Nintendo 3DS CPU Profiler Manual

Nintendo 3DS CPU Profiler

2015/10/23

Version 4.05

The content of this document is highly confidential and should be handled accordingly.

Contents

1 Overview

The Nintendo 3DS CPU Profiler is a statistical sampling profiler that helps detect which functions are using significant CPU resources. The profiler also has the ability to instrument one function at a time to accurately and precisely measure its use of CPU resources.

1.1 Installation

Unzip the profiler package to a directory of your choice.

Nintendo 3DS Specific Installation Instructions

Open the `runtime\CTR\SDK-`*`SDKVersionNumber`* folder corresponding to your SDK revision and copy the contents to your SDK install directory. If there is not an *SDKVersionNumber* for your SDK revision, use the folder corresponding to the nearest previous SDK revision. For example, if you have SDK 7.*x*, use the contents of the folder `SDK-4_1_0`.

Please see Setup for more information on how to install the profiler onto your development equipment.

1.2 Requirements

The profiler requires the following:

General Requirements

- Windows 7 64-bit or Windows 10 64-bit
- DirectX 10
  - For all of the required libraries, the DirectX End-User Runtimes (June 2010) must be separately installed onto the computer. The DirectX installer can be found for download from http://www.microsoft.com/download/en/details.aspx?id=8109.
- .NET 4.5 Runtime (search Microsoft.com for ".net 4.5 runtime" or visit http://www.microsoft.com/en-us/download/details.aspx?id=30653).

Nintendo 3DS Specific Requirements

- SDK 11.1.0 or later
- Firmware 0.24.52 or later
- One of the following:
  - PARTNER-CTR version 5_70-051 or later with HOSTIO Daemon 1.03.015 or later
  - IS-CTR-DEBUGGER and IS-CTR-HIO version 3.43 or later
- The profiler makes use of the compiler in order to demangle C++ function names. This is located via the Window's environment variable for the compiler. Either the ARMCC 5 or 4.1 can be installed to make use of this feature. If both are installed, the demangler from ARMCC 5 will be used.
- The profiler makes use of certain Cygwin features for the **Assembly** tab. Specifically, the `binutils` package for Cygwin will need to be installed in order for the profiler to have access to the `addr2line` application available therein.

1.3 Limitations

The profiler has a few limitations:

- When attempting to profile on a SNAKE kit, the System Mode cannot be set to dev1 when attempting to profile a standard (CTR) application. Only extended applications can be profiled under the dev1 System Mode. Attempting to profile a standard application will result in a timeout occurring when attempting to **Sync** the profiler.
- Currently the thread stack must be read into the profiler and then parsed. Because of this, the profiler can only support stacks that are 64 KB deep. If the stack is deeper than 64 KB, only the most recent 64 KB will be reported back, potentially leading to stacks containing UNKNOWN STACK DETAILS or shortened callstacks.
- It is not be permitted for a game to ship with the profiler code in the executable. Please disable all profiler related code prior to submission (see Nintendo 3DS CPU Profiler Game API for instructions on disabling the Nintendo 3DS CPU Profiler). Additionally, the profiler library should not be linked against your project.

1.3.1 Using with the Home Menu

The profiler cannot be used with the HOME Menu. If an application is launched from the HOME menu, only a black screen will be displayed. You must use the Test Menu when profiling. To switch into the Test Menu, open the Config tool, go to Other Settings, and then Menu. Switch from HOME Menu to Test Menu, and then restart your kit. The kit should boot up into the Test Menu, which is text only. When you are done profiling, you can switch back to the HOME Menu by the same process.

1.4 Resources

The profiler uses a portion of the available system resources. This decreases the amount of resources available for use in your game while it is being profiled. The following is the list of the resources used:

- *BufferSize* + 600 KB of Main Memory. (Where *BufferSize* is the size set in the GUI.)
- `nn::os::Thread` objects: three plus the number of cores used.
- `nn::os::Event` objects: four plus the number of cores used.

2 Taking a Profile

2.1 Setup

To initially setup the profiler for use with your game, please follow these steps.

1. Start the Debugger
2. Save the Profiler to NAND
    - For **PARTNER-CTR Debugger**: In the command window, type `nand write` *`PathToProfilerCIA`* (for example, `nand write C:\profiler\runtime\CTR\SDK-4_1_0\profiler.cia`).This will write the `profiler.cia` file to NAND. After it is written, please reboot the debugger and dev kit.
    - For **IS-CTR-DEBUGGER**: In the Open dialog, select CIA from the filetype list box and browse to the `profiler.cia` file.
3. Optional: Mark frames in your application following the steps in Frame Marking on Nintendo 3DS.

2.2 Profiling

The following steps show how to take a profile. If you run into problems, please look at Limitations or Troubleshooting. If you are still having problems after consulting those sections, please contact your local Nintendo support group.

1. Start the debugger and connect to the desired dev kit.
2. Start the Profiler by running the application "`Nintendo CPU Profiler.exe`" located in the `bin` folder.
3. Sync the Profiler by clicking the blue **Sync** button on the ribbon bar.
    - Find the files requested by any dialog boxes that appear.
4. On the **Sampled Profile** tab, select the sampling strategy, rate, and other options.
5. Click on the green **Start** button to start profiling.
6. Stop Profiling
    - To manually stop, click on the red **Stop** button.
    - To automatically stop after a set amount of time, use the drop-down just to the right of **Start** and **Stop** buttons to select a time.
    - Alternatively, profiling will automatically stop once the recording buffer is full (this may take awhile).
7. Examine the profile in the **Functions**, **Info**, **Call Tree** (if **Callstack** mode was enabled), **Sample Graph**, and **Code Coverage** tabs.
8. Optional: Click the **Save** button to save the profile.

3 User Logged Data and Events

While the profiler can be used with very little manual integration, it can be extremely useful to augment a profile with user logged data and events. The following describes available options.

3.1 Heartbeats (Frames)

When looking at data over long time periods, it is helpful to subdivide the data into meaningful groups, such as frames. Since multiple systems and different cores may operate in different rhythms, we've come up with the broader term "heartbeat" to denote any regularly occurring rhythm or framing that is useful to track.

If a heartbeat was logged during a profile, it will appear in one of the drop down boxes in the **Heartbeats** tab. When selected, the heartbeat will be graphed for that particular core. You'll notice that there are always two other heartbeats in the drop down boxes: **Fixed 60Hz Intervals** and **Fixed 1ms Intervals**. These heartbeats weren't actually recorded in the profile data, but you can select them to overlay timing information onto a particular core. Please note that these two heartbeats are unlikely to correlate with your data and are only offered as a timing reference.

If you wish to focus on particular heartbeat frames, you can select them either by clicking and dragging over the frames you want to select or by clicking on a particular frame rate in the **Heartbeats** tab (for example, the **60Hz** toggle button).

3.1.1 Frame Marking on Nintendo 3DS

The game does not need to be modified to be profiled, but detailed frame information can be tracked by adding notifications to the profiler. This will allow you to better analyze and interpret the profile results.

Note: It is required that you use at least one function from the API in order to be able to take an **Instrumented Profile**.

1. Call `nnprofRecordTopMainLoop` at the top of your main loop.
2. Call `nnprofRecordWaitForVBlank` before the game waits for vertical blank.
3. Include the profiler header.

```
#include <nn/prof.h>
```

4. Link against the profiler library.

```
LIBS += profiler
```

3.1.1.1 Automatic Frame Detection

The profiler supports the automatic detection of frames in the event that the game isn't explicitly marking frame boundaries, but it is **not recommended** that you rely on this backup method. This backup method uses calls to `nngxWaitVSync` as frame boundaries but is a poor substitute compared with manual frame marking. Using this call as the beginning of the frame isn't correct, so be aware of this situation and take it into consideration when analyzing a profile.

If the profiler does not find the `nnprofRecordTopMainLoop` function in your game Automatic Frame Detection mode will be enabled.

We highly recommend that you manually mark the beginning of the frame within the game as described in Frame Marking on Nintendo 3DS, as well as manually marking the vertical blank so that both will be marked and displayed within any profiles.

3.1.2 Inferred Heartbeats

While it is still critical for you to log the Main heartbeat, the profiler will infer heartbeats on each core based on function behavior. If there are functions that regularly execute on a periodic basis, the profiler will use the most periodic functions as an indicator of a heartbeat rhythm on that core. Inferred heartbeats are more likely to be detected at higher sampling rates (i.e. 100x or higher), since there is more data per frame to analyze. These inferred heartbeats are then selectable in the **Heartbeats** tab. Additionally, the **Info** tab has a list of all inferred heartbeats that were detected. Unfortunately, since these heartbeats are educated guesses based on the data, they are unlikely to actually line up with the conceptual beginning of a frame. If this is a problem, it is recommended that you explicitly log the heartbeat to accurately mark where the frame begins.

