Watch the Half-Life TASing introduction here https://youtu.be/viSzM2d4Ff0 and join #goldsrc-tas in https://discord.gg/sourceruns.

The way you make a TAS with the new BXT is by making a *script*. But not the usual Half-Life config file (like you used to do in HL TAS Mod). The BXT TAS scripts are called *hltas-scripts* and usually have an extension `.hltas`. They can be opened with any text editor and there is a graphical application planned for creating and editing them.

The way you open a script file is by using the `bxt_tas_loadscript` console command. It has one parameter - the filename. Absolute paths are accepted and relative paths usually base in the Half-Life root directory - the one with *hl.exe* and mod folders inside. Example: `bxt_tas_loadscript test.hltas`. The script starts executing on the first "good" frame, so you can execute things like `map c1a0;bxt_tas_loadscript cool-tas.hltas`.

~~To prevent occasional desyncs on post-FPS bugfix engines, you need to start the TAS like this: `_bxt_reset_frametime_remainder;map c1a0;bxt_tas_loadscript cool-tas.hltas`.~~ This is no longer needed after the Jan 4 2020 version of Bunnymod XT.

HLTAS script layout

The hltas scripts have two main sections - the properties section where you can set up various "global" parameters like the name of a demo that will be recorded for this script, and the frame data section which contains the actual input. Every script begins with the properties section and the two sections are separated by a line saying `frames`. Thus, the hltas scripts have the following layout:

```
<properties>
frames
<input>
```

Before the layout is considered in detail, it's worth noting that you can insert *comments* in the hltas scripts. Comments start with `//` and last till the end of the current line. As soon as `//` is found on a line, the rest of the line is simply ignored.

You can also insert any number of whitespace characters in the beginning and in the end of every line (but not in the middle of a framebulk line, which will be explained later), as well as any number of blank lines anywhere you want, those are ignored as well.

The properties section

Every property starts on its own line and has a format of `name value`, where the name and the value are delimited by any number of whitespace characters. The name cannot contain any whitespace characters.

There are a couple of properties you can set:

- the first property that every hltas script should have - `version`. This is to indicate the revision of the hltas script format. At the moment there is only one version - the first one, so every hltas script should start with a line saying `version 1`.
- `hlstrafe_version` - used for making sync-breaking changes without making old TASes desync. For new scripts you want to set this to the highest supported version, which Bunnymod XT will tell you on `bxt_tas_loadscript`, and which it will insert automatically on `bxt_tas_new`.
- `demo` - if set to a filename, starts recording a demo with this filename as soon as possible after the script started executing and stops recording after the script has finished. Executes `record <value>` as soon as possible and `stop` after the script has finished.
- `save` - if set to a filename, saves the game with that save name on the first possible frame after the script has finished executing. Executes `save <value>` as soon as possible after the script has finished.
- `seed` - the value is in form of two whitespace-separated numbers, the first one indicates the shared RNG seed and the second one - the non-shared RNG seed.
- `frametime0ms` - sets the frametime to use for 0ms frames.

The frames section

The frames section can contain the *framebulks* (which are basically input descriptions for a number of frames) and special frames which are either properties (which you might want to change multiple times in a script, like what buttons the strafing should use) or saveload statements.

Framebulks

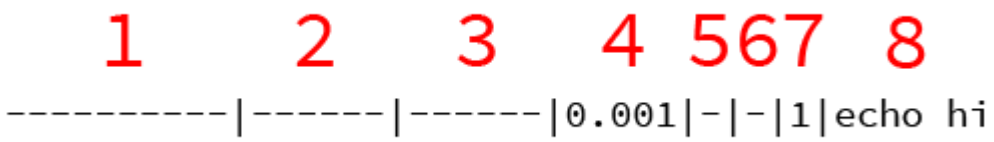
Let's start with the framebulks. These are lines formatted in a certain way. A framebulk consists of several fields separated with the `|` symbol (so-called *separators*). This is an example of a framebulk:

```
----------|------|------|0.001|-|-|1|echo hi
```

In this example all 8 possible fields are utilized. Before we look at each field in detail, note that if you don't need to set some field and every field that comes after it, you may leave them out (including the separators). The most common example of this is leaving out the last field where you can insert custom console commands like this:

```
----------|------|------|0.001|-|-|1
```

Let's consider each field in detail.



Fields 1 through 3 define the actual input. 1 is the field for autofuncs, 2 and 3 for manual input. Field 4 sets the frametime, fields 5 and 6 - the player viewangles, yaw and pitch, respectively. Field 7 sets the number of frames in this framebulk and field 8 contains any custom console commands you might want to use.

