Missing Title

2 min read • original

VirtualDub scripting language reference, v0.7
===

This is a document documenting VirtualDub's awful scripting interface. I am releasing it under the GNU General Public License (GPL), the same license as VirtualDub. This reference is current as of VirtualDub 1.6.7 (WIP), and may not apply to earlier versions. If I ever update this document again, check http://www.virtualdub.org/ for updates.

Note that I do not consider the scripting interface to be a major public interface, and thus have no qualms about breaking it at any time. I do try to keep compatibility to avoid breaking job scripts, but I have been known to goof (V1.4b). You have been warned.

-- Avery Lee <phaeron@virtualdub.org>
   June 09, 2005

VirtualDub's scripts

VirtualDub's batch system is based on the scripting language, and is stored in a file called VirtualDub.jobs in the program directory. If you click the Defer button in the save requester, information to start the processing job is queued in this file. It is text, so it can be edited by hand and produced by external programs. You will want to produce a couple of jobs to get a feel for how this works.

The command-line options, which are documented in the help file, allow you to launch VirtualDub, from a batch file or programmatically, to process a script. If you do this, you should heed the following:

1) Do not bundle VirtualDub.exe in an archive with the client application -- the GPL forbids distribution of the binary without source code. (This is not entirely true, since you may offer access to end users without requiring that they download source; read the GPL itself for details.) Furthermore, I don't like seeing VirtualDub stripped of its documentation and help file. Don't do it.

2) Make the distinction between your application and VirtualDub very clear, and let the user know you are a launching an external application and what to do if something goes wrong. I've only had one incident, but I do not want to get support email from people who think my program is at fault when an external application goofs. I get enough email already.

3) Save your script in VirtualDub.jobs, but save off the old file if it exists and restore it afterward. This protects a user's existing scripts and causes VirtualDub to post errors in the job file instead of presenting them to the user and halting the process.

4) If you use any filters which are not internal to VirtualDub, they *must* be installed in VirtualDub's plugins directory so that they are autoloaded on startup. VirtualDub does not currently allow external filters to be loaded from a script.

```
5) If you are launching VirtualDub from a batch file, you will need to use
   "start /wait" to launch it, or the batch file will continue execution
   immediately after VirtualDub loads.  For a program, launch VirtualDub using
   CreateProcess() and use WaitForSingleObject() to delay until VirtualDub
   completes.

VirtualDub's scripting system offers mainly the same controls that are available
through the user interface; there are no hidden features, and a few options
are not settable via scripting.  Capture mode is entirely unavailable from a
script, so don't even try to set up a Windows VCR this way.



Sylia: the world's worst scripting language
-------------------------------------------
I don't know what I was thinking when I created this scripting language.  It's
very loosely based on C, but it sucks more.  I must have been watching Bubblegum
Crisis before I created it.  There are only three types (void, int, string)
and three keywords defined:

    declare    var;    Declares a typeless global named var.
    true        1
    false        0

All statements are either declarations or expressions, and all statements
must end in a semicolon.  There is no flow control -- no functions, no
procedures, no if, no while, no for, no switch, no goto.  Sylia supports
class objects and arrays but they can't be defined or instantiated by scripts.
Objects can be assigned to variables, so the following is valid:

    declare foo;
    foo = VirtualDub.video.filters;
    foo.Add("bar");

Arrays and member function names can also be assigned to variables.  In the
latter case, overload resolution is deferred until the variable is
dereferenced.

Constants may be integers or strings.  Integers are 32-bit signed and can
be specified in decimal, octal (leading 0) or hex (leading 0x).  Type
suffixes are not allowed (i.e. -1L).  Strings must be double quoted, but
may contain the following C escapes:

    \a, \b, \f, \n, \r, \t, \v, \xhh, \\, \"

Strings may not contain nulls.

The following C operators are supported in expressions:

    =    assignment
    +    integer addition, string concatenation, or unary plus
    -    integer subtraction or unary minus
    *    integer multiplication
    /    integer division
    %    modulus
    []    array indexing operator
    ()    expression grouping or function dereference operator
    .    object dereference operator
    ~    bitwise not
    &    bitwise and
    |    bitwise or
    ^    bitwise xor
    !    logical not
    ==    integer equality
    !=    integer inequality
    <    integer less than
    <=    integer less than or equals
    >    integer greater than
    >=    integer greater than or equals
    &&    logical and
    ||    logical or
```

Precedence is the same as in C. Division and modulus by zero may cause
mild nausea, the destabilization of the universe, or a script error.