3.2 Marked Code Blocks

Many game developers wrap their game systems with timers so that exact running times can be easily tracked during gameplay. The profiler supports logging these marked blocks of code with the added benefit of being able to track CPU performance counters along with the execution time. When coupled with the profiler, you can use the profiler to graph all of these metrics along with a profile. Please see the Nintendo 3DS CPU Profiler Game API documentation for details on integrating the code block feature.

Once your code has been manually instrumented with code blocks, you must enable the recording of them in the profiler. To do so, use the **Marked** button on the **Instrumented Profile** tab. You can then select which performance counter group to record along with your marked code blocks.

Once recorded in a profile, the marked code blocks will be listed on the **Instrumented** tab. When selected on the **Instrumented** tab, they will be graphed in the **Sample Graph**.

4 Profiler Workflows

When using the profiler, there are many possible workflows to discover or track down problems. The following is a list of interesting use cases.

4.1 Workflow Limitations on Nintendo 3DS

Some of the following workflow descriptions state that a sampling rate of **1000x per frame** should be selected. Others may ask for **Callstack** profiles. On the Nintendo 3DS these two options can increase the profiler overhead, changing game timing. At high sampling rates, operations that occur at fixed periods will appear to take more time. A common example of this is the audio system, where an interrupt must be handled every 4.889 milliseconds.

4.2 Quickly Understanding Code Flow

Use the **Story Analysis** buttons on the **Analysis** ribbon bar tab to quickly get a handle on the code flow within a frame. This is the fastest way to understand the behavior within a frame.

Setup

- Record at least **1000x per frame**.
- Record a **Callstack** profile.
- A long profile is not necessary since you will be looking at a particular frame.

Procedure

1. In the **Functions** tab, select the core you want to examine using the Core Filter (described in Core Filtering and Column Heading).
2. In the **Analysis** tab in the ribbon bar, click on the **Brief** button in the **Story Analysis** group.
3. In the **Sample Graph** tab, zoom into a typical frame or a frame that you want to deeply understand. At this point you can see the behavior from the start of the frame to the end.

   Tip: There are a few things that can help make it easier to view the data.

   - Click the **Load** button in the **Sample Graph** toolbar to temporarily hide the **Load per Frame** graph. Click it again to bring it back.
   - Use the **Samples** drop down in the **Sample Graph** toolbar to adjust the vertical spacing of samples. A larger vertical spacing makes it easier to visually track a sample back to its function name of the left.
   - You can always order the samples by chronological order by right clicking on the **Sample Graph** and choosing **Reorder** > **Reorder Chronologically**.
   - You can quickly drill down into a function's children by double clicking on the function line in **Sample Graph**. To undo what you just expanded, remain on the same function line, right click, and choose **Deselect Direct Children**.

4. Optional: Select the frame you are looking at and click **Brief** again. This will only show interesting functions on this frame, sorted correctly for this frame. To select a frame, change the drop down mouse mode on the far left of the **Sample Graph** toolbar to **Select Frames** and click on your desired frame once.
5. To get a more detailed story, click on **Full**, **Fine**, or **Top** in the **Story Analysis** group.

   You can adjust the settings on the left side of the **Story Analysis** group or you can adjust the parameters within a particular **Story Analysis** button by clicking on the name to expand the options. See Story Analysis for more information on how to do this.

4.3 Finding Spiky Functions

With one click you can find functions that spike occasionally.

Setup

- Record at least **1000x per frame**.
- Record a **Callstack** profile.

Procedure

Click the **Rare** button within the **Spike Detection** group on the **Analysis** ribbon bar tab.

Tip: Click on the **Rare** label to expand the options and increase the **Max Spikes**. Then click on the **Rare** button again to reanalyze the profile.

4.4 Identifying the Cause of a Slow Frame

With enough samples per frame, the exact cause of a slow frame can be determined with one button click.

Setup

- Record at least **1000x per frame**.
- Record a **Callstack** profile.
- Record a profile long enough to have a slow frame.

Procedure

1. Optional: In the **Functions** tab, select the core you want to examine before clicking **Bad Frame**.
2. Click the **Bad Frame** button within the **Spike Detection** group on the **Analysis** ribbon bar tab.

Tip: Additional options are available by expanding the **Bad Frame** options.

- Increase the number of bad frames to detect.
- Enable **Use Selected Frames**, select the frames you are concerned about (see Mouse Mode), and click **Bad Frame**. Only functions that perform worse on those frames compared with all other frames will be selected.

4.5 Find all Sampled Functions at a Particular Time

You might want to know what functions were sampled at a particular time. For example, maybe a CPU performance counter indicates poor IPC at a particular time.

Setup

- Record a **Callstack** profile.
- A long profile is not necessary since you will be looking at a particular moment in time.

Procedure

1. Optional: In the **Functions** tab, deselect all functions. This will help reduce clutter in the **Sample Graph**.
2. In the **Sample Graph** tab, find a time that is interesting.
3. Right click on the **Sample Graph** and choose **This Core at This Time** > **Find All Sampled Functions**.

4.6 Measure the Time of a Sample Run in the Sample Graph

To measure a sample run in the **Sample Graph**, change the mouse mode to **Measure Time** in the **Sample Graph** toolbar (far left side).

Setup

- Change the mouse mode to **Measure Time** in the **Sample Graph** toolbar (see Mouse Mode).

Procedure

In the **Sample Graph**, left click at the start of the sample and drag the mouse to the end of the sample.

Remember: The mouse behavior changes in different mouse modes, with different shortcuts available to accomplish tasks.

- In the **Measure Time** mouse mode, you can scroll the window with the mouse by holding the Ctrl key while left clicking and dragging.
- In the **Scroll Graph** mouse mode, you can measure time if you hold the Ctrl key while left clicking and dragging.
- To make your selection stay and select multiple areas, use the **Select Time** mouse mode. Be aware that this will reevaluate the numbers in the **Functions** and **Call Tree** tabs based on the selected regions.

The measured time will be shown at the top of the selected area (if **Load per Frame** is hidden, then the time will be shown at the bottom of the selected area). When you release the left mouse button, the measured area will disappear.

5 Ribbon Bar

This section details the various controls and settings available in the ribbon bar menu.

To hide the ribbon, click the up arrow at the top right. Once hidden, the ribbon bar will show only when the mouse cursor rolls over the ribbon bar tabs. To expand the ribbon so that it stays shown, click the down arrow at the top right.

5.1 Home Tab

The **Home** tab contains the primary profiler operations for connecting to the dev kit and manipulating the profiles.

5.1.1 Connection

In order to connect to a dev kit, you must first press the **Sync** button. Pressing this button starts the following three step synchronization process:

1. Selection of a dev kit. If only one dev kit is available that kit will be connected to automatically. If more than one kit is available a window will appear with a drop down with the list of available dev kits.
2. You will be prompted to locate the executable that you wish to profile. This is used to cross-reference function addresses with function names.
3. The Dev Kit is pinged and the allocated buffer size is sent to the PC.

Once the sync process has completed, the **Start** profile button and the **Unsync** button will become active.

Press the **Unsync** button if you want to profile a different executable or if you want to connect to a different dev kit.

Current connection information is shown in the **Status Bar** at the bottom of the application.

The Nintendo 3DS CPU Profiler also comes with a **Reload** button. The **Reload** button becomes active only after the sync process has completed. Press the **Reload** button if you want to reload the game executable (for example, if you changed some code, recompiled, and want to profile the new build).

Finally, when using an IS-SNAKE-BOX there will be a button labeled **Forced Compat**, short for *Forced CTR Compatibility Mode*. The button performs the same operation as selecting **Force CTR Compatibility Mode** in the **IS-SNAKE-DEBUGGER** software. Since extended applications operate at different speeds on CTR systems compared to SNAKE systems, by using *Forced CTR Compatibility Mode*, you can check on IS-SNAKE-BOX how fast an extended application will run on a CTR system. In other words, you can use a single IS-SNAKE-BOX to check how an application would run on both the CTR and the SNAKE.

5.1.2 Buffer Size

This spinner allows for a selection between 1MB and 16MB. Above the selection of the buffer size is a calculated amount of time it will take to fill the buffer. This is the maximum profiling time based on sampling rate and buffer size. After selecting a new buffer size, it is required that you use the **Reload** button to restart the profiler if the profiler is currently running.

In some circumstances, the **Buffer Size** spinner only becomes an available setting after a **Sync** has been performed. This means that the first time the profiler connects with an application it will use the default buffer size of 1MB. If the value has been changed before, that change will be saved in the settings and the saved buffer size will be used.

Note: Buffer memory is allocated before the game runs, thus leaving less memory for the game. Consider using the Config tool to change the memory mode to repartition more memory for the game. Check Requirements for more information on the profiler's memory requirements.

5.1.3 File

The **Open** button allows you to open a previously saved profile.

The **Save** button saves the current profile to disc.

The **Script** button loads and runs scripts that can automate profiling.