You see a lot of dashes in fields 1-3. To toggle some function or input on, you replace the dash with a certain letter. This is done so the framebulks look nice and aligned.

Field 1: the autofuncs. In the fully filled-out form the field looks like this:

```
sXXljdbcgw
```

The first three symbols here control the **s**trafing. They must be either all filled, or all empty (dashes). After the letter follow two digits, the first one sets the strafing type and the second one - the strafing direction.

Strafing types:

- 0 - maximum acceleration. This is what you use most often.
- 1 - maximum angle. This is sometimes used for turns.
- 2 - maximum deceleration. You use this when you need to slow down.
- 3 - constant speed. Use this if you want to turn, maximum acceleration strafing turns too slowly, and maximum angle strafing drains too much speed.

Strafing directions:

- 0 - left. Strafes to the left of current velocity direction.
- 1 - right. Same as above, but right.
- 2 - best direction. Strafes to whatever the best direction is for the current strafing type. Only useful for max deceleration strafing or when you're making speed tests as it usually just turns around randomly.
- 3 - yaw. Strafes to the given yaw (specified in the yaw field), This is what you use the most often.
- 4 - point. Strafes to the point with the given coordinates (X and Y, space-separated, in the yaw field).

The most common thing you're gonna do is max acceleration strafe to the given yaw, which is written as `s03`.

`l` - **l**gagst. This stands for *Leave Ground at Air-Ground Speed Threshold*. What this means is jump or ducktap when it's faster to move in the air than on the ground. If you have the lgagst enabled, you will need either autojump or ducktap turned on as well to specify what you want to do when the threshold is reached.

If you specify the capital `L` instead, the lgagst check will use the uncut maxspeed for its tests regardless of if the player is ducked or not. This is useful for spots where you land while ducked but want to stand up and continue groundstrafing while standing.

`j` - auto**j**ump. When `l` is disabled, simply jumps whenever possible. When `l` is enabled, indicates that the player should jump upon reaching the optimal speed.

`d` - **d**ucktap. When `l` is disabled, ducktaps whenever possible. When `l` is enabled, indicates that the player should ducktap upon reaching the optimal speed.

If you specify the capital `D` instead, BXT will use the `frametime0ms` frametime for ducktap frames (for high fps ground friction-less ducktapping).

`b` - jump**b**ug. When enabled, jumpbugs whenever possible.

`c` - duck before **c**ollision. When enabled, ducks if the player is moving in the air, the next frame the player would collide with something and wouldn't collide if they ducked. The check excludes the ground and ceilings.

If you specify the capital `C` instead, ceilings are included in the check.

`g` - duck before **g**round. When enabled, ducks if the player is in the air and the next frame they would hit the ground if not ducked.

`w` - duck **w**hen jump. Ducks on any jump. Useful when the player has the longjump module and you want to use it.

Field 2 and 3: the manual input. In the fully filled-out form the field looks like this:

```
flrbud|jdu12r
```

Everything is simple here - **f**orward, **l**eft, **r**ight, **b**ack, **u**p, **d**own; **j**ump, **d**uck, **u**se, attack**1**, attack**2**, **r**eload.

Field 4: frametime. Frametime is 1 / FPS. For example, for 1000 FPS use 0.001. But be careful with this, for example for 100 FPS you need to use a little more than 0.01, like 0.010000001. This is due to how floating point numbers work. Basically you want to look at your cl_showfps counter and have it at 99 for 100 FPS, 999 for 1000 FPS, 249 for 250 FPS, etc.

Field 5: yaw. When you're not strafing just sets the yaw viewangle to this, when yawstrafing - the target yaw, when pointstrafing - space-separated X and Y coordinates.

Field 6: pitch. Just sets the pitch viewangle to this.

Field 7: number of frames. Number of frames this framebulk lasts. The example framebulk lasts 1 frame.

Field 8: custom console commands. If you want to insert any custom console commands in this frame, put them here. Note that if the framebulk lasts more than 1 frame, the commands are inserted every frame.

Thus,

```
s00-------|------|------|0.001|12.5|-|5|echo hi
```

equals

```
s00-------|------|------|0.001|12.5|-|1|echo hi
s00-------|------|------|0.001|12.5|-|1|echo hi
s00-------|------|------|0.001|12.5|-|1|echo hi
s00-------|------|------|0.001|12.5|-|1|echo hi
s00-------|------|------|0.001|12.5|-|1|echo hi
```

TODO: autofuncs times.