UTF-8 string encoding (VirtualDub 1.5.5+)

Beginning with VirtualDub 1.5.5, parts of the application run in Unicode under
Windows NT/2000/XP, meaning that filenames can have characters that are not
representable in 8-bit (ANSI). Script commands that take filenames, with the
exception of internal filters, now accept UTF-8 rather than ANSI. UTF-8 is
similar to UTF-16, the Unicode encoding used by Win32, except that code points
above U+007F are encoded using multi-byte sequences.

VirtualDub escapes UTF-8 sequences using C-style \x escapes when writing out
scripts, so high-bit characters are never seen in script files even when
present in filenames. This means that scripts in 1.5.5 continue to be readable
as ANSI files.

For more information on UTF-8 encoding or the Unicode standard, see the Unicode
website at http://unicode.org/.

Additional data types (VirtualDub 1.6.0+)

Starting with VirtualDub 1.6.0, Sylia supports 64-bit long integer (l) and
double-precision real (d) types, variables, and constants. The interpreter
will automatically promote or demote types as necessary to match a method
prototype; if multiple overloads are available, the first one in the list
is used.

Undecorated strings (VirtualDub 1.6.1+)

Because string paths with escaped backslashes are difficult to create from
batch files, and some languages may not have UTF-8 conversion support,
Sylia supports an alternative syntax for strings: strings prefixed with u
or U are treated as undecorated, ANSI encoded strings. An example:

```
    VirtualDub.Open(U"e:\test\test.avi");
```

The string is automatically converted from the system ANSI code page to
UTF-8 in the script system before being handed to the command system. This
allows file paths to be directly inserted in scripts without the need
for escaping or text conversion.

Script arguments (VirtualDub 1.6.4+)

An invoke (/i) switch has been added that allows parameters to be passed to
a script from the command line:

```
    virtualdub /i foo.script in.avi out.avi
```

Non-switch parameters immediately after the /i switch are placed into the
VirtualDub.params[] array. VirtualDub.params[0] would thus return the string
"in.avi" in the above example.

Command-line interface (VirtualDub 1.6.5+)

Starting with VirtualDub 1.6.5, it is possible to launch VirtualDub in
command-line mode, which makes some kinds of batch operations easier. To do
so, launch vdub.exe (32-bit) or vdub64.exe (64-bit). This will then redirect
the output of the program to standard output. Also, it will return a non-zero
error code when an error occurs.

Launching the program with /? will display command-line help.

Version query (VirtualDub 1.6.5+)

A /queryVersion flag has been added that causes VirtualDub to exit with the
build number of the executable. This makes it easier to detect and adjust for
different versions of the program. For an NT CMD batch file, the build number
will be found in the %ERRORLEVEL% environment variable.

Casting (VirtualDub 1.6.7+)

In 1.6.7, it is possible to use C-style casts to coerce between the numeric
types:

```
    Foo(1 + (int)4.0);
```

Casting from double to long or int causes truncation toward zero; if the
double value cannot be represented in the target type the result is undefined.
Casting from long to int results in the lower bits being kept, and casting
from int to long results in sign extension.

Numeric values cannot be casted to strings and vice versa. Use the Atoi(),
Atol(), Atod(), and ToString() functions for that.

Scripting function reference

All functions are member functions of objects. For instance, Open() is
a member function in the object VirtualDub, so to access it:

```
    VirtualDub.Open(...);
```

```
CAUTION: Most or all of the functions below do not do any parameter validation
         and will crash if you pass bad parameters.  Try not to do stupid
         things like add a -10x-10 resize filter!
```

Object: Sylia

```
void dprint(int/string value);
```

    Prints the named string or value to the debug output.

```
void messagebox(string text, string caption);
                    (VirtualDub 1.6.2+ or later)
```

    Displays a message box.

