.
- **base ^= exponent** -- raises first parameter to the second parameter-th power, saves back to 'base'
- **base ^= exponent** — возводит первый параметр в степень, указанную во втором, сохраняет обратно в 'base'.
- **y += z** -- adds two values and stores the sum back into 'y'.
- **y += z** — складывает два значения и сохраняет сумму обратно в 'y'.
- **y -= z** -- subtracts 'z' from 'y' and stores the difference into 'y'.
- **y -= z** — вычитает 'z' из 'y' и сохраняет разницу в 'y'.
- **y |= z** -- converts both values to integer, and stores the bitwise OR into 'y'
- **y |= z** — преобразует оба значения в целое, и сохраняет побитовое **ИЛИ** в 'y'
- **y &= z** -- converts both values to integer, and stores the bitwise AND into 'y'
- **y &= z** — преобразует оба значения в целое, и сохраняет побитовое **И** в 'y'
- **y ~= z** -- converts both values to integer, and stores the bitwise XOR into 'y' -- REAPER 4.25+
- **y ~= z** — преобразует оба значения в целое, и сохраняет побитовое **XOR** в 'y' — *REAPER 4.25+*

Some key notes about the above, especially for C programmers:
Некоторые ключевые заметки о выше изложенном, особенно для Си-программистов:

- (and) (vs { }) -- enclose multiple statements, and the value of that expression is the last statement within the block:

```
z = (
    a = 5;
    b = 3;
    a+b;
); // z will be set to 8, for example
```

- Круглые скобки **(** и **)** (в отличие от фигурных **{ }**) — заключают в себя несколько выражений, и значение этого выражения является последней инструкцией в блоке:

```
z = (
    a = 5;
    b = 3;
    a+b;
      ); // например, z будет принимать значение равное 8
```

- Conditional branching is done using the ? or ? : operator, rather than if()/else.

```
 a < 5 ? b = 6; // if a is less than 5, set b to 6
 a < 5 ? b = 6 : c = 7; // if a is less than 5, set b to 6, otherwise set c to 7
 a < 5 ? ( // if a is less than 5, set b to 6 and c to 7
   b = 6;
   c = 7;
  );
```

- Условное ветвление делается с помощью операторов **?** или **? :**, а не if()/else.

```
a < 5 ? b = 6; // если a меньше 5, установить b = 6
 a < 5 ? b = 6 : c = 7; // если a меньше 5, установить b = 6, в противном случае установить c = 7
 a < 5 ? ( // если a меньше 5, установить b = 6 и c = 7
   b = 6;
   c = 7;
  );
```

- The ? and ?: operators can also be used as the **lvalue** of expressions:

```
(a < 5 ? b : c) = 8; // if a is less than 5, set b to 8, otherwise set c to 8
```

- Операторы **?** и **? :** могут быть также использованы в левой части выражений:

```
(a < 5 ? b : c) = 8; // если a меньше 5, тогда b = 8, иначе c = 8
```

Simple math functions
Простые математические функции

- **sin(angle)** -- returns the Sine of the angle specified (specified in radians -- to convert from degrees to radians, multiply by $pi/180, or 0.017453)
- **sin(angle)** — вычисляет синус указанного угла (указанного в радианах - для преобразования из градусов в радианы, умножьте на $pi/180 или 0,017453)
- **cos(angle)** -- returns the Cosine of the angle specified (specified in radians).
- **cos(angle)** — вычисляет косинус указанного угла (указанный в радианах).
- **tan(angle)** -- returns the Tangent of the angle specified (specified in radians).
- **tan(angle)** — вычисляет тангенс указанного угла (указанный в радианах).
- **asin(x)** -- returns the Arc Sine of the value specified (return value is in radians).
- **asin(x)** — вычисляет арксинус указанного значения (вычисляет в радианах).
- **acos(x)** -- returns the Arc Cosine of the value specified (return value is in radians).
- **acos(x)** — вычисляет арккосинус указанного значения (вычисляет в радианах).

- **atan(x)** -- returns the Arc Tangent of the value specified (return value is in radians).
- **atan(x)** — вычисляет арктангенс указанного значения (вычисляет в радианах).
- **atan2(x,y)** -- returns the Arc Tangent of x divided by y (return value is in radians).
- **atan2(x,y)** — вычисляет арктангенс x делённого на y (вычисляет в радианах).
- **sqr(x)** -- returns the square of the parameter (similar to x*x, though only evaluating x once).
- **sqr(x)** — вычисляет квадрат параметра (по аналогии с x*x, но только изменяясь x раз).
- **sqrt(x)** -- returns the square root of the parameter.
- **sqrt(x)** — вычисляет квадратный корень параметра.
- **pow(x,y)** -- returns the first parameter raised to the second parameter-th power. Identical in behavior and performance to the ^ operator.
- **pow(x,y)** — вычисляет первый параметр, возведенный в степень второго параметра. Идентичные в поведении и представлении оператору ^.
- **exp(x)** -- returns the number e (approx 2.718) raised to the parameter-th power. This function is significantly faster than pow() or the ^ operator
- **exp(x)** — вычисляет число e (приблизительно 2,718), возведенное в степень параметра. Эта функция значительно быстрее, чем pow() или оператор ^
- **log(x)** -- returns the natural logarithm (base e) of the parameter.
- **log(x)** — вычисляет натуральный логарифм (базис e) параметра.

- **log10(x)** -- returns the logarithm (base 10) of the parameter.
- **log10(x)** — вычисляет логарифм (базис 10) параметра.
- **abs(x)** -- returns the absolute value of the parameter.
- **abs(x)** — вычисляет абсолютное значение параметра.
- **min(x,y)** -- returns the minimum value of the two parameters.
- **min(x,y)** — вычисляет минимальное значение из двух параметров.
- **max(x,y)** -- returns the maximum value of the two parameters.
- **max(x,y)** — вычисляет максимальное значение из двух параметров.
- **sign(x)** -- returns the sign of the parameter (-1, 0, or 1).
- **sign(x)** — вычисляет знак параметра (-1, 0 или 1).
- **rand(x)** -- returns a psuedorandom number between 0 and the parameter.
- **rand(x)** — вычисляет псевдослучайное число между 0 и параметром.

- **floor(x)** -- rounds the value to the lowest integer possible (floor(3.9)==3, floor(-3.1)==-4).
- **floor(x)** — округляет значение до меньшего целого возможного (floor(3.9)==3, floor(-3.1)==-4).
- **ceil(x)** -- rounds the value to the highest integer possible (ceil(3.1)==4, ceil(-3.9)==-3).
- **ceil(x)** — округляет значение до самого большего целого возможного (ceil(3.1)==4, ceil(-3.9)==-3).
- **invsqrt(x)** -- returns a fast inverse square root (1/sqrt(x)) approximation of the parameter.
- **invsqrt(x)** — вычисляет быструю апроксимацию обратного квадратного корня (1/sqrt(x)) параметра.

Loops Циклы

Looping is supported in JSFX via the following functions:
Зацикливание поддерживается в JSFX с помощью следующих функций:

- **loop(count,code)**

```
loop(32,
    r += b;
    b = var * 1.5;
);
```

Evaluates the first parameter once in order to determine a loop count. If the loop count is less than 1, the second parameter is not evaluated.
Be careful with specifying large values for the first parameter -- it is possible to hang your effect for long periods of time. In the interest of avoiding common runtime hangs, the loop count will be limited to approximately 1,000,000: if you need a loop with more iterations, you may wish to reconsider your design (or as a last resort, nest loops).

The first parameter is only evaluated once (so modifying it within the code will have no effect on the number of loops). For a loop of indeterminate length, see while() below.

- **loop(count,code)**

```
loop(32,
    r += b;
    b = var * 1.5;
);
```

Проверяет первый параметр один раз, чтобы определить количество циклов. Если количество циклов меньше 1, второй параметр не вычисляется.
Будьте осторожны с указанием больших значений для первого параметра - можно повесить свое действие на длительный период времени.
Во избежание общих зависаний во время выполнения, количество циклов будет ограничено примерно 1 000 000: если вам нужен loop с большей итераций, вы можете пересмотреть свой проект (или в крайнем случае, вложенные loops).
Первый параметр проверяется только один раз (поэтому его изменения в коде не будет иметь никакого влияния на количество loops). Для loop неопределенной длины, см. while() ниже.

- **while(code)**

```
while(
    a += b;
    b *= 1.5;
    a < 1000; // as long as a is below 1000, we go again.
);
```

Evaluates the first parameter until the last statement in the code block evaluates to zero.

In the interest of avoiding common runtime hangs, the loop count will be limited to approximately 1,000,000: if you need a loop with more iterations, you may wish to reconsider your design (or as a last resort, nest loops).

- **while(code)**

```
while(
    a += b;
    b *= 1.5;
    a < 1000; // до тех пор, пока находится ниже 1000, мы идем снова.
);
```

выполняет код, пока последний оператор в блоке кода не достигнет ноля.

Во избежание общих зависаний во время выполнения, количество циклов будет ограничено примерно 1 000 000: если вам нужен loop с большей итераций, вы можете пересмотреть свой проект (или в крайнем случае, вложенные loops).

