SoniFight User Guide



Alastair Lansley / Federation University Australia

[a.lansley@federation.edu.au](mailto:a.lansley@federation.edu.au)

Contents

[Introduction 3](#_Toc485521611)

[Demonstration / Quick Start 4](#_Toc485521612)

[Download and Installation 5](#_Toc485521613)

[File Structure 6](#_Toc485521614)

[User Interface Elements – Main Tab 7](#_Toc485521615)

[User Interface Elements – Edit Tab 8](#_Toc485521616)

[Edit Tab – Tree View 8](#_Toc485521617)

[Edit Tab – Details Panel 9](#_Toc485521618)

[Creating a New Config 10](#_Toc485521619)

[Creating and Using Watches 10](#_Toc485521620)

[The Clock Trigger 11](#_Toc485521621)

[Finding Pointer Trails 11](#_Toc485521622)

[Pointer Trail Checker 11](#_Toc485521623)

[Creating and Using Triggers 11](#_Toc485521624)

[The Clock Trigger 12](#_Toc485521625)

[Normal Triggers 12](#_Toc485521626)

[Continuous Triggers 12](#_Toc485521627)

[Modifier Triggers 12](#_Toc485521628)

# Introduction

SoniFight is utility software to provide additional sonification to fighting games for visually impaired players.

SoniFight is written in C# and licensed under the MIT software license. The source code is freely available for use and modification at: <https://github.com/FedUni/SoniFight>. Please see LICENSE.txt for further details, including separate licensing details for the embedded irrKlang audio library.

To run SoniFight either download a precompiled binary release or build the Visual Studio 2015 solution yourself, then launch the SoniFight executable, choose a game config for the game you want to play, click the "Run Selected Config" button and then launch the game that the selecting game config targets.

SoniFight presently ships with game configs to add sonification to Ultra Street Fighter IV Arcade Edition and Mortal Kombat 9 (aka Mortal Kombat Komplete Edition).

Once running, SoniFight will provide additional sonification cues such as clock, health and bar status updates for both players. In the Street Fighter game config, there are also triggers for a large number of menu options so that there is less need to memorise sequences of menu selections.

SoniFight also provides a user interface where you can create your own game configs for games of your choice, although the process to find pointer trails requires additional free software and can be a little bit tricky and time consuming.

To learn more about creating your own game configs as well as how the software operates through 'watches' and 'triggers' please see relevant sections in this user documentation.

# Demonstration / Quick Start

If you want to quickly get an idea of what the SoniFight software can do then a demonstration video is available at the following location:

LINK\_HERE

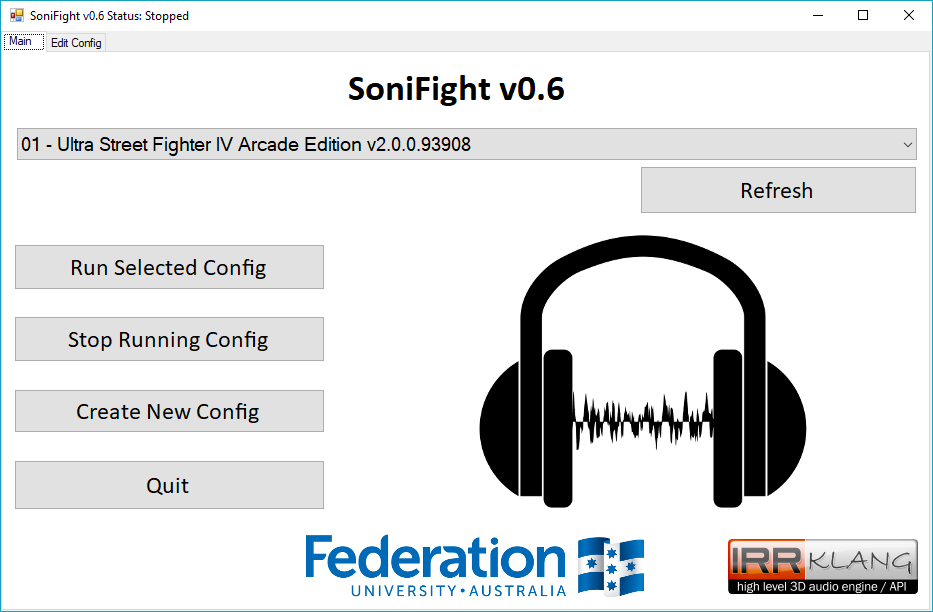


Figure 1 – The SoniFight software user interface.