    NOTE: A message box is displayed even if VirtualDub is running in

```
        batch or command-line mode, so this is best not used in
        production scripts!

string ToString(int);            (VirtualDub 1.6.2+ or later)
string ToString(long);            (VirtualDub 1.6.2+ or later)
string ToString(double);        (VirtualDub 1.6.2+ or later)
string ToString(string);        (VirtualDub 1.6.2+ or later)

    Converts a numeric value to a string, formatted as a decimal number.

    The string-to-string version does nothing and exists for convenience
    when displaying expressions of arbitrary type for debugging.

int Atoi(string s);            (VirtualDub 1.6.5+ or later)
long Atol(string s);            (VirtualDub 1.6.5+ or later)
double Atod(string s);            (VirtualDub 1.6.5+ or later)

    Converts a string to an integer, long integer, or double. An error
    is thrown if the conversion fails because the string is not a valid
    number representation. These functions are useful for converting
    command-line parameter strings to numeric form.


Object: VirtualDub
------------------

void SetStatus(string text);

    Sets the text displayed on the status bar at the bottom of VirtualDub's
    window.

void Open(string filename, int type, int xopts);
void Open(string filename, int type, int xopts, string xoptstring);

    Opens a video file.  type is one of:

    0    Autodetect
    1    AVI
    2    MPEG-1
    4    Striped AVI
    5    AVI through AVIFile (Avisynth)

    If xopts is nonzero, VirtualDub opens the file with extended open
    options.  If xoptstring is absent, the dialog is opened, otherwise
    xoptstring is processed as a MIME BASE64 encoded string of the
    binary options structure.

void Open(string filename, string type, int xopts);
void Open(string filename, string type, int xopts, string xoptstring);
                    (VirtualDub 1.5.5+ or later)
    This is the preferred form of the Open() function in 1.5.5+.
    Instead of taking a number, it takes a type name instead.  Here
    are some type names:

    "Audio/video interleave input driver (internal)"
    "AVIFile/Avisynth input driver (internal)"
    "Image sequence input driver (internal)"
    "MPEG-1 input driver (internal)"

void Open(string filename)        (VirtualDub 1.6.5+ or later)

    This is a simpler form of the Open() command that always uses
    auto-detect mode for determining the file type.

void Append(string filename);
```

```
    Appends an additional video segment onto the current file.

void Close();

    Closes the current input file.

void Preview();

    Launches a preview of the current file, with the current settings.
    This is equivalent to File > Preview.

void SaveAVI(string filename);

    Runs the processing engine to produce an output file in AVI2 format.

void SaveCompatibleAVI(string filename);

    Runs the processing engine to produce an output file in AVI1 format.

void SaveSegmentedAVI(string filename, int spacethresh, int framethresh);

    Runs the processing engine to produce an output file in multiple AVI
    files.  spacethresh is the maximum file size in megabytes, framethresh
    is the maximum number of frames.  framethresh is ignored if it is
    zero.

void SaveImageSequence(string prefix, string suffix, int mindigits,
    int format);
void SaveImageSequence(string prefix, string suffix, int mindigits,
    int format, int quality);    (VirtualDub 1.6.0+ or later)

    Saves an image sequence with filenames in the form "prefix#suffix,"
    where # is the sequence number, padded to mindigits length with
    zeroes.  Format specifies the type of file to be generated:

        0 - Windows BMP
        1 - TARGA
        2 - JPEG        (VirtualDub 1.6.0+ or later)

void SaveWAV(string filename);        (VirtualDub 1.4d or later)

    Runs the processing engine to produce an output file in WAV format.

void RunNullVideoPass();        (VirtualDub 1.6.5+ or later)

    Runs the processing engine in video-only mode but discards the output.
    This is useful with video filters or codecs that have an analysis pass
    which does not produce usable output.

void Log(string output);        (VirtualDub 1.6.5+ or later)

    Outputs an entry to the log at Info priority. When VirtualDub is run
    from the command-line, this text will also be output to the standard
    output.


Object: VirtualDub.video
------------------------

int GetDepth(int var);
void SetDepth(int var, int value);

    If var is zero, these functions affect the input depth, otherwise
    they deal with the output depth.  GetDepth() returns 0, 1, or 2.
    SetDepth() receives 16, 24, and 32 as values for the same bit depths.
    I don't remember why they don't match.

    It is highly recommended that you use SetInputFormat() and