- **while(condition) (code)** -- *REAPER 4.59+*

```
    while ( a < 1000 ) (
      a += b;
      b *= 1.5;
    );
```

Evaluates the parameter, and if nonzero, evaluates the following code block, and repeats. This is similar to a C style while() construct.

In the interest of avoiding common runtime hangs, the repeat count will be limited to approximately 1,000,000: if you need a loop with more iterations, you may wish to reconsider your design (or as a last resort, nest loops).

- **while(condition) (code)** — *REAPER 4.59+*

```
    while ( a < 1000 ) (
      a += b;
      b *= 1.5;
    );
```

Проверяет параметр, и если не равен нулю, выполняет следующий блок кода, и повторы. Это похоже на Си стиль while() конструкций.

Во избежание общих зависаний во время выполнения, количество циклов будет ограничено примерно 1 000 000: если вам нужен loop с большей итераций, вы можете пересмотреть свой проект (или в крайнем случае, вложенные loops).

Special Variables
Специальные переменные

Basic Functionality:
Основные функциональные возможности:

- **spl0, spl1 ... spl63**
  Context: @sample only
  Usage: read/write
  The variables spl0 and spl1 represent the current left and right samples in @sample code.
  The normal +0dB range is -1.0 .. 1.0, but overs are allowed (and will eventually be clipped if not reduced by a later effect).
  On a very basic level, these values represent the speaker position at the point in time, but if you need more information you should do more research on PCM audio.
  If the effect is operating on a track that has more than 2 channels, then spl2..splN will be set with those channels values as well. If you do not modify a splX variable, it will be passed through unmodified.

  See also spl(x) below, though splX is generally slightly faster than spl(X)

- **spl0, spl1…spl63**
  Контекст: только @sample
  Использование: чтение/запись
  Переменные spl0 и spl1 представляют текущие левый и правый сэмплы в разделе кода @sample.
  Нормальный диапазон +0 дБ -1.0 .. 1.0, но шире также допускается (и в конечном итоге будет обрезано, если не уменьшится последующим эффектом).
  На самом базовом уровне, эти значения представляют позицию громкоговорителя в момент времени, но если вам нужна дополнительная информация, вы должны делать больше исследований на PCM аудио. ...вы должны обратиться к фундаментальным источникам.
  Если эффект работает на треке, который имеет более чем 2 канала, соответственно и spl2..splN также будут установлены с теми же значениями каналов. Если вы не измените переменную splX, он будет передан насквозь без изменений.
  См. также далее spl(x), хотя splX, как правило, немного быстрее, чем spl(X).

- **spl(channelindex)** -- *REAPER 2.018+*
  Context: @sample only
  If you wish to programmatically choose which sample to access, use this function (rather than splX). This is slightly slower than splX, however has the advantage that you can do spl(variable) (enabling easily configurable channel mappings).
  Valid syntaxes include:

  ```
  spl(channelindex)=somevalue;
  spl(5)+=spl(3);
  ```

- **spl(индекс канала)** — REAPER 2.018+
  Контекст: только @sample
  Если вы хотите программно выбирать к сэмплу какого канала обратиться, используйте эту функцию (а не splX). Это немного медленнее, чем splX, однако имеет то преимущество, что вы можете выполнить spl(переменная) (обеспечивая легко настраиваемые карты каналов). Допустимые синтаксисы включают в себя:

  ```
  spl(индекс канала) = некоторое значение;
  spl(5)+=spl(3);
  ```

- **slider1, slider2, ... slider64**
  Context: available everywhere
  Usage: read/write
  The variables slider1, slider2, ... slider64 allow interaction between the user and the effect, allowing the effects parameters to be adjusted by the user and likewise allow the effect to modify the parameters shown to the user (if you modify sliderX in a context other than @slider then you should callsliderchange(sliderX) to notify JS to refresh the control).

  The values of these sliders are purely effect-defined, and will be shown to the user, as well as tweaked by the user.

- **slider1, slider2, ... slider64**
  Контекст: везде
  Использование: чтение/запись

Переменные slider1, slider2, … slider64 позволяют взаимодействие между пользователем и эффектом, позволяя пользователю изменять параметры эффектов, и также позволяют видеть пользователю изменения эффектов (если вы измените sliderX в контексте, отличном от @slider, то вы должны вызвать функцию sliderchange(sliderX), чтобы уведомить JS о том, что надо обновить элемент управления).

Значения этих ползунков являются чисто эффект-определяющими, и будет показано пользователю, а также меняться пользователем.
Вариант:
Значения этих переменных (slider1, slider2, ... slider64) являются сугубо эффект-определяемыми, и будут показаны пользователю, также как и доступны к регулировке пользователем.

- **trigger**
  Context: @block, @sample
  Usage: read/write
  The trigger variable provides a facility for triggering effects.

  If this variable is used in an effect, the UI will show 10 trigger buttons, which when checked will result in the appropriate bit being set in this variable.

  For example, to check for trigger 5 (triggered also by the key '5' on the keyboard):
  ```
  isourtrig = trigger & (2^5);
  ```
  Conversely, to set trigger 5:
  ```
  trigger |= 2^5;
  ```
  Or, to clear trigger 5:
  ```
  trigger & (2^5) ? trigger -= 2^5;
  ```
  It is recommended that you use this variable in @block, but only sparingly in @sample.

- **trigger**
  Контекст: @block, @sample
  Использование: чтение/запись
  Переменная trigger обеспечивает возможность для срабатывания эффектов.

  Если эта переменная используется в эффекте, UI отобразит 10 кнопок триггеров, нажатие на которые приведет к установке соответствующего бита в этой переменной.

Например, для проверки триггера 5 (запускается также клавишей '5' на клавиатуре):

```
isourtrig = trigger & (2^5);
```

И наоборот, чтобы установить триггер 5:

```
trigger |= 2^5;
```

Или, чтобы очистить триггер 5:

```
trigger & (2^5) ? trigger -= 2^5;
```

Рекомендуется использовать эту переменную в разделе @block, и очень умеренно в разделе @sample.

Audio and transport state: Состояние аудио и транспорта:

- **srate**
  Context: available everywhere
  Usage: read-only
  The srate variable is set by the system to whatever the current sampling frequency is set to (usually 44100 to 192000). Generally speaking your @init code section will be called when this changes, though it's probably a good idea not to depend too much on that.
  Контекст: везде
  Использование: только для чтения
  Переменная srate устанавливается системой независимо от того, как установлена текущая частота дискретизации (обычно от 44100 до 192000). Вообще говоря, ваш раздел кода @init будет вызываться, когда она изменится, хотя это, вероятно, хорошая идея, чтобы не слишком зависеть от этого.

- **num_ch**
  Context: most contexts (see comments)
  Usage: read-only
  Specifies the number of channels available (usually 2). Note however splXX are still available even if this count is less, their inputs/outputs are just ignored. You can change the channel count available via in_pin:/out_pin: lines.
  Контекст: большинство контекстов (см. комментарии)
  Использование: только для чтения
  Определяет количество доступных каналов (обычно 2). Заметим, однако, что

splXX по-прежнему доступны, даже если это число меньше — их входы/выходы просто игнорируются. Вы можете изменить количество каналов, доступных через строки in_pin:/out_pin:.

- **samplesblock**
  Context: most contexts (see comments)
  Usage: read-only
  The samplesblock variable can be used within @block code to see how many samples will come before the next @block call. It may also be valid in other contexts (though your code should handle invalid values in other contexts with grace).
  Контекст: большинство контекстов (см. комментарии)
  Использование: только для чтения
  Переменная samplesblock может быть использована в разделе кода @block, чтобы увидеть, сколько семплов придет до следующего вызова @block. Она также может быть действительной в других контекстах (однако ваш код должен изящно обрабатывать недопустимые значения в других контекстах).

- **tempo**
  Context: @block, @sample
  Usage: read-only
  The current project tempo, in "bpm". An example value would be 120.0.
  Контекст: @block, @sample
  Использование: только для чтения
  Темп текущего проекта в "bpm". Для примера величина может быть 120.0.

- **play_state**
  Context: @block, @sample
  Usage: read-only
  The current playback state of REAPER (0=stopped, <0=error, 1=playing, 2=paused, 5=recording, 6=record paused).
  Контекст: @block, @sample
  Использование: только для чтения
  Текущее состояние проигрывания REAPER (0=остановлено, <0=ошибка, 1=воспроизведение, 2=пауза, 5=запись, 6=пауза записи).

- **play_position**
  Context: @block, @sample

Usage: read-only
The current playback position in REAPER (as of last @block), in seconds.
Контекст: @block, @sample
Использование: только для чтения
Текущее положение воспроизведения в REAPER (по состоянию на последний @block), в секундах.

- **beat_position**
  Context: @block, @sample
  Usage: read-only
  The current playback position (as of last @block) in REAPER, in beats (beats = quarternotes in /4 time signatures).
  Контекст: @block, @sample
  Использование: только для чтения
  Текущее положение воспроизведения в REAPER (по состоянию на последний @block), в битах (бит = четвертной ноте при размере ?/4).