To run the SoniFight application, the .NET framework version 4.6.1 or later must be installed on your computer. If you do not have this installed on your computer it is freely available from Microsoft at the following URL:

<https://www.microsoft.com/en-au/download/details.aspx?id=49981>

# Download and Installation

If you just want to use the software then you can download a precompiled binary release from:

<https://github.com/FedUni/SoniFight/releases>

Once downloaded, extract the zip file and run the SoniFight executable within.

With SoniFight running, select a game config from the dropdown menu, click the **[Run Selected Config]** button and then launch the game related to the game config you’ve chosen to provide sonification.

If you want to build the software from source then you can either download a zip of the latest files from:

<https://github.com/FedUni/SoniFight>

Or if you have git source control tools installed, such as those from <https://git-for-windows.github.io/>, then you can type the following into the command prompt to clone the current master branch of the repository:

**git clone https://github.com/FedUni/SoniFight**

Once downloaded you can open the SoniFight solution in Visual Studio 2015 to build it for yourself. If you do not have Visual Studio 2015, then the Community Edition can be freely downloaded from Microsoft at:

<https://www.visualstudio.com/downloads/>

To successfully run the software, regardless of how it’s obtained, you will need the .NET framework version 4.6.1 or above installed on your computer, which may be freely obtained from:

<https://www.microsoft.com/en-au/download/details.aspx?id=49981>

# File Structure

Releases are provided as a zip archive containing pre-compiled versions of the SoniFight executable including provided game configs and the pointer trail tester utility in the following structure:

**PUT DIRECTORY STRUCTURE TREE HERE – USE TREE COMMAND THEN SCREENSHOT**

Figure 2 - The SoniFight release directory structure.

The **build.txt** file identifies the overall version of the SoniFight software. This version number may not necessarily match the individual versions of the SoniFight and pointer trail tester components which may change independently, however any increment of either component's version number will result in an increment of this overall build version number.

The **Configs** folder contains game configs for various games, where each config has its own subdirectory.

# User Interface Elements – Main Tab

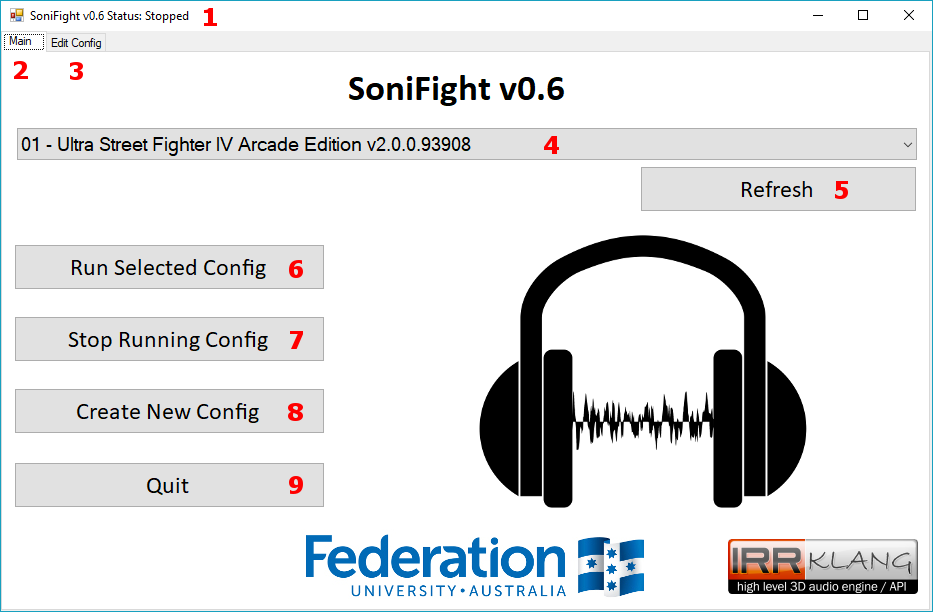


Figure 3 - SoniFight Main tab user interface elements

The numbered elements in the above figure are as follows:

1. **Title Bar** – Displays the status of whether SoniFight is stopped or running a given game config.
2. **Main tab** – The screen which provides functionality to select a game config and start or stop it.
3. **Edit tab** – The screen which provides functionality to modify a game config.
4. **Config dropdown menu** – The dropdown menu which selects which config to use when the **Run Selected Config** buttonis clicked.
5. **Refresh button** – If you copy a new folder into the Configs directory you can click this button to refresh the config dropdown menu so that it contains the new folder as an available option instead of needing to stop and restart the SoniFight software.
6. **Run Selected Config button** – Runs the SoniFight software to enable additional sonification as specified in the selected game config. You do not need to launch the game process before clicking this as it will happily wait while attempting to connect to the game config’s process without issue.
7. **Stop Running Config button** – Stops the currently running game config.
8. **Create New Config** – Creates a new, blank game config and switches to the **Edit tab**. A folder with the name of the config will be created, within which the **config.xml** file will exist, when the config is saved.
9. **Quit button** – Exits the SoniFight software.

# User Interface Elements – Edit Tab

The edit tab is used to modify game configs and is broken up into two main sections – the Tree View and buttons on the left third of the screen and the details panel on the right two-thirds of the screen.

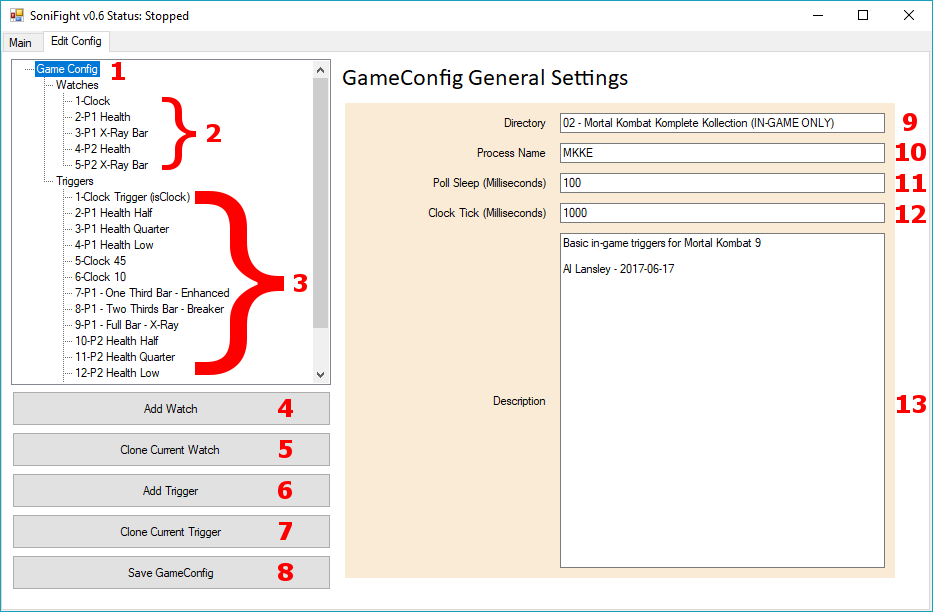


Figure 4 - The Edit Config tab with the main Game Config tree node selected

### Edit Tab – Tree View

The tree view is where you can choose to modify the game config settings including its main options, its watches and its triggers. You can, of course, modify the config.xml file manually using a text editor if you so choose, but great caution is advised as a single bad character in the wrong place will cause the config to become ‘corrupted’ and unable to be loaded. Typically, leave the config modification to from within the SoniFight user interface unless you’re confident you know precisely what you’re doing.

Clicking on the main ***Watches*** or ***Triggers*** nodes displays a brief description what they are.