    SetOutputFormat() instead. The 16, 24, and 32-bit settings are
    equivalent to the XRGB1555, RGB888, and XRGB8888 formats,
    respectively.

void SetInputFormat(int format);    (VirtualDub 1.6.0+ or later)
void SetOutputFormat(int format);    (VirtualDub 1.6.0+ or later)

    Sets the preferred input and output formats used for video
    processing.

    0    Autodetect / Same as input
    5    XRGB1555    16-bit 555 RGB
    6    RGB565        16-bit 565 RGB
    7    RGB888        24-bit 888 RGB
    8    XRGB8888    32-bit 888 RGB + dummy alpha
    9    Y8        luminance only [16, 235]
    10    YUV422_UYVY    4:2:2 YCbCr interleaved, UYVY ordering
    11    YUV422_YUY2    4:2:2 YCbCr interleaved, YUY2 ordering
    14    YUV422_Planar    4:2:2 YCbCr planar (YV16)
    15    YUV420_Planar    4:2:0 YCbCr planar (YV12/I420)
    17    YUV410_Planar    4:1:0 YCbCr planar (YVU9)

int GetMode();
void SetMode(int mode);

    Sets the video processing mode:

        0    direct stream copy
        1    fast recompress
        2    slow recompress
        3    full processing mode

int GetFrameRate(int var);
void SetFrameRate(int var, int value);

    Gets or sets a particular frame rate control value:

    var=0    frame rate decimation factor, 1=all frames
    var=1    new frame rate in microseconds per frame; 0=no change, -1
        means match duration
    var=2    nonzero if inverse telecine is enabled

    Do not set frame rate decimation at the same time as inverse telecine.
    It won't work.

void SetTargetFrameRate(int hi, int lo);    (VirtualDub 1.5.2+)

    Sets the target frame rate for frame rate conversion as a 64-bit
    rational fraction (hi divided by lo).  Note that both hi and lo are
    *unsigned*.  Values from 2147483648 to 4294967295 must be passed as
    the equivalent 32-bit negative signed value!

    As of 1.6.7, it is OK to specify all values as positive integers,
    as the large values will be interpreted as long and then automatically
    converted to int to match the function.

int GetRange(int var);
void SetRange(int startMS, int endMS);

    Gets or sets a particular range value.  If var=0, the start offset
    is used, and if var=1, the end offset is used.  The start offset is
    measured in milliseconds from the beginning, and the end offset is
    in milliseconds from the end.  Yes, I know this is stupid.

    Because the end offset is only an offset, it is not possible to
    determine the length of the video stream using these functions.

void SetRangeEmpty();                (VirtualDub 1.6.5+)

    Clears the current selection but does not affect the start/end points
    of the processing range. This is equivalent to Clear Selection (Ctrl+D)
    in the UI and is useful for avoiding spurious displayed selections
    after the script runs.

int GetCompression(int var);

    Retrieves a variable for the currently selected compressor:

        var=0    Returns the fccHandler ID for the compressor.
        var=1    Returns the maximum keyframe interval.
        var=2    Returns the quality factor (0-10000).
        var=3    Returns the data rate (0=no data rate).

void SetCompression();
void SetCompression(string fccHandler, int keyrate, int quality, int datarate);
void SetCompression(int fccHandler, int keyrate, int quality, int datarate);

    The argumentless syntax turns off video compression.  The other two
    syntaxes select a video compressor.  Note that the fccHandler can be
    specified as either an integer or a string.  If the string is shorter
    than 4 bytes, it is padded with spaces.

void SetCompData(int length, string data);

    Sets the private codec data for the video compressor that is modified
    when you click Configure in the Video Compression dialog.  This data
    is opaque to VirtualDub and is used only by the video codec.  length
    is the length of the data block in bytes, and the data is the data
    block encoded in MIME BASE64.

void EnableIndeoQC(int enableQC);

    Enables or disables Quick Compress on the Ligos (formerly Intel) Indeo
    v4.x/5.x video codec.

void SetIVTC(int enableIVTC, int ivtcmode, int offset, int polarity);

    Sets inverse telecine parameters.

    enableIVTC:    nonzero if IVTC should be enabled
    ivtcmode:    0=field based, 1=frame based
    offset:        frame offset for the IVTC pattern, -1=adaptive
    polarity:    0=field A dominant, 1=field B dominant, ignored if
            offset=-1

    There is a bug in VirtualDub 1.4c that causes the offset value to
    be cast to a bool (either 0 or 1).  This unfortunately makes setting
    adaptive IVTC impossible.  The problem is fixed in V1.4d.

int width;        [VirtualDub 1.4d (12667) or later]
int height;