5.1.4 Help

The **Manual** button will launch this document (Nintendo 3DS CPU Profiler Manual). The profiler expects the manual to be one directory up from the profiler executable. In the case that the profiler can't locate the manual, the error message will display the manual filename and directory that it expects.

5.1.5 Units

These buttons control the units used to represent the amount of time spent in each function (for sampled profiles only). The selected units will be reflected in any tab that shows profile data, such as the **Functions** tab, the **Call Tree** tab, or in the tool tips of the **Sample Graph** tab. The various units allow several ways to analyze and compare functions.

Percent

This is the percent of time the function ran compared with all other functions during the selected time period. For example, if a function was sampled 100 times out of 400 total samples, then 25% of the execution time was spent in this function. **Percent** is useful when comparing relative function times. Enable the **Error** button in order to see the margin of error on the values.

Time

This is an approximation of the time spent in each function (the approximation is calculated with statistics and is not directly measured using a timer). If one or more frames are selected or a period of time is selected, then the time is an estimate over that selection. If frames or times are not selected, then the time is an estimate over the entire duration of the profile.

Use **Time** to understand how long a function actually executed, which can be useful if you want to stay within a particular time budget or frame rate. For example, you can use this feature to determine if a particular module or system is going over its budgeted time (such as 2ms each frame for the AI to update). Please be aware that **Time** will be more accurate if you use a very low sampling rate over a very long period of time. As profiler overhead increases, **Time** will become less accurate, so a low sampling rate is critical if you want to use this metric. Also, be aware that selecting individual frames will decrease the precision/accuracy as the margin of error will increase. Enable the **Error** button to see the margin of error (note that this error calculation does not take into account profiler overhead, just statistical variation).

Avg

This is an approximation of the time spent in each function divided by the number of frames. If no frames exist for a given core, then this is the same as the **Time** unit (nothing is averaged). If the core has frames, but none are selected, then time is divided by the total number of recorded frames on that core. If the core has selected frames, then time represents the time on those selected frames divided by the number of currently selected frames. Enable the **Error** button to see the margin of error.

Note: With **Avg** selected, the sorting in the **Total** and **Self** columns in the **Functions** tab may appear wrong. This happens because the functions on different cores are divided by a different number of frames, yet the sorting is based on importance (absolute time). However, sorting is correct for all functions on a given core (sorting is correct if filtered by core).

Samples

This provides the actual number of samples taken in a function. This is the basic unit that the above units are calculated from. Enable the **Error** button to see the margin of error.

Error

This button is independent of the other **Units** buttons. It controls whether or not the margin of error is displayed next to the **Percent**, **Time**, **Avg**, or **Samples** values. The margin of error is useful to determine the precision of the results. For example, if a single frame is selected, the margin of error tends to be very large. As a result, the **Percent** or **Time** values may look precise (for example, 8.2%), but the margin of error can tell you the exact precision (for example, ±5.8%), which is important when comparing values or assessing performance. The **Error** button's state is saved in the profiler's settings so that it will remember the last selected state when you relaunch the profiler.

5.2 Analysis Tab

The **Analysis** tab has two groups of tools to help you quickly analyze a profiler: **Story Analysis** and **Spike Detection**.

5.2.1 Story Analysis

The **Story Analysis** tools help select and order functions to tell an abstract story about what happens over the course of a frame.

To use these tools, you must have taken a **Callstack** profile, you should zoom into a single frame within the **Sample Graph**, and it helps to only look at a particular core (select a single core in the **Functions** tab toolbar).

Story Analysis Options

These options help refine the selected functions.

Exclude Parents: By default, the parents of interesting functions are also selected. This toggle will exclude these parents from the results. This option has no effect on **Top Story Analysis**.

Exclude Irregular: By default, all functions can be included as part of the story. This toggle will exclude functions that do not execute on a regular basis.

Match Heartbeat: By default, all functions can be included as part of the story. This toggle will exclude functions that do not appear at an interval similar to the frame rate.

5.2.1.1 Brief Button

This button selects functions that tell a brief story about what happens over the course of a frame. It is more effective if you first select a single core to look at (see Core Filtering and Column Heading) before hitting this button. Zoom into a single frame in the **Sample Graph** to see the story.

Brief Button Options

The settings of **Max** and **Min** in the button options control which functions are selected. If you change these settings, they will not be saved permanently. However, there are three preset setting groups that you can try: **Short**, **Long**, and **Detailed**. These presets offer varying levels of detail in the generated story, going from most high-level (selects minimal number of key functions) to extremely detailed (selects many functions to show extreme, yet significant, detail).

5.2.1.2 Top Button

This button is different from the previous three (**Brief**, **Full**, and **Fine**) and implements a slightly different algorithm. This button will select the top functions, exclude particular ones based on settings, then order them chronologically.

Top Button Options

The settings in the **Top** button options will help control what gets selected. The **Max Results** setting is very useful for controlling the number of functions that get selected (however the final results may be lower than this number based on other options that exclude functions). If the results contain many similar functions, you can eliminate similar ones with either the **Exclude Similar Children** or **Exclude Similar Parents** toggle buttons.

5.2.2 Spike Detection

The **Spike Detection** tools help quickly find potentially troublesome functions that you should further investigate.

Spike Detection Options

These options help refine the selected functions.

Max Results: This determines how many functions are selected when **Rare**, **Burst**, **Flutter**, or **Bad Frame** are clicked. One strategy is to make this number very low (such as 5 or 10) to hyper focus on the worst spiking functions. Another strategy is to make this number very big (such as 100 or 200) and see if anything interesting shows up.

Exclude Similar Children: If this is enabled, children functions that are too similar to their parent will be discarded from the results. This helps to understand which high-level systems are causing the spikes, since parent functions tend to be named after the high-level systems they represent.

Exclude Similar Parents: If this is enabled, parent functions that are too similar to their children will be discarded from the results. This helps to focus on the low level functions that are directly causing the spikes, since these are more likely the leaf functions where the CPU time is actually spent doing work.

5.2.2.1 Rare Button

The **Rare** button selects functions that rarely spike (which are sampled very infrequently).

Rare Button Options

The default is 10 or fewer spikes over the duration of the profile, but this can be changed with the **Max Spikes** option. The **Min Width** option controls how large a sample run must be (number of consecutive samples) in order to be considered a spike. You can adjust the **Min Width** to make the spike detection more or less sensitive.

5.2.2.2 Burst Button

The **Burst** button selects functions that burst irregularly (that tend not be called every frame).

Burst Button Options

The **Irregularity** option controls the threshold of frame-to-frame irregularity in order for a function to be detected (larger numbers represent more irregularity). The **Min Width** option controls how large a sample run must be (number of consecutive samples) in order to be considered a spike. You can adjust the **Min Width** to make the spike detection more or less sensitive.

5.2.2.3 Wild Button

The **Wild** button selects functions that wildly over their lifetime.

Wild Button Options

The first **Min Change** option controls the minimum number of total milliseconds a function must change over the entire profile in order to be detected (reduce this number to make it more sensitive). The second **Min Change** option controls the minimum percentage a function must change over the entire profile in order to be detected (reduce this number to make it more sensitive).

5.2.2.4 Flutter Button

The **Flutter** button selects functions that are routinely called every frame that have a sudden increase or decrease from one frame to the next.

Flutter Button Options

The **Irregularity** option controls the threshold of frame-to-frame irregularity in order for a function to be detected (larger numbers represent more irregularity). The first **Min Change** option controls the minimum number of total milliseconds a function must change from frame-to-frame in order to be detected (reduce this number to make it more sensitive). The second **Min Change** option controls the minimum percentage a function must change frame-to-frame in order to be detected (reduce this number to make it more sensitive).

5.2.2.5 Swell Button

The **Swell** button selects functions that are routinely called every frame, but that gradually increase or decrease over time with a large total difference over the entire profile. For example, a function might take 5% on the first frame and then incrementally take more time each frame until by the last frame it takes 20%, but the change was gradual with no significant jumps in percentage from one frame to the next.

Swell Button Options

The **Irregularity** option controls the threshold of frame-to-frame irregularity in order for a function to be detected (larger numbers represent more irregularity). The first **Min Change** option controls the minimum number of total milliseconds a function must change over the entire profile in order to be detected (reduce this number to make it more sensitive). The second **Min Change** option controls the minimum percentage a function must change over the entire profile in order to be detected (reduce this number to make it more sensitive).

5.2.2.6 Bad Frame Button

The **Bad Frame** button selects the slowest frames and then selects functions that performed worse on those frames compared with all other frames.

Bad Frame Button Options

The **Max Frames** option controls how many slow frames it will select for the analysis. The **Use Selected Frames** option will use any user selected frames for the analysis, so you can focus on a particular frame or set of frames that you think are problematic (if no frames are selected, the **Bad Frame** analysis acts as if this option is disabled).

5.3 Heartbeats Tab

The **Heartbeats** tab allows profile data to be framed against particular heartbeats per core. In addition, a subset of frames at a given rate can be quickly selected.

