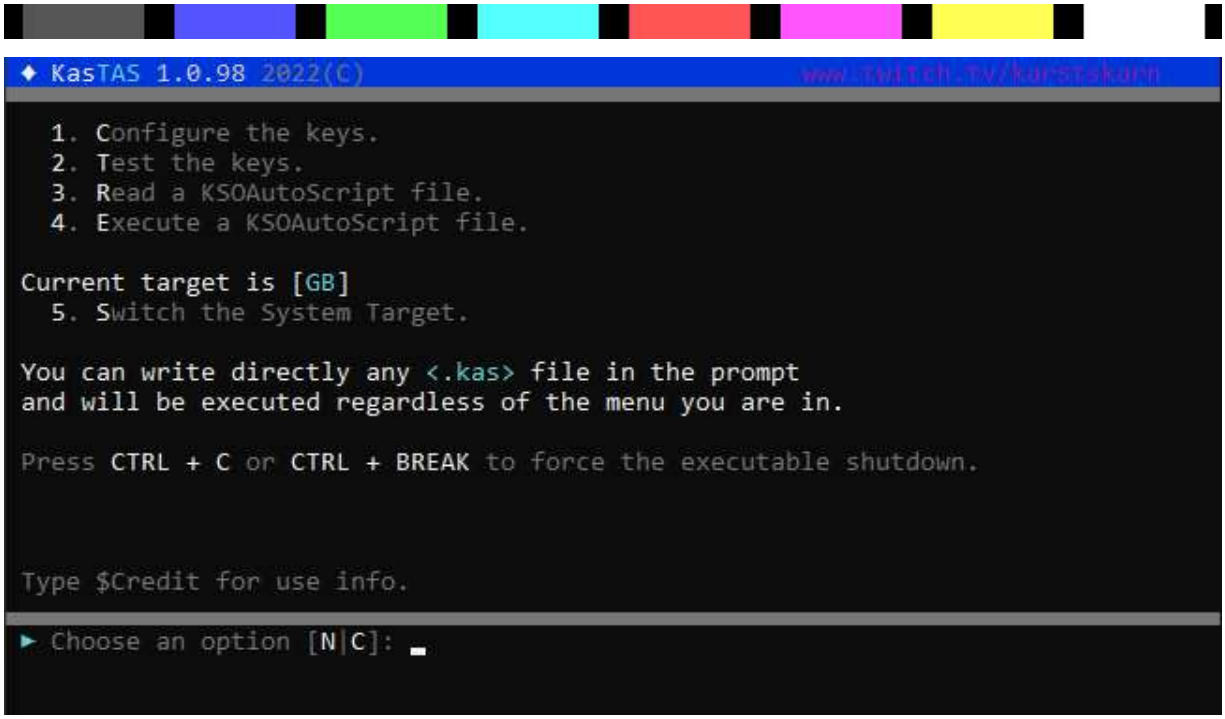

KasTAS ".kas" Scripts

Updated to 1.0.98 (24/01/2022)

By Karst Skarn (Owain#3593)

Introduction

File Format

KasTAS.exe can read virtually any format as long as it contains valid commands (Since now referred to as "instructions"). But is it recommended to use the .kas format file to avoid confusion in case you've got a messy folder.

Is it also recommended to install the included "KAS" language extension if you use Notepad++ or check if the script you write is valid using the built-in function in the KasTAS.exe known as "*3. Read a KSOAutoScript File*".

If an instruction isn't valid or has format issues it will halt the script execution in different ways depending on the kind of error it found.

Remember that you can stop the execution of any ongoing script by pressing CTRL + C or CTRL + BREAK. This will force the executable shutdown.

The ".kas" syntax and instructions

Valid buttons

Game Boy / NES

AA	**A BUTTON**
BB	**B BUTTON**
ST	**START BUTTON**
SE	**SELECT BUTTON**
UP	**UP ARROW BUTTON**
DO	**DOWN ARROW BUTTON**
LE	**LEFT ARROW BUTTON**
RI	**RIGHT ARROW BUTTON**

SNES

XX	**X BUTTON**
YY	**Y BUTTON**
BL	**BUTTON LEFT**
BR	**BUTTON RIGHT**

Buttons are **not forcibly only** for the stated devices. For example enabling the SNES Target mode you can use them for a Sega Genesis because it has the same number of buttons. Just that the names are made in reference to the stated consoles for easy reference.

There's also 10 extra buttons that can be mapped to any valid Windows Key for any extra functionality.