    Read-only variables giving the width and height of the source video.
    The values are undefined if the source video does not exist.
```

```
Object: VirtualDub.audio
------------------------

void GetMode();
void SetMode(int mode);

    Gets or sets the audio processing mode.

    0    direct stream copy
    1    full processing mode

int GetInterleave(int var);
void SetInterleave(int enabled, int preload, int interval, int is_ms,
    int offset);

    Gets or sets audio interleaving parameters.

    get    set        description
    var=0    enabled        Nonzero if audio interleaving is enabled.
    var=1    preload        Preload in milliseconds.
    var=2    interval    Interleaving interval in milliseconds or frames.
    var=3    is_ms        Nonzero if interleaving interval is in
            milliseconds.
    var=4    offset        Displacement offset in milliseconds.

int GetClipMode(int var);
void SetClipMode(int begin, int end);

    Gets or sets audio clipping parameters.  begin (var=0) specifies if
    audio should be displaced when video is left out.  This is almost
    always nonzero.  end (var=1) specifies if audio should be clipped if
    it is longer than the video.  This is also almost always nonzero.

int GetConversion(int var);
void SetConversion(int new_rate, int new_precision, int new_channels);
void SetConversion(int new_rate, int new_precision, int new_channels,
    int integral_rate, int high_quality);

    Gets or sets audio conversion parameters.

    var=0 (new_rate)    New sampling rate in Hz; 0=no change
    var=1 (new_precision)    New precision; 0=no change, 1=8-bit, 2=16-bit
    var=2 (new_channels)    0=no change, 1=mono, 2=stereo

void SetSource(int mode);
void SetSource(string file);

    Sets the audio source.  mode=0 is no audio, and mode=1 is input audio.
    The string form opens an external WAV file.

void SetCompression();
void SetCompression(int wFormatTag, int nChannels, int wBitsPerSample, int
    nAvgBytesPerSec, int nBlockAlign);
void SetCompression(int wFormatTag, int nChannels, int wBitsPerSample, int
    nAvgBytesPerSec, int nBlockAlign, int cbData, string data);

    Sets the audio compression format.  The first format clears audio
    compression.  The second and third specify parameters from a standard
    Win32 WAVEFORMATEX structure:

    wFormatTag    Specifies an audio compression format.
    nChannels    1=mono, 2=stereo.
    wBitsPerSample    8 or 16 for PCM, but may vary for others.
    nAvgBytesPerSec    Just what it says.
    nBlockAlign    The size of a compressed data block.
    These fields are present when using an audio compressor with
    private data, opaque to VirtualDub:

    cbData        Size of data in bytes
    data        MIME BASE64 encoded data string

    Usually, you will want to derive these fields by querying the compressor
    or looking at job scripts VirtualDub saves, because audio compressors
    tend to only accept specific values.  Some, but not all, of the private
    data formats are documented in MMREG.H in the Win32 Platform SDK.

void SetVolume();    [VirtualDub 1.4d (12667) or later]
void SetVolume(int v);    [VirtualDub 1.4d (12667) or later]
int GetVolume();    [VirtualDub 1.4d (12667) or later]

    Sets or gets the current audio volume amplification value.  The volume
    is expressed as an 8-bit fixed point fraction, where 256 is no amplifi-
    cation, 128 is half volume, and 512 is double volume.  The empty
    argument form of SetVolume() disables volume amplification.


Object: VirtualDub.subset
-------------------------
The subset is the edit list for processing the video stream. It consists of
a set of frame ranges, indicating which frames in the source stream are to
be processed. Frame numbers are zero-based.

In 1.5.4 and earlier, the subset is constrained to only contain ranges sorted
in source order, and thus ranges may be reordered when added. Starting with
1.5.5, the subset may contain duplicate or out-of-order ranges, allowing
portions of the video stream to be repeated or used in a different order than
in the original. For best compatibility it is recommended that ranges always
be added in ascending timeline order.


void Delete();
    Deletes the subset so that no edits are applied; the whole source
    file is processed, subject only to the start/end offsets.

void Clear();
    Removes any existing subset and creates a new one with no frames.
    This should be done before calling AddFrame()/AddRange() to start
    constructing the subset.


void AddFrame(int start, int length);
void AddRange(int start, int length);    [VirtualDub 1.4.10 or later]

    Adds a range of frames to the subset.  Subset frames should be
    isolated; they should not overlap or abut against each other (i.e.