Extended Functionality: Расширенные функциональные возможности:

- **ext_noinit**
  Context: @init only
  Set this variable to 1.0 in your @init section if you do not wish for @init to be called (and variables/RAM to be possibly cleared) on every transport start.
  Контекст: только @init
  Установите эту переменную в 1.0 в вашем разделе @init, если не хотите, чтобы @init вызывался (переменные/RAM возможно будут очищены) при каждом старте транспорта.

- **ext_nodenorm**
  Context: @init only
  Set this variable to 1.0 in your @init section if you do not wish to have anti-denormal noise added to input.
  Контекст: только @init
  Установите эту переменную в 1.0 в вашем разделе @init, если не хотите, чтобы ко входу был добавлен анти-аномальный шум.

- **reg00-reg99**
  Context: available everywhere
  Usage: read/write

The 100 variables reg00, reg01, reg02, .. reg99 are shared across all effects and can be used for inter-effect communication. Their use should be documented in the effect descriptions to avoid collisions with other effects. regXX aliases to _global.regXX.
Контекст: везде
Использование: чтение/запись
100 переменных reg00, reg01, reg02, .. reg99 являются общими для всех эффектов и могут быть использованы для связи между эффектами. Их использование должно быть задокументировано в описании эффекта, чтобы избежать конфликтов с другими эффектами. regXX являются псевдонимами для _global.regXX.

- **_global.***
  Context: available everywhere
  Usage: read/write
  Like regXX, _global.* are variables shared between all instances of all effects.
  Контекст: везде
  Использование: чтение/запись
  Как regXX, _global.* это переменные, общие для всех экземпляров всех эффектов.

Delay Compensation (PDC): Компенсация задержки (PDC):

- **pdc_delay**
  Context: @block, @slider
  Usage: read-write
  The current delay added by the plug-in, in samples. Note that you shouldnt change this too often. This specifies the amount of the delay that should be compensated, however you need to set the pdc_bot_ch and pdc_top_ch below to tell JS which channels should be compensated.
  Контекст: @block, @slider
  Использование: чтение/запись
  Текущая задержка, добавленная плагином, в семплах. Обратите внимание, что вам не следует изменять ее слишком часто. Эта переменная(?) определяет величину задержки, которая должна быть компенсирована, однако вы должны установить pdc_bot_ch и pdc_top_ch (см. ниже), чтобы указать JS, какие каналы должны быть компенсированы.

- **pdc_bot_ch, pdc_top_ch**

Context: @block, @slider
Usage: read-write
The channels that are delayed by pdc_delay. For example:

```
pdc_bot_ch=0; pdc_top_ch=2; // delays the first two channels (spl0/spl1).
pdc_bot_ch=2; pdc_top_ch=5; // delays channels spl2,spl3, and spl4.
```

(this is provided so that channels you dont delay can be properly synchronized by the host).
Контекст: @block, @slider
Использование: чтение/запись
Каналы, задержанные с помощью pdc_delay. Например:

```
pdc_bot_ch = 0; pdc_top_ch = 2 // задерживает первые два канала (spl0 и spl1).
pdc_bot_ch = 2; pdc_top_ch = 5; // задерживает каналы spl2, spl3 и spl4.
```

(это обеспечивает то, что каналы, которые вы не задерживаете, могут быть правильно синхронизированы хостом).

- **pdc_midi**
  Context: @block, @slider
  Usage: read-write
  If set to 1.0, this will delay compensate MIDI as well as any specified audio channels.
  Контекст: @block, @slider
  Использование: чтение/запись
  Если установлено значение 1.0, это приведет к задержке компенсации MIDI, а также любых указанных аудиоканалов.

Graphics and Mouse: Графика и мышь:

- gfx_* and mouse_* are also defined for use in @gfx code.
- gfx_* and mouse_* также определены для использования в разделе кода @gfx.

MIDI Functions Функции MIDI

The following functions can be used in @block or @sample sections to send and receive MIDI.
Следующие функции могут быть использованы в разделах @block или @sample для отправки и получения MIDI-сообщений.

Note that if a JS effect calls midirecv(), MIDI will not be passed through unless you also midisend() the received messages.
Обратите внимание, что если JS-эффект вызывает функцию midirecv(), MIDI-сообщения не пройдут, пока вы также не отправите(?) полученные сообщения функцией midisend().

Offset is offset from current block, in samples. msg1 is status byte, msg23 is second (and third if available) data bytes, second byte is low 8 bits (msg23&$xff), third byte is next 8 bits (msg23/256)&$xff.
Offset это смещение от текущего блока, в семплах. **msg1** это байт состояния, **msg23** это второй (и третий, при наличии) байт данных, второй байт это младшие 8 бит (msg23&$xff), третий байт это следующие 8 бит (msg23/256)&$xff.

- **midisend(offset,msg1,msg23)**

```
midisend(0,$x90 + 0,69|(127*256)); // посылает ноту 69 с велосити 127 (max) в канал 0
```

- **midirecv(offset,msg1,msg23)**

```
midirecv(offset,msg1,msg23);
(msg1&$xF0)==$x90 ? ( /* note on! */ )
```

Note: midirecv returns msg1 as a return value, as well.
Примечание: midirecv также возвращает msg1 в качестве возвращаемого значения.

So you can do things like:
Таким образом, вы можете делать следующие вещи:

```
while(
  midirecv(offs,msg1,msg23) ?
  (
    midisend(offs,msg1,msg23);
```

```
        );
    );
```

- **midisyx(offset,msgptr,len)**
  Пример:

```
    buf[0] = $xAA|0;
    buf[1] = $xBB|0;
    midisyx(offset,buf,2);  // посылает sysex: F0 AA BB F7
```

File I/O and Serialization Файловый ввод/вывод и сериализация

The following functions can be used in the @serialize section or in other sections.
Следующие функции могут быть использованы в разделе @serialize или в других разделах.

Using with @serialize:
Использование в разделе @serialize:
Pass 0 as a handle to various file_*() functions, but do not call file_open() or file_close(). Simple @serialize code will often appear the same for read and write, as file_var(0,x) will read/write x depending on the mode. If you want to have different logic per mode, you can check file_avail(0)>=0 to determine if it is in read mode.
Передавайте 0 как дескриптор различным file_*() функциям, но не вызывайте file_open() или file_close(). Простой код @serialize для чтения и записи часто выглядит одинаково, так же, как file_var(0,x) будет читать/записывать x в зависимости от режима. Если вы хотите иметь другую логику для каждого режима, вы можете проверить file_avail(0)>=0 для определения того, находится ли он в режиме чтения.

Using in other sections:
Использование в других разделах:
file_open() and file_close() can be used to open files for reading in any section.
file_open () и file_close () могут быть использованы в любом разделе для открытия файлов для чтения.

- **file_open(index or slider)**
  Пример:

  ```
  filename:0,myfile.wav
  handle = file_open(0);
  ```

  Пример:

  ```
  slider1:/mydata:mydef.wav:WAV File
  handle = file_open(slider1);
  ```

  Пример (*REAPER 4.59+*):

  ```
  handle = file_open(string);
  ```

Opens a file from either the effect filename list or from a file slider, or from a string (*REAPER 4.59+*). Once open, you may use all of the file functions available. Be sure to close the file handle when done with it, using file_close(). The search path for finding files depends on the method used, but generally speaking in 4.59+ it will look in the same path as the current effect, then in the JS Data/ directory.

Открывает файл либо из списка имен файлов эффектов, или из слайдера файла, или из строки (*REAPER 4.59+*). После открытия, вы можете использовать все доступные файловые функции. Обязательно закройте дескриптор файла, когда поработали с ним, используя file_close(). Путь поиска для нахождения файлов зависит от используемого метода, но, вообще говоря, в 4.59+ поиск будет по тому же пути, что и текущий эффект, в каталоге JS Data/.

If file_open() fails, it will return < 0 (usually -1).
Если функция file_open() завершается неудачно, то она вернет значение < 0 (обычно -1).

- **file_close(handle)**
  Пример:

```
file_close(handle);
```

  Closes a file opened with file_open().
  Закрывает файл, открытый с помощью file_open().

- **file_rewind(handle)**
  Пример:

```
file_rewind(handle);
```

  Use this to rewind the current file to the beginning, to re-read the file etc.
  Используйте эту функцию чтобы “перемотать” текущий файл в начало, чтобы перечитать файл и т.д.

- **file_var(handle,variable)**
  Пример:

```
file_var(handle,myVar);
```

This reads (or writes if in a @serialize write) the variable from(to) the current file.
Читает (или пишет, если в разделе @serialize указана запись) переменную variable из(в) текущего файла.

- **file_mem(handle,offset, length)**
  Пример:

```
amt=file_mem(handle,offset,len);
```

  This reads (or writes) the block of local memory from(to) the current file. Returns the actual number of items read (or written).
  Читает (или записывает) блок локальной памяти из (в) текущего файла. Возвращает фактическое количество прочитанных (или записанных) элементов.