Extra buttons

E0	EXTRA BUTTON 0	E5	EXTRA BUTTON 5
E1	EXTRA BUTTON 1	E6	EXTRA BUTTON 6
E2	EXTRA BUTTON 2	E7	EXTRA BUTTON 7
E3	EXTRA BUTTON 3	E8	EXTRA BUTTON 8
E4	EXTRA BUTTON 4	E9	EXTRA BUTTON 9

Note: To avoid extra delay time checking the use of those buttons their use is disabled by default. To enable them you can set "*01 EXTRA MODE = false*" to "*true*" in the file "**keydata.ini**".

Note 2: You can change the keys of buttons by editing the file "**keydata.ini**". Be sure it is a valid Windows Key specially with special function keys. Refer to Microsoft Windows Forms Keys Enum documentation to be sure of how every key is called.

https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.keys

Instruction table

Here you can see all the 15 valid instructions that KasTAS 1.0.98 accepts.

Control instructions

PB	PRESS BUTTON
RB	RELEASE BUTTON
RA	RELEASE ALL BUTTONS
SP	SOFT PUSH
FP	FAST PUSH
TP	TURBO PUSH
KS	KEY STRING

Time control instructions

HW	HOLD/WAIT
SR	SYNC RESET
SW	SYNC WAIT
SM	SET MARKER
DM	DISPLAY MARKER
CM	COMPARE MARKER

Miscellany instructions

RE	TEXT COMMENT / REMARK
EN	END OF SCRIPT

Control instructions

Control instructions are printed in yellow in the KasTAS.exe console and are the ones used to refer to which buttons are pressed or how long they are pressed.

- **PB Short for "Press Button"**

  Syntax

  PB [BUTTON]

  Press the stated button. It has no time arguments. Note that the stated button will keep being pressed until a "RB" or "RA" instruction is found later in the script.

  Estimated time to process: 0~2 ms.

  Example

  ```
  00 PB ST
  ```

  This example will press the start button.

- **RB Short for "Release Button"**

  Syntax

  RB [BUTTON]

  Releases the stated button. It has no time arguments. If the stated button wasn't pressed at all this instruction will be omitted and won't throw an error.

  Estimated time to process: 0~2 ms.

  Example

  ```
  00 RB AA
  ```

  This example will release the A button.

- **RA Short for "Release All"**

  Syntax

  RA

  Releases all buttons that were pressed until this instruction was readed. If no buttons are pressed the instruction will be omitted and won't throw an error.

  Estimated time to process: 0~2 ms.

Example

```
00 RA
```

This example will release all buttons.

- **SP Short for "Soft Push"**

  Syntax

  SP [BUTTON] [TIME]

  Pushes and holds for the stated time and then releases the stated button. Note that soft push adds 25ms after the release of the button to avoid cases where the targetEmulator/Game/Software doesn't detect the button release because it was too fast. Its functionality is pretty concrete and can be easily interchanged with "FP" instructions.

  Estimated time to process: 0~2 ms + [TIME] + 25 ms.

  Example

  ```
  00 SP BB 100
  ```

  This example will press the B button during 100 milliseconds + 25 milliseconds after releasing it. (A total of 125 milliseconds in this case).

- **FP Short for "Fast Push"**

  Syntax

  FP [BUTTON] [TIME]

  Pushes and holds for the stated time and then releases the stated button. This instruction doesn't add time after the button is released.

  Estimated time to process: 0~2 ms + [TIME].

  Example

  ```
  00 FP UP 100
  ```

  This example will press the Up arrow button during 100 milliseconds and then release it.

- **TP Short for “Turbo Push”**

  Syntax

  TP [BUTTON] [TIME]

  Pushes and releases every ~1 millisecond during the stated time. This function will make the button behave like a burst creating really fast pushes.

  Estimated time to process: 0~2 ms + [TIME].

  Example

  ```
  00 TP AA 500
  ```

  This example will do a burst of pushing the button A during 500 milliseconds and then stop.

  Note: Depending on the target software maybe doesn’t detect all of the times the button has been pushed. There exists the possibility that the last time the button is released isn’t noticed at all by the target software and the button stays pressed in that software. If this happens or there's a possibility of this happening it's recommended to add a “RA” instruction after the “TP” instruction to ensure the target software doesn’t get stuck with the button pressed.