    don't do [0,5] and [5,5]).  Starting with 1.5.5, overlapping or
    out-of-order ranges are allowed, but abutting ranges should still
    be avoided.

    The AddRange() function was added in 1.4.10 as a better-named synonym.
    It is equivalent to AddFrame(), and the AddFrame() syntax is now
    deprecated.

void AddMaskedRange(int start, int length); [VirtualDub 1.4.8 or later]

    Adds a range of masked frames to the subset. A masked frame is a
    frame that repeats the previous frame, wherever it may come from;
    it is used as a placeholder for a frame that is otherwise unusable,
    such as one with corrupted data.
```

```
Object: VirtualDub.params        (VirtualDub 1.6.1+ or later)
--------------------------------------------------------------------

string operator[](int index);        (VirtualDub 1.6.1+ or later)

    Retrieves parameters from an invoke (/i) command-line switch.
    The parameter indices are zero-based, so params[0] is the first
    parameter. If the indicated parameter does not exist, an error
    is thrown.


Object: VirtualDub.project        (VirtualDub 1.6.5+ or later)
--------------------------------------------------------------------

void ClearTextInfo();            (VirtualDub 1.6.5+ or later)

    Clears the text information assocated with output AVI files. By
    default annotations are loaded and preserved from the input file.

void AddTextInfo(string fourcc, string text);

    Adds a text annotation for use with subsequent AVI output files, such
    as author and name strings. fourcc should be one of the standard AVI
    text annotation chunk four character codes (FOURCCs), such as "ISBJ"
    and "IART." Note that the second parameter, which specifies the text,
    is a raw 8-bit string that is directly copied into the AVI file, and
    is _not_ UTF-8 -- this is because AVI itself doesn't have a viable
    way to indicate the code page encoding of such strings.


Object: VirtualDub.video.filters
--------------------------------

void Clear();

    Removes all filter instances from the filter list.

void Add(string filter);        [VirtualDub 1.4c or older]
int Add(string filter);            [VirtualDub 1.4d or newer]

    Adds a new instance of the specified filter to the bottom of the
    video filter list.

    Starting with VirtualDub 1.4d, this function returns the instance
    number of the added filter.


Object: VirtualDub.video.filters.instance[nFilt]
------------------------------------------------

nFilt is a zero-based index of the filter instance to be modified.

void Remove();

    Removes the selected filter from the list.  Note that this changes
    the indices of the filters after it.

int GetClipping(int var);
void SetClipping(int x1, int y1, int x2, int y2);

    Gets or sets filter input clipping parameters.  var ranges from 0 to 3
    and corresponds to x1, y1, x2, and y2 respectively.  All values are
    in pixels from the edge.
```

```
... Config(...);

    This is a function that may be provided by the video filter.  It is
    required for a filter to work in batch mode if the filter has
    user-definable parameters.  However, no syntax is enforced on this
    member and the parameter syntax will vary from filter to filter.



Filter configuration functions
------------------------------
2:1 reduce        No configuration.
2:1 reduce (HQ)        No configuration.
3x3 average        No configuration.
blur            No configuration.
blur more        No configuration.

box blur        void Config(int width, int power);

brightness/contrast    void Config(int brightness, int contrast);

deinterlace        void Config(int mode);
                0=blend, 1=dup1, 2=dup2, 3=discard1,
                4=discard2, 5=unfold, 6=fold

emboss            void Config(int direction, int height);

field swap        No configuration.

fill            void Config(int x1, int y1, int x2, int y2, int color);
                x1...y2: insets in pixels, color is 24-bit RGB
                (like HTML but with 0x instead of #)

flip horizontally    No configuration.
flip vertically        No configuration.

general convolution    void Config(int c0, int c1, int c2, int c3, int c4,
                int c5, int c6, int c7, int c8, int bias,
                int clip);

grayscale        No configuration.

hsv            void Config(int h, int s, int v);

            h: Hue adjustment, 0...65535 -> 0...360 degrees
            s: Saturation multiplier, 0...131072 -> 0-200%
            v: Value multiplier, 0...131072 -> 0-200%

invert            No configuration.

levels            void Config(int inputlo, int inputhi, int gammacorr,
                int inputmid, int outputlo, int outputhi);
            void Config(int inputlo, int inputhi, int gammacorr,