- **file_avail(handle)**
  Пример:

```
len=file_avail(handle);
```

  Returns the number of items remaining in the file, if it is in read mode. Returns < 0 if in write mode. If the file is in text mode (file_text(handle) returns TRUE), then the return value is simply 0 if EOF, 1 if not EOF.
  Возвращает количество элементов, оставшихся в файле, если он находится в режиме чтения. Возвращает <0, если файл в режиме записи. Если файл находится в текстовом режиме (т.е. file_text(handle) возвращает TRUE), то возвращает значение просто 0 если достигнут конец файла (EOF), и 1 если нет.

- **file_riff(handle,nch,samplrate)**
  Пример:

```
file_riff(handle,nch,samplrate);
nch ? file_mem(handle,0,file_avail(0));
```

  If the file was a RIFF WAV file, or a valid .OGG Vorbis file, this will set the first parameter to the number of channels, and the second to the samplerate.
  Если файл, указанный в handle, был RIFF WAV-файл, или OGG Vorbis-файл, то установит первый параметр (nch) равным числу каналов, а второй (samplrate)

равным частоте сэмплирования.

- **file_text(handle,istext)**
  Пример:

```
istext=file_text(handle);
istext ? use_diff_avail syntax;
```

If the file was a text file (and ended in .txt), this will return 1. If you need to use different file_avail() logic for text files (you often will), you can query it this way.
Если файл был текстовый (и заканчивался на .txt), эта функция вернет 1. Если вам нужно использовать другую логику file_avail() для текстовых файлов (что будете делать часто), вы можете запросить это так, как показано в примере.

Text file notes
Примечания к работе с текстовыми файлами
Note that if in an extended file-slider code section, and the extension of the file is .txt, it will read one line at a time, ignoring non-number lines. Note that file_avail() should be called to check for EOF after each read, and if it returns 0, the last file_var() should be ignored.
Заметим, что если в расширенном разделе кода file-slider, и расширение файла .txt, будет считываться одна строка за один раз, игнорируя нечисловые строки. Обратите внимание, что file_avail() должна вызываться для проверки EOF (конца файла) после каждого чтения, и если она возвращает 0, последнюю file_var() следует игнорировать.

You can also use file_mem(offs,bignum) and it will read the maximum available.
Вы также можете использовать file_mem(offset,bignum) — будет считываться доступный максимум.

The format of each line in the text file can be either a floating point number, a binary number beginning with 'b', i.e. b0101010111, or a hexadecimal number beginning with 'x', i.e. xDEADF000.
Формат каждой строки в текстовом файле может быть либо число с плавающей точкой, либо двоичное число, начинающееся с 'b', то есть b0101010111, либо шестнадцатеричное число, начинающееся с 'x', т.е. xDEADF000.

Additionally text files can create their own symbolic constants (using =), and combine them using basic +, -, |, & etc operators.

Дополнительно текстовые файлы могут создавать свои собственные символические константы (с помощью =) и комбинировать их с помощью базовых операторов +, -, |, & и т.п.

- **file_string(handle,str)** — *REAPER 4.59+*
  Reads or writes a string from/to the file handle. If operating on a normal file, the string will be a line of text (possibly including newline or other characters). If in @serialize, the string will be encoded as a blob with length, which means that it is binary-safe (you can include NUL characters within the string etc).
  Читает или записывает строку из/в файла handle. При работе с обычным файлом строка будет текстовой (возможно, включение новой строки или другие символы). Если эта функция вызывается в разделе в разделе кода @serialize, то строка будет закодирована как blob (Массив двоичных данных. В СУБД BLOB — специальный тип данных, предназначенный, в первую очередь, для хранения изображений, аудио и видео, а также компилированного программного кода)с длиной, а это значит, что она будет binary-safe (т.е. вы можете включать в строку символы NUL и т.п.).

Memory/Slider/FFT/MDCT Functions

- **mdct(start_index, size), imdct(start_index, size)**
  Example:

```
mdct(0, 512);
```

  Performs a modified DCT (or inverse in the case of imdct()) on the data in the local memory buffer at the offset specified by the first parameter. The second parameter controls the size of the MDCT, and it MUST be one of the following: 64, 128, 256, 512, 1024, 2048, or 4096. The MDCT takes the number of inputs provided, and replaces the first half of them with the results. The IMDCT takes size/2 inputs, and gives size results.

  Note that the MDCT must NOT cross a 65,536 item boundary, so be sure to specify the offset accordingly.

  The MDCT/IMDCT provided also provide windowing, so your code is not required to window the overlapped results, but simply add them. See the example effects for more information.

- **fft(start_index, size), ifft(start_index, size)**
  fft_permute(index,size), fft_ipermute(index,size)
  Example:

```
buffer=0;
fft(buffer, 512);
fft_permute(buffer, 512);
buffer[32]=0;
fft_ipermute(buffer, 512);
ifft(buffer, 512);
// need to scale output by 1/512.0, too.
```

  Performs a FFT (or inverse in the case of ifft()) on the data in the local memory buffer at the offset specified by the first parameter. The size of the FFT is specified by the second parameter, which must be 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, or 32768. The outputs are permuted, so if you plan to use them in-order, call **fft_permute(idx, size)** before and **fft_ipermute(idx,size)** after your in-order use. Your inputs or outputs will need to be scaled down by 1/size, if used.

Note that the FFT/IFFT require real/imaginary input pairs (so a 256 point FFT actually works with 512 items), while the real FFT/IFFT (rfft() and irfft(), below) provide a real-FFT.

Note that the FFT/IFFT must NOT cross a 65,536 item boundary, so be sure to specify the offset accordingly.

- **convolve_c(dest,src,size)**
  Used to convolve two buffers, typically after FFTing them. convolve_c works with complex numbers. The sizes specify number of items (the number of complex number pairs).
  Note that the convolution must NOT cross a 65,536 item boundary, so be sure to specify the offset accordingly.

  Используется для свертки двух буферов, как правило, после применения к ним операции быстрого преобразования Фурье (FFT). convolve_c работает с комплексными числами. Размеры указывают количество элементов (число комплексных пар чисел).
  Обратите внимание, что свертка НЕ должна пересекать границы в 65536 элементов, так что не забудьте указать соответственно смещение.

Memory Utility
Утилиты памяти

- **freembuf(top)**
  The freembuf() function provides a facility for you to notify the memory manager that you are no longer using a portion of the local memory buffer.

  For example, if the user changed a parameter on your effect halving your memory requirements, you should use the lowest indices possible, and call this function with the highest index you are using plus 1, i.e. if you are using 128,000 items, you should call freembuf(128001); If you are no longer using any memory, you should call freembuf(0);

  Note that calling this does not guarantee that the memory is freed or cleared, it just provides a hint that it is OK to free it.

- **freembuf(top)**
  Функция freembuf () обеспечивает возможность для Вас предупреждение менеджера памяти, что вы больше не используете часть локального буфера памяти.

  Например, если пользователь изменил параметр на вашем эффекте так, что он потребовал вдвое больше памяти, вы должны использовать самые низкие показатели, и вызвать эту функцию с наибольшим индексом, который вы используете плюс 1, то есть, если вы используете 128000 пунктов, вы должны вызвать freembuf (128001); Если вы больше не используете любую память, вы должны вызвать freembuf (0);

  Обратите внимание, что вызов не гарантирует, что память освобождается или снята, он просто подсказывает, что это ВОЗМОЖНО, чтобы освободить его.

- **memcpy(dest,source,length)**
  The memcpy() function provides the ability to quickly copy regions of the local memory buffer. The regions may overlap, but neither region may cross a 65,536 item boundary (they may be on different pages, however).

- **memcpy(назначение, источник, длина)**
  Функция memcpy() предоставляет возможность быстрого копирования регионов локального буфера памяти. Регионы могут пересекаться, но ни один регион не может пересечь границу в 65536 пунктов (при этом они могут быть на разных страницах).

- **memset(dest,value,length)**
  The memset() function provides the ability to quickly set a region of the local memory buffer to a particular value. Unlike memcpy(), this region may be of any length and cross any boundaries.

- **memset(назначение, источник, длина)**
  Функция memset() предоставляет возможность быстро установить область локального буфера памяти на конкретное значение. В отличие от memcpy(), этот регион может быть любой длины и пересекать любые границы.

- **Slider Functions**
  For these functions, the parameter can be the variables slider1-sliderN, in which

case that slider is refreshed. Otherwise, it can be a bitmask of which sliders have changed, where 1 would be the first slider, 2 would be the second, 4 would be the third, 32768 being the 16th slider, and so on.