- **KS Short for “Keyboard String”**

  Syntax

  KS [STRING]

  Pushes and holds the stated string. Note that must be a valid Windows keyboard string and the time it keeps it pushed is fixed at 50 milliseconds.

  Estimated time to process: 0~2 ms + 100 ms.

  Example

  ```
  00 KS F4
  ```

  This example will press the F4 keyboard key during 50 milliseconds.

  Note: This function is mainly to communicate with specific and concrete functions that for example an emulator has and is not meant to be used in the creation of the script or macro normal functions. That’s why the time is fixed.

Time control instructions

Time control instructions are printed in blue in the KasTAS.exe console and are the ones used to control time synchronization or time between instructions.

- **HW Short for "Hold / Wait"**

  Syntax

  HW [TIME]

  Waits the stated time and then resumes the script execution.

  Estimated time to process: 0~2 ms + [TIME].

  Example

```
00 PB AA
01 HW 1000
02 RB AA
```

  This example will press the A button, then wait 1000 milliseconds and then release the A button.

- **SR Short for "Sync Reset"**

  Syntax

  SR

  Depending on the user computer the execution of a script can be slightly delayed and this delay can accumulate over time specially in long scripts. All the unexpected delay is kept on record; for example if there's a "FP AA 500" instruction that will keep the A button pressed for 500 milliseconds but at the end it took 502 milliseconds, that difference of 2 milliseconds will be added to an internal record.

  Every time an SR instruction is found this record is set to zero which means it is effectively reseted. This is meant to work with the SW instruction. Check the SW instruction below for more info.

  Estimated time to process: 0~2 ms + [TIME].

  Example

```
00 SR
```

  This will simply reset the accumulated delay counter.

- **SW Short for "Sync Wait"**

  Syntax

  SW [TIME]

  Waits the stated time but it subtracts the delayed time. This is useful for example if there's a precise button push you want to be done but for some reason some delay did accumulate over the execution script and you want to get rid of it.

  It will count the time since the last "SR" instruction and if there's none since the script execution did begin.

  If the "SW" instruction had a long enough time to compensate for all the delay the milliseconds of delay erased will appear in green at the right of the instruction line in the execution window. Otherwise it will appear as red and will mean that despite erasing some delay it stills existing some.

  Estimated time to process: 0~2 ms + [TIME] - [SYNC DELAY].

  Example

```
00 SR
01 PB BB
02 HW 500
03 RB BB
04 TP AA 100
05 FP UP 150
06 FP LE 150
07 FP DO 150
08 SW 1000
```

  This example will press do some random button presses and suppose that it did accumulate 10 milliseconds of delay meanwhile it was executing all those instructions it will wait 1000 milliseconds - 10 milliseconds resulting in a real wait of 990 milliseconds.

- **SM Short for "Set Marker"**

  Syntax

  SM

  All milliseconds elapsed in a script execution are counted. This allows the script to be really precise if these instructions are properly used. The "SM" instruction resets the counter of milliseconds elapsed to zero. Note that will count the milliseconds that the instruction itself takes to realize the operation afterwards.

  Estimated time to process: 0~2 ms.

  Example

```
00 SM
```

  This will simply reset milliseconds elapsed and set it to zero.

- **DM Short for “Display Marker”**

  Syntax

  DM

  This instruction will show the elapsed time until this instruction was found. Is meant to be used as a debug feature or quick visual control feature to be able to calibrate properly the “CM” exposed below.

  Estimated time to process: 0~2 ms.

  Example

```
00 SM
02 PB BB
03 HW 500
04 RB BB
05 TP AA 100
06 FP UP 150
07 DM
```

  This will display the number of milliseconds elapsed until the DM instruction was found. In this case it should be more or less ~750 milliseconds.

- **CM Short for “Compare Marker”**

  Syntax

  CM [MILLISECONDS]

  This instruction will check if the stated milliseconds are ahead or behind the current elapsed milliseconds. If the execution current milliseconds are behind the stated milliseconds in the instruction it will wait until the number matches. If the execution time is ahead of the stated milliseconds it will skip the instruction and won’t wait.

  Note that this instruction doesn’t reset in fact the millisecond counter like “SM” does.

  Estimated time to process: 0~2 ms.

  Example

```
00 SM
01 PB BB
02 HW 500
03 RB BB
04 TP AA 100
05 FP UP 150
06 CM 800
```

  This will wait until the millisecond counter gets to 800 to resume the script execution. Taking in consideration that all the instructions behind the “CM” would take approximately ~750 milliseconds. This “CM” instruction in this case will wait approximately ~50 milliseconds.