5.3.1 Group Selection

Group selection allows particular groups of frames to be selected according to their rate. The following table explains each button's action.

Table 5-1: Group Selection Buttons and Their Actions

Button	Action
60Hz	Select all frames on all cores running at 60 Hz or faster (16.6 ms or less)
30Hz	Select all frames on all cores running between 30 Hz and 60 Hz (16.6 – 33.3 ms)
20Hz	Select all frames on all cores running between 20 Hz and 30 Hz (33 – 50 ms)
15Hz	Select all frames on all cores running between 15 Hz and 20 Hz (50 – 66.6 ms)
12Hz	Select all frames on all cores running between 12 Hz and 15 Hz (66.6 – 83.3 ms)
Under 12Hz	Select all frames on all cores running lower than 12 Hz (greater than 83.3 ms)
Under 30Hz	Select all frames on all cores running lower than 30 Hz (greater than 33.3 ms)
Invert	Invert frame selection (selected frames are deselected and unselected frames are selected)
Deselect All	All frames are deselected

Selection of frames can be viewed in the **Sample Graph** tab. Note that all of the buttons can be toggled, except for **Invert** and **Deselect All**. If you toggle a particular button on, the corresponding frames that match the rate will be selected (highlighted in orange). Toggling a button off will deselect frames that match the rate. The toggling action allows you to select or deselect several ranges independently, for example allowing you to select frame rate ranges. You can manually override these group selections by directly selecting or deselecting individual frames within the **Sample Graph**.

When frames are selected or deselected, all displayed values (for example percentages in the **Functions** tab) are recalculated to only include the selected frames. If no frames are selected, the entire data set is reflected in the values.

5.3.2 Heartbeat Cores

For each core that was recorded during a profile, a numbered **Core** group will appear in the **Heartbeats** tab. Within that **Core** group, you can select heartbeat framing for that core along with some toggles to make that framing apply to all cores or as the master frame rate.

There are three types of selections that appear in the drop down box:

1. Fixed intervals (**Fixed 60Hz Intervals**, **Fixed 1ms Intervals**).
2. Heartbeats that were recorded directly by your game (for example **MainHeartbeat**).
3. Inferred heartbeats (periodic rhythms that were automatically detected in your game).

Fixed intervals are offered in order to draw fixed time markers within the **Sample Graph** for that core. By design, they can't be used as frame boundaries or to select frames, since the game's profile data would never be in-sync with these fixed intervals and the resulting information would be misleading.

Recorded heartbeats and inferred heartbeats show useful frame markings on the **Sample Graph** that can then be used for further analysis or selection. For example, you can select all **30Hz** frames from the **Group Selection** ribbon bar group. Recorded heartbeats are manually logged in the game code using the functions as described in the Nintendo 3DS CPU Profiler Game API. Inferred heartbeats are automatically detected, based on periodic function behavior. However, if the sampling rate is too low or there are no detectable patterns, then there may not be any inferred heartbeats. Also note that inferred heartbeats will have arbitrary frame boundaries based on the most periodic function detected, so it is unlikely that it will line up with the true beginning of your frame (we recommend that you deliberately record a heartbeat in your code if you want accurate frame boundaries). Lastly, inferred heartbeats are ordered from most represented in the profile to least represented, so pay more attention to inferred heartbeats that appear higher up within the drop down box.

If a recorded heartbeat or inferred heartbeat is selected, then that heartbeat can be used across all cores by checking the **Apply to All Cores** toggle button. Additionally, the currently selected heartbeat can be used as the master frame rate (by checking the **Use as Frame Rate** toggle button), which will cause it to be drawn in the timeline at the bottom of the **Sample Graph**. By default, the recorded heartbeat (frame marker) is used as the master frame rate and drawn in the **Sample Graph** timeline.

5.4 Sampled Profile Tab

There are several settings that can affect a **Sampled Profile**. Please consider the following settings.

As the settings on this tab all relate to taking a profile, this tab is only visible when connected to a dev kit. When the connection is established, this tab will automatically appear in the **Ribbon Bar** and become the selected tab.

5.4.1 Profile Buttons

The **Start** and **Stop** profile buttons are available on the far left side. Once the **Start** button is pressed, it will change to a **Stop** button so that the profile can be manually stopped without moving the mouse cursor.

The number to the right of the **Start** button will time how long you have been taking a profile. It will begin counting when you hit the **Start** button.

The drop down box next to the **Start** button controls when the profile will stop. To manually stop the profile by pressing the **Stop** button, leave the drop down selected to **Manual Stop**. If you want to record for a set period of time, choose a time in the drop down box. You can still **Stop** a profile at any time before the specified amount of time has elapsed.

Profiling will always stop when the profile buffer is full.

Important: The **Start** button on the **Sampled Profile** tab will start a sampled profile (as opposed to an **Instrumented Profile**). If you want to start an **Instrumented Profile**, switch to the **Instrumented Profile** tab and use that **Start** button.

5.4.2 Sampling Strategy and Rate Buttons

This group of controls will determine the event that triggers a sample to be taken and the rate at which the event should trigger. In general, there are two types of events: **Sample by Time** and **Sample by Performance Counter**.

Sample by Time uses a timer (jittered for good sampling) to determine when to sample. **Sample by Performance Counter** samples every *n* counter events as specified by the **Sampling Rate** drop down box.

Sampling Strategy Drop Down Box

This control determines the event that triggers a sample to be taken. The following options are available for sampling strategies (where *n* is defined in the **Sampling Rate Drop Down Box**):

Sample by Time

The default sampling option. Samples are taken roughly *n* times per frame 60 Hz frame.

Sample by Miss ICache

Samples are taken every *n* times that an instruction cache miss occurs.

Sample by Miss DCache Read

Samples are taken every *n* times that a data cache read misses.

Sample by Miss DCache Write

Samples are taken every *n* times that a data cache write misses.

Sample by Branch Misprediction

Samples are taken every *n* times a branch misprediction occurs.

Sample by Cycles Stalled Instruction

Samples are taken every *n* cycles that were stalled due to waiting for an instruction to be delivered from memory.

Sample by Cycles Stalled D-Hazard

Samples are taken every *n* cycles that were stalled due to a data hazard (waiting for data to be delivered from memory or waiting for data to be computed).

Sample by Cycles Stalled LSU Full

Samples are taken every *n* cycles that were stalled due to the load store unit being full.

Sampling Rate Drop Down Box

The profiler has several sample rates to choose from, based on the **Sampling Strategy** selected. If **Sample by Time** is selected, the rates reflect the number of times to sample during a 60 Hz frame. For example, a rate of **500x per frame** will result in 500 samples per 60 Hz frame. In the case that the game runs at 30 frames per second, a rate of 500x will effectively sample the game at double the stated rate, or 1000 times per frame. A game that runs at 20 frames per second will effectively be sampled at triple the stated rate, or 1500 times per frame. If one of the **Sample by Performance Counter** choices are selected, then the rate will be in terms of that performance counter.

When sampling by time, as the sampling rate is changed, the estimated profiling time is displayed. As sampling by performance counter does not use a timer no estimate of the profiling time can be given.

5.4.3 Data per Sample

This group of controls determines what is recorded with each sample.

The number of bytes per sample will be shown on the top left side of the **Data per Sample** group. This number is based on all of the **Data per Sample** settings selected.

5.4.3.1 Performance Counters Drop Down

The **Performance Counters** drop down box allows you to choose which performance counters will be recorded with each sample.

Please see Performance Counter Groups for more information on what performance counters are available and what kind of data they record.

5.4.3.2 Callstack Recording Options

There are three choices for what kind of function data should be recorded with each sample:

Leaf Only: When a sample event is triggered only the function that is currently executing is recorded. As very little data is recorded, this option will generally allow for much longer profile times. Recording only the leaf function has the following implications:

- Good: Minimal amount of overhead.
- Good: Longer recording times as minimal data is recorded.
- Bad: No in-depth view of what was going on in the application.

Inferred: This option starts disabled. In order for it to become available, a **Callstack** profile must first be taken. When a sample event is triggered only the function that is currently executing is recorded along with minimal other information, so longer profile times can be achieved. When the data is then viewed on the PC, previous **Callstack** profiles are used to infer the most likely callstack for that sample event. The goal for **Inferred** is to blend the benefits of **Leaf Only** and **Callstack** profiling:

- Good: Get the Good aspects of both the **Leaf Only** and **Callstack** recording options.
- Bad: Callstacks are inferred, so some of the inferences can lead to factually incorrect callstacks being reported. This must be kept in mind when viewing the results.