- **Slider Функции**
  Для этих функций, в качестве параметра могут выступать переменные slider1-sliderN, в этом случае слайдер обновляется. В противном случае, это может быть битовая маска, в соответствии с которой слайдеры изменяются, где 1 будет первый слайдер, 2 будет второй, 4 будет третий, 32768 будет 16-й слайдер, и так далее.
  Пример:
  0000000000000001(bin) = 1(dec) — 1-й слайдер
  0000000000000010(bin) = 2(dec) — 2-й слайдер
  0000000000000100(bin) = 4(dec) — 3-й слайдер
  …
  1000000000000000(bin) = 32768(dec) — 16-й слайдер

- **sliderchange(mask or sliderX)**
  Example:

```
sliderchange(slider4);
```

  or

```
sliderchange(2 ^ sliderindex);
```

  The sliderchange() function provides a facility for you to notify REAPER/JS that you have changed a sliderX variable so that it can update the display. This function is not necessary to call from the @slider code section, it is provided so that other code sections can update the sliders. Note that this function only updates the display, it does not send an automation message to the host.

  Функция sliderchange() обеспечивает возможность для Вас, чтобы уведомить REAPER / JS, что вы изменили переменную sliderX так что он может обновить экран. Эта функция не является необходимой для сообщения из секции @slider кода, предусмотрено так, чтобы другие участки кода могли обновить sliders. Обратите внимание, что эта функция обрабатывает только обновление экрана, она не посылает сообщение об автоматизации в хост.

- **slider_automate(mask or sliderX)**
  Example:

  ```
  slider_automate(slider4);
  ```
  or

  ```
  slider_automate(2 ^ sliderindex);
  ```

  The slider_automate() function provides a facility for you to notify REAPER/JS that you have changed a sliderX variable so that it can update the display, and record the move as automation. This function is not necessary to call from the @slider code section, it is provided so that other code sections can write programmatic automation messages.

  Функция slider_automate() обеспечивает возможность для Вас, чтобы уведомить REAPER / JS, что вы изменили переменную sliderX так что он может обновить экран, и записывать этот шаг как автоматизации. Эта функция не является необходимой для сообщения из секции @slider кода, предусмотрено так, чтобы другие участки кода могли написать программные сообщения автоматизации.

Strings Строки

Note: the functionality available in this section requires REAPER 4.59+
Примечание: функциональные возможности, доступные в данном разделе требуют (наличия) REAPER 4.59+

Strings can be specified as literals using quotes, such as "This is a test string". Much of the syntax mirrors that of C: you must escape quotes with backslashes to put them in strings ("He said \"hello, world\" to me"), multiple literal strings will be automatically concatenated by the compiler. Unlike C, quotes can span multiple lines. There is a soft limit on the size of each string: attempts to grow a string past about 16KB will result in the string not being modified.

Строки могут быть определены как литералы с использованием кавычек (литералы в кавычках), например, "Это тестовая строка". Большая часть синтаксиса отражает структуру (сходна с синтаксисом?) языка C(Си?): чтобы поместить кавычки внутри строки вы должны экранировать их обратной косой чертой ("He said \"hello, world\" to me"), несколько символьных строк будут автоматически объединены компилятором. В отличие от Си, содержимое кавычек может занимать несколько строк(?). Существует программное ограничение на размер каждой строки: попытки сделать размер строки более 16 КБ приведут к тому, что строка не будет изменена.

Strings are always refered to by a number, so one can reference a string using a normal JS variable:
Пока не могу толком красиво придумать :(Смысл вроде понятен, но по-русски красиво пока не получается
В JS на строки всегда ссылаются по номеру(?),...
...так что можно ссылаться на строку с помощью обычной JS-переменной:

```
x = "hello world";
gfx_drawstr(x);
```

Literal strings are immutable (meaning they cannot be modified). If you wish to have mutable strings, you have three choices:
Литеральные строки неизменяемы (то есть они не могут быть изменены). Если вы хотите иметь (использовать?) изменяемые строки, у вас есть три варианта:

- You can use the fixed values of 0-1023:
- Вы можете использовать фиксированные значения 0-1023:

```
x = 50; // string slot 50 строковая последовательность, текстовый фрагмент 50 (это как?)
```

```
strcpy(x, "hello ");
strcat(x, "world");
gfx_drawstr(x);
```

This mode is useful if you need to build or load a table of strings.
Этот режим удобен, если вам нужно построить или загрузить таблицу строк.

- You can use # to get an instance of a temporary string:
- Вы можете использовать символ # для получения экземпляра временной строки: *(это дословно), или* временного экземпляра строки *(?)*

```
x = #;
strcpy(x, "hello ");
strcat(x, "world");
gfx_drawstr(x);
```

Note that the scope of these temporary instances is very limited and unpredictable, and their initial values are undefined.
Следует помнить, что область видимости этих временных экземпляров весьма ограничена и непредсказуема, а их начальные значения не определены.

- Finally, you can use named strings, which are the equivalent of normal variables:
- Наконец, вы можете использовать именованные строки, которые эквивалентны обычным переменным:

```
x = #myString;
strcpy(x, "hello world");
```

The value of named strings is defined to be empty at script load, and to persist throughout the life of your script. There is also a shortcut to assign/append to named strings:
При загрузке скрипта значения именованных строк определяются как пустые, и сохраняются на протяжении всей жизни вашего скрипта. Существует также ярлык (краткий способ?) для присвоения значения/добавления к именованным строкам:

```
#myString = "hello ";  // same as strcpy(#myString, "hello ");
#myString += "world"; // same as strcat(#myString, "world");
#myString = "hello ";  // то же, что и strcpy(#myString, "hello ");
#myString += "world"; // то же, что и strcat(#myString, "world");
```

String functions Строковые функции

- **strlen(str)** -- returns length of string
- **strlen(str)** — возвращает длину строки.

- **strcpy(str, srcstr)** -- copies srcstr into str, returns str
- **strcpy(str, srcstr)** — копирует srcstr в str. Возвращает str.

- **strcat(str, srcstr)** -- appends srcstr to str, returns str
- **strcat(str, srcstr)** — добавляет srcstr к str. Возвращает str.

- **strcmp(str, str2)** -- compares str to str2, case sensitive, returns -1, 0, or 1
- **strcmp(str, str2)** — сравнивает str и str2 с учетом регистра. Возвращает -1, 0, или 1.

- **stricmp(str, str2)** -- compares str to str2, ignoring case, returns -1, 0, or 1
- **stricmp(str, str2)** — сравнивает str и str2 без учета регистра. Возвращает -1, 0, или 1.

- **strncmp(str, str2, maxlen)** -- compares str to str2 up to maxlen bytes, case sensitive, returns -1, 0, or 1
- **strncmp(str, str2, maxlen)** — сравнивает str и str2 вплоть до maxlen байт с учетом регистра. Возвращает -1, 0, или 1.

- **strnicmp(str, str2, maxlen)** -- compares str to str2 up to maxlen bytes, ignoring case, returns -1, 0, or 1
- **strnicmp(str, str2, maxlen)** — сравнивает str и str2 вплоть до maxlen байт без учета регистра. Возвращает -1, 0, или 1.

- **strncpy(str, srcstr, maxlen)** -- copies srcstr into str, but stops after maxlen bytes. returns str
- **strncpy(str, srcstr, maxlen)** — копирует srcstr в str, но останавливается после maxlen байт. Возвращает str.

- **strncat(str, srcstr, maxlen)** -- appends srcstr to str, but stops after maxlen bytes of srcstr have been read. returns str

- **strncat(str, srcstr, maxlen)** — добавляет srcstr к str, но останавливается после maxlen байт, прочитанных из srcstr. Возвращает str.

- **strcpy_from(str,srcstr, offset)** -- copies srcstr to str, starting offset bytes into srcstr. returns str.
- **strcpy_from(str,srcstr, offset)** — копирует srcstr в str, со смещением в offset байт в srcstr. Возвращает str.

- **strcpy_substr(str,srcstr, offset, maxlen)** -- copies srcstr to str, starting offset bytes into srcstr, and up to maxlen bytes. if offset is less than 0, offset is from end of source string. If maxlen is less than 0, length is limited to output string length shortened by maxlen. returns str.
- **strcpy_substr(str,srcstr, offset, maxlen)** — копирует srcstr в str, со смещением в offset байт из srcstr, и вплоть до maxlen байт. (Другими словами, копирует диапазон от offset до maxlen байт из srcstr ?) Если offset меньше 0, то смещение идет от конца строки-источника. Если maxlen больше 0, то длина ограничивается до длины выходной строки, укороченной до maxlen (*не совсем понял*). Возвращает str. **ПРОВЕРИТЬ!!!**

- **str_getchar(str, offset[, type])** -- returns the data at byte-offset offset of str. if offset is negative, position is relative to end of string. Type defaults to signed char, but can be specified to read raw binary data in other formats (note the single quotes, these are single/multi-byte characters):
  str_getchar(str, offset[, type]) — возвращает данные из str по байт-смещению offset. Если offset отрицательно, то позиция отсчитывается по отношению к концу строки. Type по умолчанию указывает на тип signed char (знаковое целое?), но может быть указано так, чтобы читать необработанные двоичные данные в других форматах (обратите внимание на одинарные кавычки, это одиночные/многобайтовые символы): **ПРОВЕРИТЬ!!!**
  можно если что так и оставить — оригинал, а в скобках перевод (только опять не уверен, что все перевел правильно - char вроде это символ, а int — целое?):
  - 'c' - signed char (знаковое целое)
  - 'cu' - unsigned char (беззнаковое целое)
  - 's' - signed short (знаковое короткое целое)
  - 'S' - signed short, big endian (знаковое короткое целое, обратный порядок байтов)
  - 'su' - unsigned short (беззнаковое короткое целое)
  - 'Su' - unsigned short, big endian (беззнаковое короткое целое, обратный порядок байтов)