                int inputmid, int outputlo, int outputhi,
                int lumaonly);

                All are 0-255 except gammacorr which is a
                24-bit fixed point fraction.  If lumaonly is
                nonzero the filter works in luma instead of
                RGB.

logo            void Config(string logoFile, int xpos, int ypos,
                int alphaEnable, int premultDisable, int xj,
                int yj, int opacity);
            void Config(string logoFile, int xpos, int ypos,
                string alphaFile, int premultDisable, int xj,
                int yj, int opacity);

            logoFile:     filename of logo image
            xpos, ypos:     position in pixels from placement
                    origin (see xj/yj below)
            alphaEnable:    non-zero to enable per-pixel alpha
                    blending based on alpha channel
            premultDisable:    non-zero to use non-premultiplied
                    alpha
            xj:        horiz. justification
                0 - left
                1 - center
                2 - right
            yj:        vert. justification
                0 - top
                1 - middle
                2 - bottom
            opacity:    constant alpha to apply over entire
                    logo, 0...65536


null transform        No configuration.

resize            void Config(int w, int h, int/string mode);
            void Config(int w, int h, int/string mode, int framew,
                int frameh, int color);

                filter modes:
                    0     "nearest" or "point"
                    1    "bilinear"
                    2    "bicubic"
                    3    (precise bilinear, no string)
                    4    (precise bicubic, no string)


rotate            void Config(int mode);
                0=left90, 1=right90, 2=180


rotate2            void Config(int angle, int filtmode, int color, int
                expandbounds);

                angle: 24-bit fraction (16777216 = 360d)
                filtmode: 0=point, 1=bilinear, 2=bicubic

sharpen            void Config(int power);
smoother        void Config(int threshold, int use_blur_pass);
threshold        void Config(int threshold);

temporal softener    No configuration.
(motion blur)

temporal smoother    void Config(int power);

TV            void Config(mode);
                0=Y, 1=I, 2=Q, 3=avg3x3, 4=avg5x5, 5=5x5+tmp,
                6=chromaup, 7=chromadown

other filters
-------------
The version of Donald Graft's smart deinterlacer bundled with VirtualDub:

    void Config(int motion_only, int blend, int luma_threshold, int
        scene_threshold);

My subtitler:

    void Config(int enable_supersampling, string script_filename);

Others, you'll have to ask the filter author.



Object: VirtualDub.audio.filters                   (VirtualDub 1.5+)
---------------------------------------------------------------------
This subobject controls the filter graph for advanced audio mode.


void Clear();

    Removes all filters from the advanced filter graph.

int Add(string name);

    Adds an audio filter to the graph.  The name of the filter is UTF-8
    encoded.  Returns the ID# of the filter.

void Connect(int srcfilt, int srcpin, int dstfilt, int dstpin);

    Establishes a connection between two filters, from an output pin on a
    source filter (srcfilt/srcpin) to an input pin on a destination filter
    (dstfilt/dstpin).  Both filter and pin numbers are zero-based.


Object: VirtualDub.audio.filters.instance[nFilt]   (VirtualDub 1.5+)
---------------------------------------------------------------------
nFilt is the zero-based index of the audio filter in the filter graph.

void SetInt(int parmidx, int value);
void SetLong(int parmidx, int valuehi, int valuelo);
void SetDouble(int parmidx, int valuehi, int valuelo);
void SetString(int parmidx, string value);
void SetRaw(int parmidx, int length, string base64value);

    Sets configuration parameters for an audio filter.

    "parmidx" is the index of the configuration parameter.

    For long (64-bit int) and double (64-bit FP) parameters, the value
    is broken into two 32-bit integers, with the high 32-bits being passed
    first.

    For string parameters, the string value is a Unicode string encoded
    as UTF-8.

    For raw (binary) parameters, "length" refers to the raw unencoded
    length in bytes, and "base64value" is the binary data encoded using
    MIME BASE64 encoding.


void SetLong(int parmidx, long value);        [VirtualDub 1.6+]
void SetDouble(int parmidx, double value);    [VirtualDub 1.6+]

    These overloads are equivalent to the raw methods of the same name
    in 1.5, except they take long and double values directly.


=======================
```