Callstack: When a sample event is triggered the entire callstack is recorded. This option records a variable amount of memory, depending on the depths of the callstacks recorded. Recording the callstack has the following implications:

- Good: The **Total** time a function has executed for (the time within the function and all functions it calls) will be shown in the **Functions** tab in addition to its **Self** time.
- Good: A call tree will be constructed from the individual profiles and displayed in the **Call Tree** tab. The call tree shows the multiple places where each function was called from and the amount of time spent in each function of the tree.
- Bad: The buffer is filled much more quickly, resulting in shorter possible profile times.
- Bad: There is much more overhead from the profiler, which hurts the frame rate and the accuracy of the profile (since cache is negatively affected by more profiler work).

Note: While only a single **Callstack** profile is required to take an **Inferred** profile, it is recommended that you take many **Callstack** profiles first. Having more **Callstack** profiles will improve the system used to infer callstacks. Try to take at least 5 **Callstack** profiles before taking an **Inferred** profile. The Scripting system can be an easy way to automatically accomplish this.

5.4.4 Cores

The buttons in this section turn on/off profiling for each core. Sometimes it is advantageous to only profile on a single core since it will allow significantly longer profile times and reduce profiler overhead on the other cores. There must be at least one core selected.

5.4.5 Thread Selection

This control lets you select which thread to profile. By default, the **Active** thread is profiled, meaning that at the time of each sample, the current thread's function or callstack is recorded. If you want to profile only one specific thread, then click on the **Refresh Threads** button once the game is running. The profiler will ping the game and get a list of active threads. Wait a moment for the game to respond with the thread list and click the thread selection drop down box to choose the thread to profile.

When profiling with **Active** threads, a portion of time will be attributed to *System Idle Thread* where the profiler is incapable of determining the thread and function that was last active. By restricting the profile to a single thread, you can observe what function was last active and no time will be attributed to *System Idle Thread*.

5.5 Instrumented Profile Tab

This tab is used to take an **Instrumented Profile**. There are three choices to make before taking an instrumented profile.

1. Choose a function to profile (optional if **Marked** is selected).
2. Choose a performance counter group to record.

3. Choose to have **Marked** code blocks recorded (optional if a function is chosen).

As the settings on this tab all relate to taking a profile, this tab is only visible when connected to a dev kit. When the connection is established, this tab will automatically appear in the **Ribbon Bar** and become selectable.

5.5.1 Allowing for Instrumentation

Certain conditions must be to allow for an instrumented profile to be taken.

To instrument with the Nintendo 3DS CPU Profiler it is required that the profiler library must be linked to the application and at least one profiler API be called. We recommend using `nnprofRecordTopMainLoop` as this also provides a mechanism to get more precise frame rate details. Please see Frame Marking on Nintendo 3DS and Nintendo 3DS CPU Profiler Game API for more information.

5.5.2 Profile Buttons

The **Start** and **Stop** profile buttons are available on the far left side. Once the **Start** button is pressed, it will change to a **Stop** button so that the profile can be manually stopped without moving the mouse cursor.

The number to the right of the **Start** button will time how long you have been taking a profile. It will begin counting when you hit the **Start** button.

The drop down box next to the **Start** button controls when the profile will stop. To manually stop the profile by pressing the **Stop** button, leave the drop down selected to **Manual Stop**. If you want to record for a set period of time, choose a time in the drop down box. You can still **Stop** a profile at any time before the specified amount of time has elapsed.

Profiling will always stop when the profile buffer is full.

Important: The **Start** button on the **Instrumented Profile** tab will start an instrumented profile (as opposed to a **Sampled Profile**). If you want to start an **Sampled Profile**, switch to the **Sampled Profile** tab and use that **Start** button.

5.5.3 Choose a Function to Instrument

To choose a function to instrument, follow these steps:

1. Take a **Sampled Profile** so that the **Functions** tab is populated.
2. Click the **Add** button so that the button is highlighted.
3. Click on a function in the **Functions** tab, **Last** tab, **Call Tree** tab, or **Code Coverage** tab that you want to profile.
   The function name should appear in the **Instrumented Profile** tab. When using the **Code Coverage** tab, it helps to use the filtering box at the bottom to find a particular function.

To choose a different function, click the **Add** button again and click on a different function in either the **Functions** tab or the **Code Coverage** tab. To remove your function from being selected, click on the red **X** button next to the function name in the **Instrumented Profile** tab.

5.5.3.1 Limitations on Selecting Functions to Instrument

Not all functions are selectable for use with an **Instrumented Profile**. This is due to some of the limitations of how the instrumentation in the profiler works. In order to instrument your code, the profiler will rewrite small portions of your code. The following restrictions are in place to ensure that this overwrite will work correctly.

- Functions must be ARM code. The profiler enforces this restriction when you attempt to **Add** a function.
- The function cannot use arguments that are passed via the stack. This comes from the instruction overwriting that takes place for instrumentation to occur correctly. The profiler attempts to enforce this restriction, though it cannot be fully enforced, so please be mindful of this limitation.

- The first instruction cannot be a branch. As many branches are based on known locations in code, instrumentation of branch code will result in incorrect behavior. The profiler enforces this restriction when you attempt to **Add** a function.
- The first instruction cannot be PC-relative as the first instruction will be overwritten for instrumentation. The profiler enforces this restriction when you attempt to **Add** a function.
- No code may branch to the first instruction. As the first instruction will be overwritten for instrumentation, branching to this location will likely lead to incorrect behavior. This restriction is currently not enforced, so please be mindful of this limitation.

Note: The validity check for function selection errs on the side of caution to avoid errors appearing in the game or profiler. It is possible that a function that should be allowed for instrumentation is marked as invalid during this check, though it should only happen rarely.

5.5.4 Choose a Performance Counter Group

You can record performance counters along with an instrumented function or along with code blocks. Simply select your desired group from the **Performance Counter Drop Down Box**.

Recording performance counter data will use up buffer space slightly faster and cause additional profiler overhead.

Please see Performance Counter Groups for more information on what performance counters are available and what kind of data they record.

5.5.5 Recording Marked Code Blocks

To turn on recording of code blocks, select the **Marked** button. Note that code blocks must be manually instrumented in the game's source code using the Nintendo 3DS CPU Profiler Game API. This button simply turns on recording of them during a profile.

Caution: Recording code blocks can hurt the recorded profile results.

5.6 Directories Tab

The **Directories** tab allows you to specify a variety of different important directories that the profiler will use to locate files.

Each of the directories can be edited in the following manner:

1. **Directly edit the text box**: You can type the directory in the text box. If the directory does not exist, the background of the text box will turn orange. Once a valid directory is entered, the background will return to normal. If an invalid directory is in the text box and focus leaves the text box, the directory will be replaced with the last known valid directory.
2. **Browse for the directory**: You can click the browse directory button to the right of the text box to find the desired directory.

Note: You can reset the directory back to default by directly editing the text box and clearing the contents.

5.6.1 Code Directory

The **Code Dir** is used to specify the folder which the **Assembly Tab** should use as an attempt to find your source code. This is useful if looking at a profile that was taken on someone else's machine which does not have a matching location for their code folder.

To find the source file, the profiler uses the `addr2line` tool from the Cygwin package `binutils` to turn specific addresses into filenames and line numbers. When a **Code Dir** is specified, the **Assembly Tab** searches for the code file it will start by looking for the base file name in the folder specified in **Code Dir**. If the file is not found it will continually add a directory from the base filename until it finds the file or runs out of directories to add.

If **Code Dir** is not specified, or if a search of the **Code Dir** came up empty, the **Assembly Tab** will default to trying to load in the file as returned by `addr2line`.

Code Dir will default to an empty box.

5.7 Quick Ribbon Bar

The Quick Ribbon Bar is located in the top-right corner of the profiler. It provides a quick way to perform the following actions, even when the Ribbon Bar is minimized.

- **Sync** to a dev kit.
- **Unsync** from a dev kit.
- **Start** a **Sampled Profile**.
- **Stop** taking a profile.
- **Save** the current profile to disc.

6 Functions Tab

The **Functions** tab displays all of the functions that were sampled during a profile.

6.1 Sorting Functions

Functions can be sorted by **Total**, **Self**, **Diff Total**, **Diff Self**, **Core**, **Thread**, **Module**, **Opcodes**, **Seen In Calls**, **Seen Out Calls**, or **Name** by clicking on each respective heading. Please note that the **Total** column will only be shown if a **Callstack** profile was taken (see Callstack Recording Options). The headings **Diff Total** and **Diff Self** will only appear when toggled on by clicking the **Diff** button. The headings **Thread** and **Module** will only appear when toggled on by clicking the **Thread** and **Module** buttons, respectively. The headings **Opcodes**, **Seen In Calls**, and **Seen Out Calls** will only appear when toggled on by clicking the **Stats** button.

By default, the functions are sorted by **Self**, which is the amount of time spent in each function (not including any sub-functions). Click on the **Total** heading to sort by **Total**, which is the amount of time spent in each function and every function it calls.

6.2 Functions Tab Toolbar

The following controls are in the **Functions** tab toolbar.