    - 'i' - signed int (знаковое целое средней точности)
    - 'I' - signed int, big endian (знаковое целое средней точности, обратный порядок байтов)
    - 'iu' - unsigned int (беззнаковое целое средней точности)
    - 'Iu' - unsigned int, big endian (беззнаковое целое средней точности, обратный порядок байтов)
    - 'f' - float (число с плавающей точкой/запятой)
    - 'F' - float, big endian (число с плавающей точкой/запятой, обратный порядок байтов)
    - 'd' - double (число двойной точности)
    - 'D' - double, big endian (число двойной точности, обратный порядок байтов)

- **str_setchar(str, offset, value[, type])** -- sets the value at byte-offset "offset" of str to value (which may be one or more bytes of data). If offset is negative, then offset is relative to end of the string. If offset is the length of the string, or between (-0.5,0.0), then the character (or multibyte value if type is specified) will be appended to the string.
- **str_setchar(str, offset, value[, type])** — устанавливает значение по байт-смещению offset в строке str в value (которое(ая?) может быть одним или многими байтами данных). Если offset отрицательно, то позиция отсчитывается по отношению к концу строки. Если смещение (или offset?) это длина строки, или находится в диапазоне (-0.5,0.0), то символ (или многобайтовое значение, если такой тип был указан) будут добавлены к строке. **ПРОВЕРИТЬ!!!**

- **strcpy_fromslider(str, slider)** -- gets the filename if a file-slider, or the string if the slider specifies string translations, otherwise gets an empty string. slider can be either an index, or the sliderX variable directly. returns str.
- **strcpy_fromslider(str, slider)** — получает имя файла если это file-slider, или строку если slider задает (определяет?) string translations(?), иначе получает пустую строку. Slider может быть или индексом, или непосредственно значением переменной sliderX. Возвращает str. **ПРОВЕРИТЬ!!!**

- **sprintf(str,format, ...)** -- copies format to str, converting format strings:
  sprintf(str,format, ...) — копирует format в str, converting format strings:
    - %% = %
    - %s = string from parameter
    - %d = parameter as integer
    - %i = parameter as integer

- %u = parameter as unsigned integer
- %x = parameter as hex (lowercase) integer
- %X = parameter as hex (uppercase) integer
- %c = parameter as character
- %f = parameter as floating point
- %e = parameter as floating point (scientific notation, lowercase)
- %E = parameter as floating point (scientific notation, uppercase)
- %g = parameter as floating point (shortest representation, lowercase)
- %G = parameter as floating point (shortest representation, uppercase)

Many standard C printf() modifiers can be used, including:
Можно использовать многие стандартные модификаторы функции printf() языка Си, включая:

- %.10s = string, but only print up to 10 characters
- %-10s = string, left justified to 10 characters
- %10s = string, right justified to 10 characters
- %+f = floating point, always show sign
- %.4f = floating point, minimum of 4 digits after decimal point
- %10d = integer, minimum of 10 digits (space padded)
- %010f = integer, minimum of 10 digits (zero padded)

Values for format specifiers can be specified as additional parameters to sprintf, or within {} in the format specifier (such as %{varname}d, in that case a global variable is always used).
Значения для спецификаторов формата могут быть определены как дополнительные параметры для sprintf, или внутри { } в спецификаторе формата (например, %{varname}d, в этом случае всегда используется глобальная переменная).

- **match(needle, haystack, ...)** -- search for needle in haystack
  matchi(needle, haystack, ...) -- search for needle in haystack (case insensitive)
  For these you can use simplified regex-style wildcards:

match(needle, haystack, ...) — поиск needle в haystack
matchi(needle, haystack, ...) — поиск needle в haystack (без учета регистра)
Для них можно использовать упрощенные символы подстановки в стиле регулярных выражений:

- * = match 0 or more characters

- *? = match 0 or more characters, lazy
- + = match 1 or more characters
- +? = match 1 or more characters, lazy
- ? = match one character
- * = совпадают 0 или более символов
- *? = совпадают 0 или более символов, “ленивый” поиск
- + = совпадают 1 или более символов
- +? = совпадают 1 или более символов, “ленивый” поиск
- ? = совпадает один символ

Examples: Примеры:

match("*бла*", "эта строка содержит слово бла внутри себя") == 1
match("*бла", "эта строка оканчивается на слово бла") == 1

You can also use format specifiers to match certain types of data, and optionally put that into a variable:
Также вы можете использовать спецификаторы формата для поиска соответствия определенных типов данных, и при желании поместить их в переменную:

- %s means 1 or more chars
- %0s means 0 or more chars
- %5s means exactly 5 chars
- %5-s means 5 or more chars
- %-10s means 1-10 chars
- %3-5s means 3-5 chars.
- %0-5s means 0-5 chars.
- %x, %d, %u, and %f are available for use similarly
- %c can be used, but can't take any length modifiers
- Use uppercase (%S, %D, etc) for lazy matching
- %s означает 1 или более символов
- %0s означает 0 или более символов
- %5s означает точно 5 символов
- %5-s означает 5 или более символов
- %-10s означает 1-10 символов
- %3-5s означает 3-5 символов
- %0-5s означает 0-5 символов
- %x, %d, %u и %f так же доступны для использования
- %c может быть использован, но не может принимать модификаторы произвольной длины (?)

- Используйте верхний регистр (%S, %D и т.д.) для “ленивого” поиска совпадений.

The variables can be specified as additional parameters to match(), or directly within {} inside the format tag (in this case the variable will always be a global variable):
Переменные могут быть определены как дополнительные параметры для match(), или непосредственно в { } внутри тега формата (в этом случае переменная всегда будет глобальной переменной):

```
match("*%4d*","some four digit value is 8000, I say",blah)==1 && blah == 8000
match("*%4{blah}d*","some four digit value is 8000, I say")==1 && blah == 8000
```

Graphics Графика

Effects can specify a @gfx code section, from which the effect can draw its own custom UI and/or analysis display.
Эффекты могут содержать раздел кода @gfx, из которого будут отображаться собственные пользовательский интерфейс и/или дисплей анализатора.

These functions and variables must only be used from the @gfx section.
Эти функции и переменные должны быть использованы только в разделе @gfx.

Примечание переводчика:
Место пока зарезервировано.

- **gfx_lineto(x,y,aa)** *-- the aa parameter is optional in REAPER 4.59+*
  Draws a line from gfx_x,gfx_y to x,y. if aa is 0.5 or greater, then antialiasing is used. Updates gfx_x and gfx_y to x,y.
- **gfx_lineto(x,y,aa)** — *параметр aa является обязательным в REAPER 4.59+*
  Рисует линию от точки gfx_x,gfx_y до точки x,y. Если параметр aa равен 0.5 или более, то используется антиалиасинг. После этого обновляет значения gfx_x и gfx_y до x,y.

- **gfx_line(x,y,x2,y2[,aa])** — *REAPER 4.59+*
  Draws a line from x,y to x2,y2, and if aa is not specified or 0.5 or greater, it will be antialiased.
  Рисует линию от точки x,y до точки x2,y2, и если параметр aa не указан или равен 0.5 или более, то она будет сглажена.

- **gfx_rectto(x,y)**
  Fills a rectangle from gfx_x,gfx_y to x,y. Updates gfx_x,gfx_y to x,y.
  Рисует залитый прямоугольник от точки gfx_x,gfx_y по диагонали к x,y. После этого обновляет значения gfx_x и gfx_y до x,y.

- **gfx_rect(x,y,w,h)** — *REAPER 4.59+*
  Fills a rectngle at x,y,w,h pixels in dimension.
  Рисует залитый прямоугольник в точке x,y с размерами w,h пикселей.

- **gfx_setpixel(r,g,b)**
  Writes a pixel of r,g,b to gfx_x,gfx_y.
  Выводит (рисует?) пиксель с цветом r,g,b в точке gfx_x,gfx_y.

- **gfx_getpixel(r,g,b)**
  Gets the value of the pixel at gfx_x,gfx_y into r,g,b.
  Получает в r,g,b значение цвета (значения цветовых составляющих?) пикселя с координатами gfx_x,gfx_y.

- **gfx_drawnumber(n,ndigits)**
  Draws the number "n" with "ndigits" of precision to gfx_x, gfx_y, and updates gfx_x to the right side of the drawing. The text height is gfx_texth
  Выводит число n с точностью ndigits (количество цифр после запятой) в точку gfx_x, gfx_y, обновляет gfx_x до значения координаты правой стороны нарисованного (выведенного) числа. Высота текста задается в gfx_texth.