The numbered elements in the above figure are as follows (further details on each are available in their individual sections below):

1. **Game Config** – The main config settings node holds details of the directory the config lives in, the process name to query the base address of, the poll sleep delay, click tick delay and description.
2. **Watches** – These tree nodes specify the pointer trail to a memory location and the type of data to read from that location. In essence, they ‘watch’ a memory location and read a value from it every poll sleep milliseconds.
3. **Triggers** – These tree nodes specify which watch to read data from, and the conditions under which they should ‘trigger’ a sonification event.
4. **Add Watch** – Adds a new, blank watch with a unique Id.
5. **Clone Current Watch** – Creates a new watch with a unique Id value which is populated with the details of the currently selected watch. The text “-CLONE” is appended to the name of the new watch to easily distinguish it from the original.
6. **Add Trigger** – Adds a new, blank trigger with a unique Id.
7. **Clone Current Trigger** – Creates a new trigger with a unique Id value which is populated with the details of the currently selected trigger. The text “-CLONE” is appended to the name of the new trigger to easily distinguish it from the original.
8. **Save GameConfig** – Saves the current configuration settings to the ***config.xml*** file within the directory specified for the game config.

### Edit Tab – Details Panel

The details panel is where you can edit and view configuration settings of a game config, its watches and its triggers. The available options will vary depending on which of these items is selected in the left-side Tree View. For the main Game Config tree node, the available options are:

1. **Directory** – The directory to save this game config to. If you change this value you will need to manually copy any required audio samples into the new directory for the config to work.

1. **Process Name** – The name of the process to connect to. You can find the name of a running process by looking at the running processes in Windows Task Manager (shortcut: Ctrl+Shift+Esc). The process name should not contain the .EXE suffix of the executable.
2. **Poll Sleep (Milliseconds) –** The length of time in milliseconds (i.e. one thousandths of a second) to wait before polling all watches and triggers e.g. if this value is 100 milliseconds then SoniFight will poll for changes 10 times per second. The value must be between 1 and 200 milliseconds, where smaller values will poll more often and use higher CPU. Typically, a value of 100 milliseconds is sufficient to provide a sonification cue in a tenth of a second while using very little CPU resources.
3. **Clock Tick (Milliseconds) –** The estimated length of time in milliseconds that one ‘tick’ of the game clock (aka round timer) takes. This is used to help keep track of the game state so SoniFight knows we’re either in a live game or in the menus by enforcing that at least two ‘ticks’ have passed before allowing any triggers marked as “In-Menu” to provoke sonification events.
4. **Description** – This is simply an optional multi-line text box where you can write some details regarding the game config should you wish.

# Creating a New Config

To create a new, blank game config click the **[Create New Config]** button from the main tab.

To save the current game config in a different directory, change the directory name of an existing config and then click the **[Save GameConfig]** button. Please note that you will need to manually copy all the audio samples from the original to the new config directory for them to be used – that is, samples are not shared between directories and all samples used by a game config must exist in the same directory as the ***config.xml*** file itself.

# Creating and Using Watches

To create a new, blank watch click the **[Add New Watch]** button from the edit tab.

To clone an existing watch, select a watch from the left-hand watch sub-tree and then click the **[Clone Current Watch]** button. After which a new watch will exist with the details copied from the original watch and the word “CLONE” added to the name.

A **Watch** itself is simply a few pieces of data that help to locate a memory address and the type of data that should be read from that memory address. However, the watch address isn’t a single value – as due to technologies such as Address Space Layout Randomization (ASLR) and the current state of memory usage on the host machine, a single stored memory address would not be sufficient to consistently reproduce the location of a given game value such as the clock or a player’s health.

Instead, a watch must use a manner of ***relative addresses*** in the form of a ***pointer trail***. This is a series of ‘jumps’, starting at the beginning of where the game process is loaded into memory, that will always lead us to the memory address of a value of interest such as the clock etc.

A Watch has the following user interface elements:

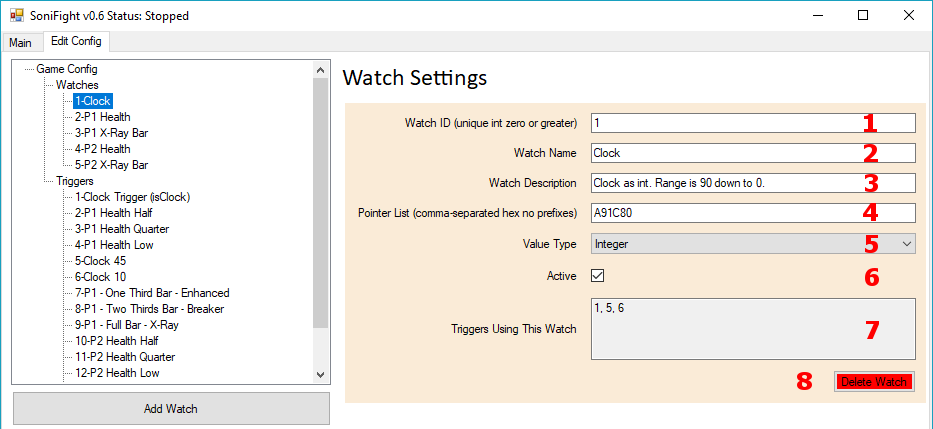


Figure 5 – Watch configuration details.