6.2.1 Core Filtering and Column Heading

If a profile contains multiple cores, you can click on the **Cores** drop-down in the toolbar to show a list allowing for selection between **All Cores** and each individual core. Only cores that were profiled will be available in the **Cores** drop-down. The default selection is **All Cores** and each new profile taken or loaded will reset the **Cores** drop-down selection to **All Cores**.

When a particular core is selected, such as **Core 1**, the **Core** column will be hidden since all functions shown belong to the same core. This can be helpful when focusing on a particular core and can help optimize horizontal space in the tab.

6.2.2 Threading Filtering and Thread Column Heading

By default, the **Thread** column is hidden (to conserve horizontal space). However, if you want to see which thread a function belongs to, click the **Thread** button in the toolbar. Click the button again to hide the column.

If you want to only view functions belonging to a particular thread, click the **Thread** drop-down button and select a particular thread. When an individual thread is selected, the **Thread** column will be hidden since all shown functions belong to the same thread. The default selection is **All Threads** and each new profile taken or loaded will reset the **Thread** drop-down selection to **All Threads**.

By default, threads are named with a 32-bit identifier (such as 0x1052D298). If the profiler is aware of a name for the thread, that name will be displayed in the drop-down. However, the **Thread** column will always only use the 32-bit identifier.

6.2.3 System Classification Filter

If you want to only view functions belonging to a particular system (like Graphics, Physics, Animation, etc.), click the **System Classification** drop-down button and select a particular system. The default selection is **All Systems** and each new profile taken or loaded will reset the **System Classification** drop-down selection to **All Systems**.

6.2.4 Diff Total and Diff Self Column Headings

By default, the **Diff Total** and **Diff Self** columns are hidden (to conserve horizontal space), but can be enabled by clicking on the **Diff** button in the **Functions** toolbar. Click the button again to hide the columns.

The **Diff Total** and **Diff Self** columns compare the current profile with the last loaded profile, showing a percent difference for each function. For example, if a particular function's **Self** percentage is 10% in the last loaded profile and 12% in the current profile, then a value of +2% will be shown in the **Diff Self** column. Likewise, if a particular function's **Self** percentage is 10% in the last loaded profile and 8% in the current profile, then a value of -2% will be shown in the **Diff Self** column.

To see how your game's performance has changed over time, you can first load a profile taken a while ago (i.e. yesterday, a week ago, a month ago), and then take a current profile. Within the **Diff Total** and **Diff Self** columns it will show how much each function has increased or decreased as a percentage. However, be aware that if a particular function was optimized to take less time (and its percentage decreased), percentage-wise it will cause all other functions to increase (even if the absolute amount of time they took was the same between both profiles). To sort all functions by the absolute value of the percentage difference, click on either the **Diff Total** or **Diff Self** column headers. The sorting is based on absolute value in order to prioritize functions that had the largest percentage change, whether that change was positive or negative.

Note: The **Diff Total** column is only available on callstack profiles.

6.2.5 Module Column Heading

By default, the **Module** column is hidden (to conserve horizontal space), but can be enabled by clicking on the **Module** button in the **Functions** toolbar. Click the button again to hide the column.

The **Module** column contains the name of the module that each function came from. To sort all functions by the module they came from (alphabetical sort), click on the **Module** column header.

Note: The **Module** column is only useful if there were relocatable modules used in the application.

6.2.6 Opcodes Column Heading

By default, the **Opcodes** column is hidden (to conserve horizontal space), but can be enabled by clicking on the **Stats** button in the **Functions** toolbar. Click the button again to hide the column.

This column shows how many assembly opcodes are in each function (the size of the function). To sort all functions by the number of assembly opcodes (most to least), click on the **Opcodes** column header.

6.2.7 Seen In Calls and Seen Out Calls Column Headings

By default, the **Seen In Calls** and **Seen Out Calls** columns are hidden (to conserve horizontal space), but can be enabled by clicking on the **Stats** button in the **Functions** toolbar. Click the button again to hide the column.

This column shows the number of in or out calls seen during the profile compared with the number of possible in or out calls detected through static and runtime analysis. For example, if a given function was seen to be called from 6 different functions during profiling, yet through analysis it was determined that it could have been called from 30 different functions, then the value "6/30" would appear in the **Seen In Calls** column for that function. To sort all functions by the number of possible in or out calls (the denominator of the reported value), click on the **Seen In Calls** or **Seen Out Calls** column header. If you are interested in viewing the function names for the seen in and out calls, select the function and click on the **Call Tree** tab to further inspect the call flow. If you are interested in viewing the function names of all possible in or out calls, then right click on the function name in the **Functions** tab and select **Show Details**.

6.2.8 Deselect All Button

To quickly deselect all selected items, click on the **Deselect All** button.

6.2.9 Select Top Button

To quickly select the top 50 functions, click on the button marked **Select Top** in the **Functions** toolbar.

6.2.10 Args Button

By default, function names don't include their argument list when seen in the **Functions** tab and **Call Tree** tab (in order to make the list more readable and speed up GUI drawing time). If you wish to show the function arguments, click the **Args** button.

6.2.11 Copy Button

The **Copy** button will copy to the clipboard whatever is currently shown using the current sorting and filtering. You can then paste the content into an email, document, or Excel spreadsheet. Because the copied content uses tab delimiters, it will retain the column organization when pasted into an Excel spreadsheet.

The **Copy** button takes into account what is currently selected. If any items are selected at the time the **Copy** button is pressed, then only the selected items will be copied. If nothing is currently selected, everything will be copied.

6.2.12 Selected Only Button

The **Selected Only** button forces the list display to only show items which are currently selected. If this mode is enabled and an item is deselected, it will be removed from the displayed list.

6.3 Filtering Functions

Functions can be filtered using the **Filter** text box at the bottom of the **Functions** tab. This allows you to find a particular function or if you want to look at a class of functions or a certain namespace (such as GX2).

Type any string into the text box and a case-insensitive search is performed for the items that contain the string. Only items matching the search text will be displayed. To negate the results (show everything that does not match the filter), click on the exclamation point button to the right of the text box. If there are no matches for the search, the text box will be highlighted orange.

In order to improve the filtering results, only the function name up to and including the first left parenthesis is considered. This eliminates results where a match occurs in the function arguments, since showing matches to function arguments is typically not desirable and results in unwanted matches.

The following are some helpful filters you can use:

Table 6-1: Helpful Simple Filters

Filter Text	! Btn	Description
`(`	N	Show all C++ functions.
`(`	Y	Show all C functions.
`;`	N	Show all functions that share code with other functions.

Regular expressions are also supported (using .NET RegEx, which is Perl 5 compatible). To have the filter interpreted as a regular expression, click on the **RegEx** button to the right of the text box. Unlike basic filters, regular expressions are case-sensitive.

The following are some helpful regular expressions:

Table 6-2: Helpful Regular Expression Filters

Filter Text	! Btn	Description
`^GX2`	N	Show all functions starting with "GX2".
`^GX2`	Y	Show all functions not starting with "GX2".
`^GX2|^__`	Y	Remove multiple functions (not starting with "GX2" or "__").
`(?i)^cos|^sin|^tan`	N	Show all trigonometric functions (case-insensitive for all).
`(?i)a(?-i)nim`	N	Show functions with either "anim" or "Anim".
`(?i)math.*vec`	N	Show functions with "math" and "vec" in that order (case-insensitive).
`(?<!trans)pose`	N	Show functions with "pose" which are not preceded by "trans".
`skin(?!g)`	N	Show functions with "skin" which are not followed by "g".

Note: All filters in the **Functions**, **Call Tree**, and **Code Coverage** tabs act similarly on functions and share saved filters.

6.3.1 Quick Regular Expression Primer

Regular expressions are a concise method for matching strings of text. The following are some rules that you might find helpful when specifying a regular expression:

- There should be no spaces in a regular expression (unless you're searching for a space).
- A vertical bar indicates Boolean "or". For example, `cos|sin` can match "cos" or "sin".
- (?i) indicates case-insensitive. Using it once causes everything afterward to be affected. For example, `(?i)cos|sin` matches "cos" or "sin" with both being case-insensitive.
- A caret indicates the beginning of a function only. For example, `^GX2` matches functions starting with "GX2".
- A period can match any character. An asterisk specifies zero or more of the preceding element. Together they can specify any character zero or more times. For example, `math.*vec` matches "math" and "vec" with zero or more characters between them.

6.3.2 Saved Filters

Saved filters allow for using the same filters between profiling sessions, with any saved filters being loaded when the profiler is opened. By default, the **Filter Drop Down** box is loaded with example filters that you can customize. If you want to save your own filter in the drop down box, press the **Save** ("**+**") button after typing the filter (the **Negate** and **RegEx** button states will also be saved with the filter text). To delete a filter from the drop down box, select it first, then press the **Delete** ("**-**") button.

Note: The saved filters in the **Functions** tab will also be available in the **Call Tree** and **Code Coverage** tabs. However, please note that the **Filter Drop Down** box in the **Threads** tab and **Counters** tab are not shared since they are used to search for thread and counter names, not function names.