Audio filter parameters

These are the configuration parameters for the audio filters built into VirtualDub 1.5.6.

Filter	Index/type	Description
Gain	0 (double)	Gain factor (-8.0 to 8.0)
New rate	0 (uint32)	New frequency in Hz
Ratty pitch shift	0 (double)	Pitch shift ratio (0.5 to 2.0)
Lowpass/highpass	0 (uint32)	Cutoff frequency in Hz
	1 (uint32)	Filter taps (approx quality)
Resample	0 (uint32)	New frequency in Hz
	1 (uint32)	Filter taps (approx quality)
Stretch	0 (double)	Stretch ratio (0.25 to 4.0)

VirtualDub.jobs file format

Jobs in VirtualDub are stored as scripts in plain text format, with additional control parameters stored as specially-formatted text strings. It is best to let VirtualDub modify its job control file instead of modifying it directly, but it may be useful to do this with a text editor or with an external program.

All control lines are comment lines with a token starting with a dollar sign ($), and optionally followed by arguments, with only one control op per line:

```
// $numjobs 3
```

This line tells VirtualDub that there are three jobs in this file. Since all non-scripting commands are stored as comments, it is possible to execute the job file as an ordinary script, although error control is different in job control -- individual sections are executed as separate scripts and errors are isolated between them, so that an error in one script does not prevent others from executing.

Each additional job in the file is of the form:

```
    // $job "Job 1"
    // $input "f:\mkrtest.avi"
    // $output "f:\test.avi"
    // $state 2
    // $start_time 01c01df3 c2eb68c0
    // $end_time 01c01df3 eb3d85b0
    // $script

    <script commands>

    // $endjob
```

Notice that the job ($job), input filename ($input), and output filename ($output) arguments are strings but do not contain escapes. Also, the input and output filenames are used for display purposes; they should match the script but changing them will not change the files used.

```
$state controls the execution state of the job entry:

    0    WAITING        Job is ready to be executed.
    1    INPROGRESS    Job is currently being executed.  If VirtualDub sees
            this tag when loading a jobs file, it assumes that
            the operation crashed, and the job is switched to
            the ABORTED state.
    2    DONE        Job is completed and does not need to be executed.
    3    POSTPONED    Job is ready to be executed but has been postponed
            by the user and thus should be skipped.
    4    ABORTED        Job was started but did not complete properly.
            VirtualDub won't reattempt this job unless it is
            switched to WAITING by the user.
    5    ERR        An error was enountered while executing this job.
            VirtualDub won't reattempt this job unless it is
            switched to WAITING by the user.

$start_time and $stop_time contain the starting and stopping times,
respectively, of the given job.  The times are stored as two 32-bit hexa-
decimal values, with the first being the dwHighDateTime value of a Win32
FILETIME structure, and the second being dwLowDateTime.  Zero for both
values indicates no time for that entry.  It makes no sense to have
a stop time without a start time.

The actual Sylia script is bracketed by the $script and $endjob markers;
this means that the script must come last, after all job parameter
commands.  Any command is actually valid in this script, including
multiple operations; this may be helpful if multiple operations need to
be sequenced, and subsequent operations can't be done if the initial
ones fail.  Scripts run with whatever environment exists at the time of
invocation, so the script needs to set all parameters appropriately, and
can't assume configuration variables will be set in any particular manner.
```

Original URL:
http://virtualdub.com/docs/vdscript.txt