The watch details in the above figure are as follows:

1. **Watch ID** – This is a positive integer value that uniquely identifies this watch for use by triggers. Each new watch will be assigned a new unique value, or you may assign your own. Once set, it’s not recommended to change the watch ID because any triggers which use a watch will not have the associated watch ID automatically updated if the watch ID is changed.
2. **Watch Name** – A name for the watch, optional but useful.
3. **Watch Description** – A description for the watch, again optional but useful.
4. **Pointer List** – This is a series of one or more comma-separated hexadecimal values used to offset from the game process’ base address to find a useful value such as the clock or player health. Do not include any 0x prefixes or such to indicate that offsets are in hexadecimal format. Further details on how pointer lists work internally can be found in the **How SoniFight Works** section of this document.
5. **Value Type** – The above ***Pointer List*** provides enough information to locate given memory address, but once there we need to know what type of data we should read from that memory address. This dropdown provides the following options for data types to read from the address:
   * **Integer** (4 bytes),
   * **Short** (2 bytes),
   * **Long** (8 bytes),
   * **Float** (4 bytes),
   * **Double** (8 bytes),
   * **Boolean** (1 byte),
   * **String (UTF-8)** – 1 byte per character, read up until the null terminator or 33 chars and trimmed to remove whitespace,
   * **String (UTF-16)** – 2 bytes per character, read up until the null terminator or 33 chars and trimmed to remove whitespace.
6. **Active** - This checkbox is used to toggle whether this watch is in use or not. The default is checked (active). If the checkbox is unchecked then this watch will not be polled, and as such cannot activate any triggers that might depend upon it. Sometimes you may choose to deactivate a watch in a given game config without having to delete its data – this is the mechanism to do so.
7. **Triggers Using This Watch** – This read-only textbox simply displays the ID values of triggers which depend upon this watch so you can easily tell if it’s important or not. If no triggers depend on this watch then the value displayed will be ***None***. Note that the active status of any given watch or trigger does not affect whether a given trigger ID may be displayed here – if the trigger depends on the watch, then it will show up regardless of whether either is active or not.
8. **Delete Watch Button** – Deletes the currently selected watch. There is no undo option, but the change is not permanently applied until the **[Save GameConfig]** button is clicked, so if you wanted to revert to the last saved version then you could either:
   * Quit and restart the software then select the config you want to reload, or
   * Switch back to the main tab and select a different config from the dropdown menu, then click the edit tab so it loads, then go back to the main tab and select the config you were originally on to reload the previously saved version of it and click the edit tab again to edit it.

# Creating and Using Triggers

To create a new, blank trigger click the **[Add New Trigger]** button from the edit tab.

To clone an existing trigger, select a trigger from the left-hand trigger sub-tree and then click the **[Clone Current Trigger]** button. After which a new trigger will exist with the details copied from the original trigger and the word “CLONE” added to the name.

A **Trigger** is a condition that we check against to determine whether we should play a sample (i.e. provide a sonification event) or not. Each trigger has its own unique ID, but also depends on a watch (as specified by the watch ID) along with some criteria such as equals, more than, less than etc. and a value that must match that criteria.