Jump instructions

Jump instructions are printed in dark magenta in the KasTAS.exe console and are the ones used to jump to another line of script. For example used in repetitive sequences.

- **JM Short for "Jump"**

  Syntax

  JM [LINE]

  This instruction will jump to the stated line. Beware this instruction will create an endless loop if the line is behind this instruction and must be used carefully because of that.

  Note: In the script execution the first line is always considered the number 0. Consider this with editing with notepads that start in line 1.

  Estimated time to process: 0~2 ms.

  Example

```
00 PB AA
01 PB BB
02 HW 500
03 RA
04 SP ST 100
05 TP AA 500
06 JM 3
```

  This will execute the script and when it finds the "JM" instruction will jump to the 3rd line and continue the execution from there creating a loop.

- **JT Short for "Jump X Times"**

  Syntax

  JT [TIMES] [LINE]

  This instruction will jump the stated times to the stated line. Once the instruction has been found and executed the stated amount of times it will be ignored and will continue the execution beyond that point.

  Note: Only one "JT" instruction is considered as active if it finds another "JT" instruction in the script between the goal jump line and the current jump instruction; the first one will be forgotten and the active "JT" instruction will be set to the last one it did found.

  This can potentially cause unexpected behaviors depending on the context of the whole script or can end up causing odd infinite loops.

Note 2: In order to keep the format integrity, if the number of times the jump must be done is less than 10 the number must be written with a zero in front. For example "JT 4 3" isn't valid while "JT 04 3" is the valid way to be written. Otherwise the instruction will be considered as an error.

Note 3: Following the **Note 2** rule the maximum number of repetitions is actually 99 because this argument doesn't support more (or less) than two digits.

Estimated time to process: 0~2 ms.

Example

```
00 PB AA
01 PB BB
02 HW 500
03 RA
04 SP ST 100
05 TP AA 500
06 JT 04 3
07 PB AA
08 HW 1000
09 RB AA
```

This will jump 4 times to the line 3 and then the execution will continue past the 6th line and continue as normal.

Miscellany instructions

In the version 1.0.98 there's only two additional instructions which use isn't essential as the ones stated previously.

- **RE Short for "Remark / Text Comment"**

  Syntax

  RE [TEXT]

  This instruction is made to be able to quickly and easily label or remark a part of the script. Anything written after the "RE" instruction won't be readed as instruction data and will be just shown on screen.

  Estimated time to process: 0~2 ms.

  Example

```
00 SM
01 PB BB
02 HW 500
03 RE HELLO!
04 TP AA 100
05 FP UP 150
06 CM 800
```

  This will simply show "HELLO!" at the third line in the KasTAS script execution window.

- **EN Short for “End”**

  Syntax

  EN

  This instruction ends the script execution. It's not mandatory to be introduced at the end of the script because anyways the execution will end when it finds no more lines to read but it may be useful to end a script execution if the end isn’t the last line.

  Estimated time to process: 0~2 ms.

  Example

```
00 RE SCRIPT START
01 PB BB
02 HW 500
03 JM 8
04 TP AA 100
05 FP UP 150
06 TP AA 100
07 EN
08 HW 250
09 FP UP 150
10 TP AA 100
11 JM 4
```

  This will jump from line 3 to the line number 8 skipping the ones in between and then it will jump from line 11 to the line number 4 where it will execute some instructions until it reaches line 7 where there’s an “EN” instruction and the script will end there.

Considerations

When you use the built-in function in the KasTAS.exe known as “*3. Read a KSOAutoScript File*” errors will appear in red. Anything that doesn’t fit the specified formats explained in this document will be treated as an error and may cause unexpected behavior.

Take special caution with spaces because since there’s no end marker as in other languages (“;”) the spaces are mainly what determine the data structure.

Normally extra spaces after for example a [TIME] or a [BUTTON] value won’t cause a fatal error but if it's possible avoid them.

The errors that can lead to a code execution unexpected stop are mostly the ones that are the result of not respecting the single space between words in the instruction. These ones are detected in the “Read Mode”. Also if you use “JT” instructions read carefully the “JT” notes that this document states.

Despite all the stuff stated above no error will corrupt/damage the script in any way and just fixing the cause will make it work again.

In case of getting the executable stuck in for example an infinite loop between jumps or a series of unexpected script format errors, remember that **CTRL + C** or **CTRL + BREAK** will immediately close the program without any problem.