- **gfx_drawchar($'c')**
  Draws the character 'c' (can be a numeric ASCII code as well), to gfx_x, gfx_y, and moves gfx_x over by the size of the character.
  Рисует (Выводит) символ 'c' (также это может быть цифровой ASCII-код), в точку gfx_x,gfx_y, и смещает gfx_x на величину размера символа.

- **gfx_drawstr(str)** — *REAPER 4.59+*
  Draws a string at gfx_x, gfx_y, and updates gfx_x/gfx_y so that subsequent draws will occur in a similar place:
  gfx_drawstr("a"); gfx_drawstr("b");

  will look about the same as:
  gfx_drawstr("ab");

  Рисует строку в gfx_x, gfx_y и обновления gfx_x / gfx_y так, чтобы последующее обращение происходило в подобном месте:
  gfx_drawstr (""); gfx_drawstr ("б");

  будет выглядеть примерно так же, как:
  gfx_drawstr ("б");

- **gfx_measurestr(str,x,y)** — *REAPER 4.59+*
  Measures the drawing dimensions of a string with the current font (as set by gfx_setfont).
  Меры размеры рисунок строки с текущим шрифтом (как установлено gfx_setfont).

Измеряет размер строки **str**, рисуемой текущим шрифтом (установленным с помощью gfx_setfont)) и возвращает значения в **x y**.

- **gfx_setfont(idx[,fontface, sz, flags])** — *REAPER 4.59+*
  Can select a font and optionally configure it. idx=0 for default bitmapped font, no configuration is possible for this font. idx=1..16 for a configurable font, specify fontface such as "Arial", sz of 8-100, and optionally specify flags, which is a multibyte character, which can include 'i' for italics, 'u' for underline, or 'b' for bold. These flags may or may not be supported depending on the font and OS. After calling gfx_setfont, gfx_texth may be updated to reflect the new average line height.

  Может выбрать шрифт и, возможно, его настроить. idx = 0 для дефолтного растрового шрифта, настройка для этого шрифта не возможна. IDX = 1 .. 16 для конфигурации заданного шрифта, укажите FontFace такой как "Arial", размер 8-100, а при желании укажите идентификаторы, которые являются многобайтными символами, которые могут включать в себя "i" для курсивом, 'u' для подчеркивания , или 'b' для толстого шрифта. Эти идентификаторы могут или не могут быть поддержаны в зависимости от шрифта и ОС. После вызова gfx_setfont, gfx_texth могут быть обновлены, чтобы отразить новую среднюю высоту строки.

- **gfx_printf(str)** — *REAPER 4.59+*
  Formats and draws a string at gfx_x, gfx_y, and updates gfx_x/gfx_y accordingly (the latter only if the formatted string contains newline).
  Форматирует и рисует строку в gfx_x, gfx_y и обновления gfx_x / gfx_y соответственно (последний только если отформатированная строка содержит символ новой строки).

- **gfx_blurto(x,y)** — *REAPER 2.018+*
  Blurs the region of the screen between gfx_x,gfx_y and x,y, and updates gfx_x,gfx_y to x,y.
  “Размывает” прямоугольную область экрана между gfx_x, gfx_y и x,y и обновляет значения gfx_x, gfx_y до x,y.

- **gfx_blit(source, scale, rotation)** — *REAPER 2.018+*
  If three parameters are specified, copies the **entirity(?)** - м.б. entity? - of the source bitmap to gfx_x,gfx_y using current opacity and copy mode (set with gfx_a,

gfx_mode). You can specify scale (1.0 is unscaled) and rotation (0.0 is not rotated, angles are in radians).
For the "source" parameter specify -1 to use the main framebuffer as source, or 0..127 to use the image specified (or PNG file in a filename: line).
Если указаны три параметра, то копирует объект исходного растрового изображения в позицию gfx_x,gfx_y, используя текущие прозрачность и режим копирования (задаются в gfx_a и gfx_mode). Можно указать масштаб scale (1.0 без масштабирования) и поворот на угол rotation (0.0 без вращения, углы в радианах).
Для параметра “source” (источник) укажите -1 чтобы использовать главный кадровый буфер в качестве источника, или 0..127, чтобы использовать указанное изображение (или PNG-файл, заданный в filename: line).

- **gfx_blit(source, scale, rotation[, srcx, srcy, srcw, srch, destx, desty, destw, desth, rotxoffs, rotyoffs])** — *REAPER 4.59+*
  Srcx/srcy/srcw/srch specify the source rectangle (if omitted srcw/srch default to image size), destx/desty/destw/desth specify dest rectangle (if not specified, these will default to reasonable defaults -- destw/desth default to srcw/srch * scale).
  Srcx/srcy/srcw/srch задают прямоугольную область источника (если опущены srcw/srch, то по умолчанию берется размер изображения), destx/desty/destw/desth specify задают прямоугольную область приемника (если они не указаны, то по умолчанию будут установлены в приемлемые значения, т.е. — destw/desth будет установлено в srcw/srch * scale).

- **gfx_blitext(source, coordinatelist, rotation)** — *REAPER 2.018+*
  This is a version of gfx_blit which takes many of its parameters via a buffer rather than direct parameters.
  Это версия gfx_blit, котораяй принимает многие из ее параметров через буфер, а не напрямую.

  For the "source" parameter specify -1 to use the main framebuffer as source, or 0..127 to use the image specified (or PNG file in a filename: line).
  Для параметра “source” (источник) укажите -1 чтобы использовать главный кадровый буфер в качестве источника, или 0..127, чтобы использовать указанное изображение (или PNG-файл, заданный в filename: line).

  coordinatelist should be an index to memory where a list of 10 parameters are stored, such as:

Параметр (переменная?) coordinatelist должен быть указателем на область памяти, где хранится список из 10 параметров, таких как:

coordinatelist=1000; // use memory slots 1000-1009 используем слоты памяти 1000-1009 для
coordinatelist[0]=source_x;
coordinatelist[1]=source_y;
coordinatelist[2]=source_w;
coordinatelist[3]=source_h;
coordinatelist[4]=dest_x;
coordinatelist[5]=dest_y;
coordinatelist[6]=dest_w;
coordinatelist[7]=dest_h;
coordinatelist[8]=rotation_x_offset; // only used if rotation is set, represents offset from center of image используется только если задан rotation, представляет собой смещение по x от центра изображения
coordinatelist[9]=rotation_y_offset; // only used if rotation is set, represents offset from center of image используется только если задан rotation, представляет собой смещение по y от центра изображения

Вызов:
gfx_blitext(img,coordinatelist,angle);

- **gfx_getimgdim(image, w, h)** — *REAPER 2.018+*
  Retreives the dimensions of image (representing a filename: index number) into w and h. Sets these values to 0 if an image failed loading (or if the filename index is invalid).
  Получает в w и h размеры изображения image (представленного как filename: index number). Устанавливает эти значения в 0, если изображение не удалось загрузить (или индекс файла недействителен).

- **gfx_setimgdim(image, w,h)** — *REAPER 4.59+*
  Resize image referenced by index 0..127, width and height must be 0-2048. The contents of the image will be undefined after the resize.
  Изменяет размер изображения image, заданного индексом 0..127, ширина и высота должны быть в пределах 0-2048. После изменения размера содержимое image будет не определено.

- **gfx_loadimg(image, filename)** — *REAPER 4.59+*
  Load image from filename (see strings) into slot 0..127 specified by image. Returns the image index if success, otherwise -1 if failure. The image will be resized to the dimensions of the image file.
  Загружает изображение из файла filename (см. strings) в слот 0..127, указанный в image. В случае успешной загрузки возвращает индекс изображения, иначе, если ошибка, возвращает -1. Изображение image будет изменено до размеров изображения из файла.

- **gfx_gradrect(x,y,w,h, r,g,b,a[, drdx, dgdx, dbdx, dadx, drdy, dgdy, dbdy, dady])** — *REAPER 4.59+*
  Fills a gradient rectangle with the color and alpha specified. drdx-dadx reflect the adjustment (per-pixel) applied for each pixel moved to the right, drdy-dady are the adjustment applied for each pixel moved toward the bottom. Normally drdx=adjustamount/w, drdy=adjustamount/h, etc.
  Заполняет градиентный прямоугольник указанного цветом **r,g,b** и прозрачностью **a**. drdx-dadx отражают корректировку (попиксельную), применяемую для каждого пикселя, смещенного вправо, drdy-dady — для каждого пикселя, смещенного вниз. Обычно drdx=adjustamount/w, drdy=adjustamount/h и т.д., где adjustamount — величина(степень?) корректировки.

- **gfx_muladdrect(x,y,w,h, mul_r, mul_g, mul_b[, mul_a, add_r, add_g, add_b, add_a])** — *REAPER 4.59+*
  Multiplies each pixel by mul_* and adds add_*, and updates in-place. Useful for changing brightness/contrast, or other effects.
  Умножает каждый пиксель на mul_* и прибавляет add_*, тут же месте обновляет. Полезна для изменения яркости/контраста или других эффектов.