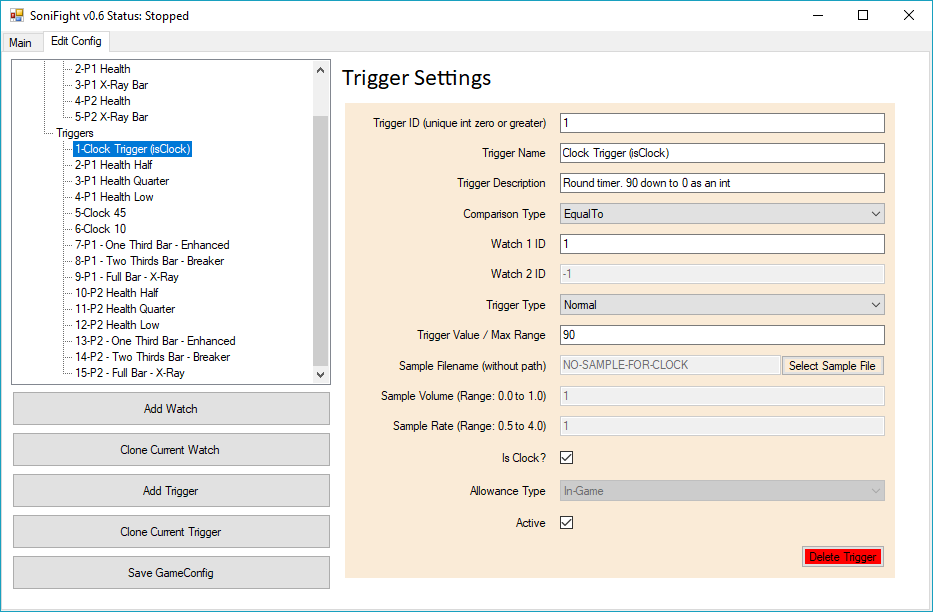


Figure 6 - Trigger configuration details

The trigger details in the above figure are as follows:

1. **Trigger ID** – This is a positive integer value that uniquely identifies this trigger. Each new trigger will be assigned a new unique value, or you may assign your own unique value. As triggers depend on watches but not vice versa you are free to change the trigger ID as you please, so long as the trigger ID remains unique.
2. **Trigger Name** – A name for the trigger, optional but useful.
3. **Trigger Description** – A description for the trigger, yet again optional but useful.
4. **Comparison Type** – The comparison type is how we compare the trigger value (described below) against its previous value. The comparison must pass through this threshold to trigger a sonification event. Available comparison types are as follows:
   * **Equal To** (threshold: Not Equal To),
   * **Less Than** (threshold: Greater Than Or Equal To),
   * **Less Than Or Equal To** (threshold: Greater Than),
   * **Greater Than** (threshold: Less Than Or Equal To),
   * **Greater Than Or Equal To** (threshold: Less Than),
   * **Not Equal To** (threshold: Equal To),
   * **Changed** (threshold: Not Equal To Previous Value), or
   * **Distance** (no threshold – only valid for triggers with a Trigger Type of **Continuous**).
5. **Watch 1 ID** – The ID of the watch associated with this trigger (e.g. the memory location from which we read a value from to compare against the above comparison type).
6. **Watch 2 ID** – The ID of the secondary watch associated with this trigger. Only used if the comparison type is ***Distance*** and the Trigger Type (described below) is ***Continuous***.
7. **Trigger Type –** The type of the trigger, out of the following options:
   * **Normal** – A standard trigger which may play once or many times per round while ***In-Game*** or ***In-Menu***, depending on how it passes the threshold of its comparison type,
   * **Continuous** – There may only be a single continuous trigger in any given game config. Its role is to loop a sample (i.e. play it at all times while the game state is ***In-Game***). A continuous trigger is the only type of trigger which may have a secondary ***Watch 2 ID*** from which it reacts to changes between the 1st and 2nd triggers when the ***Comparison Type*** is ***Distance***.
   * **Modifier –** Changes the continuous trigger somehow… or whatever.
8. **Trigger Value / Max Range** – For a trigger with a type of ***Normal*** or ***Modifier*** this is the value that the comparison type must meet while passing the threshold to actually trigger a sonification event. For a ***Continuous*** trigger then this number means **Max Range** which specifies the maximum difference between the numbers for the continuous triggers sample to hit zero (while when the calculated distance between the watch 1 and watch 2 values is 0 the resulting ‘distance’ will be 1.0).
9. **Sample Filename** - blah

### The Clock Trigger

To do.

### Finding Pointer Trails

To do.

### Pointer Trail Checker

To do.

# Creating and Using Triggers

To do.

### The Clock Trigger

To do.

### Normal Triggers

To do.

### Continuous Triggers

To do.

### Modifier Triggers

To do.