6.4 Selecting Functions

Click on a function to select it and click again to unselect it. Up to 500 functions can be selected (each will be highlighted in a different color). Selected functions will be graphed in the **Sample Graph** and highlighted in the **Call Tree** (the **Call Tree** is only visible if the profile was taken with **Callstack** profile data - see Callstack Recording Options).

To quickly select the top 50 functions, click on the button marked **Select Top** in the **Functions** toolbar.

To quickly deselect all functions, click on the **Deselect All** button in the **Functions** toolbar.

6.4.1 Alternative Selection Methods

Multiple items can be selected or deselected with Shift+Click (select/deselect an item without holding the Shift key, then hold the Shift key and click on a second item). The range between the two clicks will be either selected or deselected.

To quickly select only a single item and deselect everything else, Alt+Click can be used. This deselects all and then selects the clicked upon item.

6.5 Right-Click Context Menu

A right-click in the tab will bring up a context menu containing a variety of options.

6.5.1 Copy Function Name to Clipboard

This will copy the full function name with arguments to the clipboard.

6.5.2 Deselect All Other Functions

The Deselect All Other Functions option will force all functions other than the one click on to become deselected. This will not result in an unselected function becoming selected.

6.5.3 Show Assembly

The **Show Assembly** option will move focus to the Assembly Tab, showing the assembly of the clicked function.

6.5.4 Show Details

The **Show Details** option will open a new Function Details Window, showing the details of the clicked function.

6.6 Units

To change the units displayed before each item, use the **Unit** buttons in the **Home** tab of the ribbon bar (see Units).

6.7 Resizing the Window

All tabs in the left tab group, including the **Functions**, **Threads**, **Instrumented**, and **Counters** tabs, can be stretched horizontally by dragging the right edge with the mouse cursor. All of these tabs share the same display area, so resizing the width of one tab will result in all tabs in the tab group being resized.

6.8 Reading Obscured Function Names

If a function name is very long, it might be obscured by the right edge of **Functions** tab. You can either resize the **Functions** tab or you can hover over the function and a tooltip containing the full name will be displayed. Note that the tooltip will contain the full function signature including function arguments, regardless of the state of the **Args** button.

6.9 Multiple Function Names on a Single Line

Some lines will be a concatenation of several function names, each separated by a semi-colon (only visible when the **Args** button is selected or by hovering over the function and viewing the tooltip). For example, you might see the following function name listed on a single line in the **Functions** tab:

```
function_name_1; function_name_2
```

This happens when different functions get compiled into the same exact code. If the profiler samples execution in one of these shared functions, it can't determine which function name was called, so it concatenates all function names together on the same line.

6.10 System Idle Thread

When a profile contains multiple threads, there are times in which no thread is active, so this time is attributed to *System Idle Thread*. In reality, a particular thread is waiting on a system call or such, but the profiler can't determine which thread is waiting.

6.11 Unresolved Functions

When a profile contains a function it cannot resolve to a name, it will label the function as an internal unresolved function. These generally take the form of `filename.ext_internal (unresolved)` where '`filename.ext`' is the name of the file from which function symbols were loaded.

7 Last Tab

The **Last** tab displays all of the functions from the last sampled profile. Not only is this useful for comparison purposes, but it is extremely convenient for selecting new functions to be instrumented, since once you take an **Instrumented Profile**, the normal **Functions** tab is hidden since it no longer contains data. The functions in the **Last** tab cannot be selected, but most of the operations that work in the **Functions** tab also work in this tab, such as all of the sorting and filtering functionality. Note that the units for function data on the **Last** tab are restricted to percentage and samples.

8 Threads Tab

The **Threads** tab displays all of the threads that were sampled during a profile.

8.1 Sorting Threads

Threads can be sorted by **Self**, **Core**, or **Name** by clicking on each respective heading. By default, the threads are sorted by **Core**, and then sorted by **Self** within each core.

8.2 Threads Tab Toolbar

The following controls are in the **Threads** tab toolbar.

8.2.1 Core Filtering and Column Heading

If a profile contains multiple cores, you can click on the **Cores** drop-down in the toolbar to show a list allowing for selection between **All Cores** and each individual core. Only cores that were profiled will be available in the **Cores** drop-down. The default selection is **All Cores** and each new profile taken or loaded will reset the **Cores** drop-down selection to **All Cores**.

When a particular core is selected, such as **Core 1**, the **Core** column will be hidden since all functions shown belong to the same core. This can be helpful when focusing on a particular core and can help optimize horizontal space in the tab.

8.2.2 Deselect All Button

To quickly deselect all selected items, click on the **Deselect All** button.

8.2.3 Top Button

The **Top** button selects the first 45 items in the list, depending on the currently selected sorting mechanism.

8.2.4 Copy Button

The **Copy** button will copy to the clipboard whatever is currently shown using the current sorting and filtering. You can then paste the content into an email, document, or Excel spreadsheet. Because the copied content uses tab delimiters, it will retain the column organization when pasted into an Excel spreadsheet.

The **Copy** button takes into account what is currently selected. If any items are selected at the time the **Copy** button is pressed, then only the selected items will be copied. If nothing is currently selected, everything will be copied.

8.3 Selecting Threads

Click on a thread to select it and click again to unselect it. Up to 45 threads can be selected (each will be highlighted in a different color). Selected threads will be graphed in the **Sample Graph**.

To quickly select the top 45 threads, click on the button marked **Top** in the **Threads** toolbar.

To quickly deselect all threads, click on the **Deselect All** button in the **Threads** toolbar.

8.3.1 Alternative Selection Methods

Multiple items can be selected or deselected with Shift+Click (select/deselect an item without holding the Shift key, then hold the Shift key and click on a second item). The range between the two clicks will be either selected or deselected.

To quickly select only a single item and deselect everything else, Alt+Click can be used. This deselects all and then selects the clicked upon item.

8.4 Filtering Threads

Threads can be filtered using the **Filter** text box at the bottom of the **Threads** tab. This allows you to find a particular thread when a large number of threads are shown. Typically there aren't that many threads, so this is usually unnecessary. It is usually better to filter by core using the **Core** drop down box in the **Threads** toolbar.

Type any string into the text box and a case-insensitive search is performed for the items that contain the string. Only items matching the search text will be displayed. To negate the results (show everything that does not match the filter), click on the exclamation point button to the right of the text box. If there are no matches for the search, the text box will be highlighted orange.

Regular expressions are also supported (using .NET RegEx, which is Perl 5 compatible). To have the filter interpreted as a regular expression, click on the **RegEx** button to the right of the text box. Unlike basic filters, regular expressions are case-sensitive.

Please see the Quick Regular Expression Primer for more information on Regular Expressions.

8.4.1 Saved Filters

Saved filters allow for using the same filters between profiling sessions, with any saved filters being loaded when the profiler is opened. By default, the **Filter Drop Down** box is loaded with example filters that you can customize. If you want to save your own filter in the drop down box, press the **Save** ("**+**") button after typing the filter (the **Negate** and **RegEx** button states will also be saved with the filter text). To delete a filter from the drop down box, select it first, then press the **Delete** ("**-**") button.

Note: The saved filters in the **Threads** tab are saved independently of all other filters.

8.5 Resizing the Window

All tabs in the left tab group, including the **Functions**, **Threads**, **Instrumented**, and **Counters** tabs, can be stretched horizontally by dragging the right edge with the mouse cursor. All of these tabs share the same display area, so resizing the width of one tab will result in all tabs in the tab group being resized.

9 Instrumented Tab

The **Instrumented** tab displays all instrumented functions and all instrumented code blocks that were recorded during a profile. The **Instrumented** tab will only appear if either of these types of data were recorded.

9.1 Instrumented Tab Toolbar

The following controls are in the **Instrumented** tab toolbar.

9.1.1 Deselect All Button

To quickly deselect all selected items, click on the **Deselect All** button.

9.1.2 Expand All Button

The **Expand All** button expands all possible items in the view.

9.1.3 Collapse All Button

The **Collapse All** button collapses all possible items in the view.

9.1.4 Select Similar Button

When the **Select Similar** button is enabled selecting or deselecting an item causes similar items to be selected or deselected.

On the **Instrumeted** tab, the **Select Similar** button will allow for selecting similar values across instrumentation and code blocks. Only expanded blocks are evaluated for **Select Similar**. For example, a single click on one block's call count will result in the call count for all expanded blocks becoming selected (or deselected).

9.1.5 Copy Button

The **Copy** button will copy to the clipboard whatever is currently shown using the current sorting and filtering. You can then paste the content into an email, document, or Excel spreadsheet. Because the copied content uses tab delimiters, it will retain the column organization when pasted into an Excel spreadsheet.

The **Copy** button takes into account what is currently selected. If any items are selected at the time the **Copy** button is pressed, then only the selected items will be copied. If nothing is currently selected, everything will be copied.