Special frames

The special frames include properties (in the same `name value` format as the ones from the properties section):

- `buttons` - set this to 4 whitespace-separated digits to set the buttons the strafing will use. The first digit is air-left button, second - air-right, third - ground-left and the fourth - ground-right. The buttons are 0 through 7, starting from W, counter-clockwise, so: 0 = W; 1 = WA; 2 = A; 3 = SA; 4 = S; 5 = SD; 6 = D; 7 = WD. For example, to strafe with WA/WD on the ground and D/A in the air (aka A and D but backwards), you write `buttons 6 2 1 7`. To reset the buttons to the default setting (when the strafing decides the best buttons to use), simply write `buttons` without any parameters.

- `lgagstminspeed` - the parameter is a number, if your horizontal speed is below this number, lgagst will not trigger. The default value for this is 30.

- `save` - the parameter is a filename, when this is encountered, a saveload is performed. `save <filename>;load <filename>` is executed and the script continues the execution as usual. Useful for fastfire.

- `seed` - the parameter is the shared RNG seed to use for the next level load (a saveload frame or a changelevel).

- `reset` - the parameter is the non-shared RNG seed. Saves, restarts the game setting the non-shared RNG seed, loads and continues executing the script normally.

- `strafing yaw` or `strafing vectorial` - switches to regular or vectorial strafing.

- `target_yaw` - sets the angle which the vectorial strafing will make the player look at. Possible values are:
  - `target_yaw velocity_avg +-0.1` - makes the player look towards where he's moving +- 0.1 degrees. You can change this tolerance, bigger values lead to faster script execution (as in in real-time on your PC, not in-game) but more camera twitching. This parameter averages two last velocity values to further smooth the camera movement.
  - `target_yaw velocity +-0.1` - just like the previous one but a little more twitchy.
  - `target_yaw 90 +-0.1` - set the angle directly. This will make the player look at yaw 90 regardless of where he's moving.
  - `target_yaw from 30 to 35` - will make the player look anywhere between those angles. The order matters: `from 10 to 350` covers almost the entire angle range, while `from 350 to 10` is a very narrow range, opposite of the previous one.

- `change` - smoothly changes the player angle over time:
    - `change yaw to 180 over 1 s` - smoothly changes the player yaw to 180 over 1 second.
    - `change pitch to -20 over 0.3 s` - smoothly changes the player pitch to -20 over 0.3 seconds (300 milliseconds).
    - `change target_yaw to 270 over 4 s` - smoothly changes the vectorial strafing target yaw to 270 over 4 seconds.

Script examples

```
version 1
demo example
frames
s03l-d----|------|------|0.010000001|179|-|84
s03l-d----|------|------|0.010000001|90|-|32
s03l-d----|------|------|0.010000001|0|-|55
s03l-d----|------|------|0.010000001|270|-|20
```

This defines a script that does some strafing and ducktapping at 100 FPS. We start by max accel yawstrafing with lgagst-ducktapping to the yaw = 179 for 84 frames, then yawstrafe to the yaw = 90 for 32 frames, then to 0 for 55 frames and to 270 for 20 frames. This will also record a demo called example.

```
version 1
demo test
frames
lgagstminspeed 50
s03lj-----|------|------|0.001|81.8|-|51

buttons 3 3 0 0
s03lj-----|------|--u---|0.001|81.8|-|1
buttons

s03l-d-c--|------|------|0.010000001|88.6|-|89
```

Here we change the lgagst min speed to 50, then strafe and lgagst-jump to yaw = 81.8 for 51 frames at 1000 FPS, then change the buttons to SA/SA W/W and strafe for one more frame, additionally pressing the use key. In the end, we reset the buttons and strafe with lgagst-ducktap and duck before collision to yaw = 88.6 at 100 FPS for 89 frames.

Reference

`sXXljdbcgw|flrbud|jdu12r|<frametime>|<yaw>|<pitch>|<number of frames>|custom console commands`

strafe, type, dir
lgagst, auto**j**ump, **d**ucktap, jump**b**ug, duck before **c**ollision, duck before **g**round, duck **w**hen jump
forward, **l**eft, **r**ight, **b**ack, **u**p, **d**own
jump, **d**uck, **u**se, attack**1**, attack**2**, **r**eload

host_framerate - frametime - 1 / FPS
1000 FPS: 0.001
100 FPS: 0.010000001

yaw or space-separated point coordinates for point strafing

Attempt to make something simpler

Header:

```
version 1
demo mydemo
frames
```

Frames:

The line you'll need 90% of times (changing the yaw and the frame count of course):

```
s03lj-----|------|------|0.001|88.6|-|89
```

This means go fast into some yaw for some number frames at **1000 fps** and also jump when it's best to do so. In this case the yaw is **88.6** and the frame count is **89**.