- **gfx_deltablit(srcimg,srcx,srcy,srcw,srch, destx, desty, destw, desth, dsdx, dtdx, dsdy, dtdy, dsdxdy, dtdxdy)** — *REAPER 4.59+*
  Blits from srcimg(srcx,srcy,srcw,srch) to destination (destx,desty,destw,desth). Source texture coordinates are s/t, dsdx represents the change in s coordinate for each x pixel, dtdy represents the change in t coordinate for each y pixel, etc. dsdxdy represents the change in dsdx for each line.

- **gfx_transformblit(srcimg, destx, desty, destw, desth, div_w, div_h, table)** — *REAPER 4.59+*

Blits to destination at (destx,desty), size (destw,desth). div_w and div_h should be 2..64, and table should point to a table of 2*div_w*div_h values (this table must not cross a 65536 item boundary). Each pair in the table represents a S,T coordinate in the source image, and the table is treated as a left-right, top-bottom list of texture coordinates, which will then be rendered to the destination.

- **gfx_r, gfx_g, gfx_b, gfx_a**
  These represent the current red, green, blue, and alpha components used by drawing operations (0.0..1.0).
  Эти специальные переменные представляют собой текущие значения цветовых составляющих (red, green, blue) и прозрачности (alpha), используемых в операциях рисования (0.0..1.0).

- **gfx_w, gfx_h**
  These are set to the current width and height of the UI framebuffer.
  Эти специальные переменные показывают текущие значение ширины и высоты графического пользовательского интерфейса.

- **gfx_x, gfx_y**
  These set the "current" graphics position in x,y. You can set these yourselves, and many of the drawing functions update them as well.
  Эти специальные переменные устанавливают(ся) в текущую графическую позицию x,y. Вы можете установить их самостоятельно, а многие графические функции обновляют их значения после своего выполнения.

- **gfx_mode**
  Set to 0 for default options. Add 1.0 for additive blend mode (if you wish to do subtractive, set gfx_a to negative and use gfx_mode as additive). Add 2.0 to disable source alpha for gfx_blit(). Add 4.0 to disable filtering for gfx_blit().
  Установите в 0 для параметров по умолчанию. Добавьте *(или все же Установите?)* 1.0 для режима аддитивного смешивания (если хотите сделать смешивание субтрактивным, установите gfx_a в отрицательное значение и используйте gfx_mode как аддитивный). Добавьте 2.0 чтобы запретить прозрачность источника (изображения?) для функции gfx_blit(). Добавтье 4.0 чтобы запретить фильтрацию для функции gfx_blit().

- **gfx_clear**
  If set to a value of >= 0.0, this will result in the framebuffer being cleared to that

color. the color for this one is packed RGB (0..255), i.e. red+green*256+blue*65536. The default is 0. Or you could just clear manually using lice_rectto.
Если установлено в значение >= 0.0, то результатом будет очистка буфера кадров до указанного цвета. Для задания цвета используется упакованный RGB (0..255), т.е. red+green*256+blue*65536. По умолчанию 0. Или вы можете просто очистить вручную с помощью lice_rectto.

- **gfx_dest** — *REAPER 4.59+*
  Defaults to -1, set to 0..127 to have drawing operations go to an offscreen buffer (or loaded image).
  По умолчанию -1, установите значение 0..127, чтобы операции рисования проходили за пределами кадрового буфера (или загруженного изображения).

- **gfx_texth**
  Set to the height of a line of text in the current font. Do not modify this variable.
  Устанавливает высоту строки текста в текущем шрифте. Не изменяйте эту переменную.

- **mouse_x, mouse_y, mouse_cap**
  mouse_x and mouse_y are set to the coordinates of the mouse. If the user is clicking and dragging the mouse within the UI/graphics area, mouse_cap is set to nonzero (1 if left mouse down, 2 if right - 4,8,16 will be set for control,shift, alt).
  mouse_x и mouse_y устанавливаются в координаты мыши. Если пользователь кликает и перемещает мышь в пределах пользовательского интерфейса/графической области, то mouse_cap устанавливается в ненулевое значение (1 если нажата левая клавиша мыши, 2 если правая - при использовании клавиш модификаторов, значение будет больше на 4,8 или 16 для Ctrl, Shift или Alt).

User defined functions and namespace psuedo-objects
Пользовательские функции и пространства имен псевдо-объекты
Правильнее, наверное, так:
Пользовательские функции и псевдо-объекты пространства имен

Note: the functionality available in this section requires REAPER 4.25+
Примечание: для функциональных возможностей, доступных в данном разделе требуется REAPER 4.25 +

JS now supports user defined functions, as well as some basic object style data access.
JS теперь поддерживает пользовательские функции, а также некоторый базовый доступ к данным в объектом стиле(?).

Functions can be defined anywhere in top level code (i.e. not within an existing () block, but before or after existing code), and in any section, although functions defined in @init can be used from other sections (whereas functions defined in other sections are local to those sections). Functions are not able to be called recursively -- this is enforced by functions only being able to call functions that are declared before the current function, and functions not being able to call themselves. Functions may have 0 to 40 parameters. To define a function, use the following syntax:
Функции могут быть определены в любом месте кода верхнего уровня (т.е. не в пределах существующего () блока, а до или после существующего кода), и в любом разделе, хотя функции, определенные в разделе @init могут быть использованы в других разделах (тогда как функции, определенные в других разделах являются локальными для этих разделов). Функции не могут быть вызваны рекурсивно — это соблюдено для того, чтобы они могли вызывать только те функции, которые объявлены до текущей функции. Функции не могут вызывать сами себя. Функции могут иметь от 0 до 40 параметров. Чтобы определить функцию, используйте следующий синтаксис:

```
function getSampleRate()
(
  srate; // возвращаем srate
);

function mySine(x)
(
  // аппроксимация по Тейлору
```

```
      x - (x^3)/(3*2) + (x^5)/(5*4*3*2) - (x^7)/(7*6*5*4*3*2) + (x^9)/(9*8*7*6*5*4*3*2);
    );

    function calculateSomething(x y)
    (
      x += mySine(y);
      x/y;
    );
```

Which would then be callable from other code, such as:
Которые затем будут вызываться из другого кода, например:

```
   y = mySine($pi*18000/getSampleRate());
    z = calculateSomething(1,2);
```

Note that the parameters for functions are private to the function, and will not affect global variables. If you need more private variables for a function, you can declare additional variables using a local() statement between the function declaration and the body of the function. Variables declared in the local() statement will be local to that function, and persist across calls of the function (though calls to a function from two different sections (such as @init and @sample) will have two different local states.

Обратите внимание, что параметры для функций являются закрытыми для функции, и не повлияют на глобальные переменные. Если для функции вам нужно больше частных переменных, можно объявить их с помощью выражения local() между объявлением функции и телом функции. Переменные, объявленные в выражении local(), будут локальными для этой функции, и сохраняются между вызовами функции (хотя вызовы функции из двух различных разделов (например, @init и @sample) будут иметь два различных локальных состояния.

Пример:

```
   function mySine(x) local(lastreq lastvalue)
    (
      lastreq != x ? (
        lastreq = x; // сохраняем последнее введенное значение
        // аппроксимация по Тейлору
        lastvalue = x - (x^3)/(3*2) + (x^5)/(5*4*3*2) - (x^7)/(7*6*5*4*3*2) +
(x^9)/(9*8*7*6*5*4*3*2);
      );
```

```
      lastvalue; // result of function is cached value Результатом функции является кэшированное  значение

    );
```

In the above example, mySine() will cache the last value used and not perform the calculation if the cached value is available. Note that the local variables are initialized to 0, which happens to work for this demonstration but if it was myCosine(), additional logic would be needed.
В приведенном выше примере, mySine() будет кэшировать последнее использованное значение и не выполнять вычисление если доступно значение в кэше. Следует помнить, что локальные переменные при инициализации устанавливаются в 0, что сработало в данном примере, но если бы это было myCosine(), то потребовалась бы дополнительная логика.

JS also supports relative namespaces on global variables, allowing for psuedo object style programming. Accessing the relative namespace is accomplished either by using a "this." prefix for variable/function names, or by using the instance() declaration in the function definition for variable names:
JS также поддерживает относительные пространства имен на (для, среди?) глобальных переменных, позволяя псевдо-объектный стиль программирования. Доступ к относительному пространству имен осуществляется либо с помощью префикса "this." для имен переменных/функций, либо с помощью объявления instance() в определении функции для имен переменных:

```
 function set_foo(x) instance(foo)
  (
    foo = x;
  );
  // or
  function set_foo(x)
  (
    this.foo = x;
  );

  whatever.set_foo(32); // whatever.foo = 32;
  set_foo(32); // set_foo.foo = 32;

  function test2()
```

```
  (
    this.set_foo(32);
  );
  whatever.test2(); // whatever.foo = 32
```

Additionally functions can use the "this.." prefix for navigating up the namespace hierarchy, such as:
Дополнительно функции могут использовать префикс "this.." для перемещения по иерархии пространства имен, например:

```
 function set_par_foo(x)
  (
    this..foo = x;
  );
  a.set_par_foo(1); // sets foo (global) to 1
  a.b.set_par_foo(1); // sets  a.foo to 1
```