Ducktap version:

```
s03l-d----|------|------|0.001|88.6|-|89
```

If you want 100 FPS:

```
s03lj-----|------|------|0.010000001|88.6|-|89
```

If you want to also duck before **c**ollision (if you're gonna bump into something mid-air but you won't if you duck, it ducks):

```
s03lj--c--|------|------|0.001|88.6|-|89
```

Or maybe to attempt jump**b**ug on every landing:

```
s03lj-b---|------|------|0.010000001|88.6|-|89
```

Or duck right before each landing (before **g**round):

```
s03lj---g-|------|------|0.001|88.6|-|89
```

More restrictive version (for TASVideos)

This one shouldn't contain any autofuncs. It should be possible to determine the length of a TAS just from the input file.

Important point: this file is NOT for writing by hand, it will be generated automatically.

Important point: this file should NOT be in any way abusable, so any valid input file should represent a valid input you can make in the game.

Actual specification (may change slightly):

The format is a JSON file. At the top level there is an object, which contains:

- "version" - number, format version.
- "author" - string, author's name.
- "rerecords" - number, >= 0, rerecord count.
- "bxt_version" - string, BXT version.
- "os" - string, can be either "windows" or "linux".
- "hash" - object.
- "movevars" - object.
- "skill" - number, value of the skill console variable (difficulty).
- "seed" - object.
- "map" - string, mapname.
- "pause" - bool, whether bxt_autopause 1 should be used.
- "frames" - array.

The "hash" object contains:

- "hw" - string, SHA1 of hw.dll or hw.so depending on "os". Inb4 someone uses D3D or Software.
- "client" - string, SHA1 of the client library.
- "server" - string, SHA1 of the server library.

The "seed" object contains:

- "shared" - number, 32-bit int, shared RNG seed.
- "nonshared" - number, 32-bit int, non-shared RNG seed.

The "movevars" object contains:

- "gravity" - number, value of sv_gravity.
- "stopspeed" - number, value of sv_stopspeed.
- "maxspeed" - number, value of sv_maxspeed.
- "accelerate" - number, value of sv_accelerate.
- "airaccelerate" - number, value of sv_airaccelerate.
- "wateraccelerate" - number, value of sv_wateraccelerate. (This is unused in HL1, may be used somewhere else)
- "friction" - number, value of sv_friction.
- "edgefriction" - number, value of edgefriction.
- "waterfriction" - number, value of sv_waterfriction. (This is unused in HL1, may be used somewhere else)
- "bounce" - number, value of sv_bounce.

- “stepsize” - number, value of sv_stepsize.
- “maxvelocity” - number, value of sv_maxvelocity.

The “frames” array contains objects which contain:
- “t” - number, frame type. One of the following: 0 - input, 1 - saveload, 2 - reset, 3 - parameters.
- “d” - object, optional, frame data. What it contains depends on “t”. If “t” = 1 (saveload) this object can be omitted.

The “d” object contains:
If type is 0 (input):
- “f” - number, <= 0.25, > 0. Frametime.
- “b” - number, 32-bit uint. Buttons. Bits:
    0: +forward.
    1: +moveleft.
    2: +moveright.
    3: +back.
    4: +moveup.
    5: +movedown.
    6: +jump.
    7: +duck.
    8: +use.
    9: +attack.
    10: +attack2.
    11: +reload.
    12: +speed.
    13: +alt1.
- “v” - array. Viewangles. Contains:
    - number. Pitch.
    - number. Yaw.
- “s” - array. Speeds. Contains:
    - number. Forwardspeed.
    - number. Backspeed.
    - number. Sidespeed.
    - number. Upspeed.
- “c” - string. Custom console commands, WHITELISTED.
- “wait” - number, optional (default = 0), >= 0. Number of waits to put into the command buffer.

If type is 1 (saveload):
- nothing else (we’re gonna generate the save name automatically).

If type is 2 (reset):
- “seed” - number, 32-bit int, non-shared RNG seed.

If type is 3 (parameters) - at least one should be present, if any of those is present, set the respective thing:

- “seed” - number, optional, 32-bit int, shared RNG seed.
- “autopause” - bool, optional, enable or disable bxt_autopause.