9.2 Instrumented Items

Instrumented items can be expanded to view all of the data that was recorded during the profile. The following measurements should appear for each instrumented item.

- **Function Calls** *or* **Blocks**: The number of times the function or code block was entered and exited during the profile. If these are not the same number, the lesser of the two is used (for example, sampling stops while in a recursive function).
- **Max Recursive Depth**: Displays the max recursive depth that was detected during the profile. This measurement only appears if the value is greater than zero.

- **Time per Call** *or* **Time per Block**: This is the average time per call or block that was measured (time = total time / number of calls).
- **Performance Counter Data**: These measurements are based on the performance counters that were selected at the time of the profile. These values are over the duration of the profile. These values might be more useful if they are divided by the number of calls (for values that aren't percentages).

9.3 Selecting Instrumented Items to Graph

Selecting data fields within each instrumented item will cause them to be graphed within the **Sample Graph**. Up to 45 data fields can be selected at a time.

To select similar data fields within all expanded items, enable the **Select Similar** button in the **Instrumented** tab toolbar and click on a single data field.

9.3.1 Alternative Selection Methods

Multiple items can be selected or deselected with Shift+Click (select/deselect an item without holding the Shift key, then hold the Shift key and click on a second item). The range between the two clicks will be either selected or deselected.

To quickly select only a single item and deselect everything else, Alt+Click can be used. This deselects all and then selects the clicked upon item.

9.4 Profiler Overhead

The choice to only allow one function at a time to be instrumented was deliberate, due to profiler overhead considerations. For each recorded enter and exit of an instrumented function, there is typically between 150us to 300us of overhead (with about half of that overhead showing up in the reported **Time** and **Cycles**).

To better understand the overhead that exists within each function and code block measurement, instrumented profiles contain a code block, titled **Zero-Sized Block (Per Block Overhead)**, that can be used to study and graph the overhead. This is an actual code block called from the game that does no work inside it. Use the results to temper your understanding of other instrumented functions and code blocks.

9.5 Resizing the Window

All tabs in the left tab group, including the **Functions**, **Threads**, **Instrumented**, and **Counters** tabs, can be stretched horizontally by dragging the right edge with the mouse cursor. All of these tabs share the same display area, so resizing the width of one tab will result in all tabs in the tab group being resized.

10 Counters Tab

The **Counters** tab displays all of the counters that were sampled during a profile.

10.1 Types of Counters

The profiler currently supports the following general counters. An individual platform may contain more counters.

10.1.1 Performance Counters

For more information on Performance Counters, please see Performance Counter Groups.

10.2 Sorting Counters

Counters can be sorted by **Core**, **Group**, or **Name** by clicking on each respective heading. By default, the counters are sorted by **Core**.

10.3 Counters Tab Toolbar

The following controls are in the **Counters** tab toolbar.

10.3.1 Core Filtering and Column Heading

If a profile contains multiple cores, you can click on the **Cores** drop-down in the toolbar to show a list allowing for selection between **All Cores** and each individual core. Only cores that were profiled will be available in the **Cores** drop-down. The default selection is **All Cores** and each new profile taken or loaded will reset the **Cores** drop-down selection to **All Cores**.

When a particular core is selected, such as **Core 1**, the **Core** column will be hidden since all functions shown belong to the same core. This can be helpful when focusing on a particular core and can help optimize horizontal space in the tab.

10.3.2 Group Filtering and Column Heading

The **Groups** drop-down in the **Counters** tab allows for selection between **All Groups** and any of the groups that were present in the profile (such as **Performance Counters**). Only groups that are in the profile will be listed in the **Groups** drop-down. The default selection is **All Groups** and each new profile taken or loaded will reset the **Groups** drop-down selection to **All Groups**.

When a particular group is selected, for example **Performance Counters**, the **Group** column will be hidden since all groups shown are of the same type.

Specifically for the group **Performance Counters**, the drop-down will contain additional info about which performance counter group was recorded for the profile (see Performance Counter Groups). For example, if a performance counter group named **Example Group** was profiled, the **Groups** drop-down will contain an entry for **Performance Counters (Example Group)**.

10.3.3 Deselect All Button

To quickly deselect all selected items, click on the **Deselect All** button.

10.3.4 Select Similar Button

When the **Select Similar** button is enabled selecting or deselecting an item causes similar items to be selected or deselected.

On the **Counters** tab, the **Select Similar** button will allow for selecting similar counters across cores. For example, if a performance counter group was profiled, a single click on the first counter on core 0 will select (or deselect) the first counter on all profiled cores.

10.3.5 Copy Button

The **Copy** button will copy to the clipboard whatever is currently shown using the current sorting and filtering. You can then paste the content into an email, document, or Excel spreadsheet. Because the copied content uses tab delimiters, it will retain the column organization when pasted into an Excel spreadsheet.

The **Copy** button takes into account what is currently selected. If any items are selected at the time the **Copy** button is pressed, then only the selected items will be copied. If nothing is currently selected, everything will be copied.

10.4 Filtering Counters

Type any string into the text box and a case-insensitive search is performed for the items that contain the string. Only items matching the search text will be displayed. To negate the results (show everything that does not match the filter), click on the exclamation point button to the right of the text box. If there are no matches for the search, the text box will be highlighted orange.

Regular expressions are also supported (using .NET RegEx, which is Perl 5 compatible). To have the filter interpreted as a regular expression, click on the **RegEx** button to the right of the text box. Unlike basic filters, regular expressions are case-sensitive.

Please see the Quick Regular Expression Primer for more information on regular expressions.

10.4.1 Saved Filters

Saved filters allow for using the same filters between profiling sessions, with any saved filters being loaded when the profiler is opened. By default, the **Filter Drop Down** box is loaded with example filters that you can customize. If you want to save your own filter in the drop down box, press the **Save** ("**+**") button after typing the filter (the **Negate** and **RegEx** button states will also be saved with the filter text). To delete a filter from the drop down box, select it first, then press the **Delete** ("**-**") button.

Note: The saved filters in the **Counters** tab are saved independently of all other filters.

10.5 Selecting Counters

Click on a counter to select it and click again to unselect it. Up to 45 counters can be selected (each will be highlighted in a different color). Selected counters will be graphed in the **Sample Graph** tab.

To quickly deselect all functions, click on the **Deselect All** button.

10.5.1 Alternative Selection Methods

Multiple items can be selected or deselected with Shift+Click (select/deselect an item without holding the Shift key, then hold the Shift key and click on a second item). The range between the two clicks will be either selected or deselected.

To quickly select only a single item and deselect everything else, Alt+Click can be used. This deselects all and then selects the clicked upon item.

10.6 Resizing the Window

All tabs in the left tab group, including the **Functions**, **Threads**, **Instrumented**, and **Counters** tabs, can be stretched horizontally by dragging the right edge with the mouse cursor. All of these tabs share the same display area, so resizing the width of one tab will result in all tabs in the tab group being resized.

10.7 Units

To change the units displayed before each item, use the **Unit** buttons in the **Home** tab of the ribbon bar (see Units).

11 Info Tab

The **Info** tab displays general information about the profile.

11.1 Info Tab Toolbar

The following controls are in the **Info** tab toolbar.

11.1.1 Expand All Button

The **Expand All** button expands all possible items in the view.

11.1.2 Collapse All Button

The **Collapse All** button collapses all possible items in the view.

11.1.3 Copy Button

The **Copy** button will copy to the clipboard whatever is currently shown using the current sorting and filtering. You can then paste the content into an email, document, or Excel spreadsheet. Because the copied content uses tab delimiters, it will retain the column organization when pasted into an Excel spreadsheet.

The **Copy** button takes into account what is currently selected. If any items are selected at the time the **Copy** button is pressed, then only the selected items will be copied. If nothing is currently selected, everything will be copied.

11.2 Environment

The **Environment** section has information on the profiler's operating environment at the time of the profile. This includes the following information:

- **User**: The name of the user logged into the computer.
- **Computer**: The name of the computer running the profiler.
- **Time**: The time at which the profile was taken.
- **Game**: The path to the executable file on the hard drive.
- **Captured By**: The name and type of kit used to capture the profile.

11.3 Sampling Settings

The **Sample Settings** section shows the settings that were used to generate the profile. These are listed on a per-core basis.

- The rate at which sampling was requested, either by time or by performance counter event trigger.
- Whether callstack profiling was enabled.
- Whether performance counters were recorded with the profile, and if so, which performance counter group was selected.
- Whether frame markers were seen on the core.

Example Output

```
Core 0: 10x per frame, Callstack Sampling, Has frame markers
```

11.4 Profile Statistics

The **Profile Statistics** section shows general information about the profile. This includes the following:

- **Buffer Used**: The size of the raw data buffer.
- Listed per Core
  - **Samples**: The number of samples seen on the core.
  - **Duration**: The duration of the profile on the core. It is possible for there to be a slight variance between core durations.
  - **Functions seen**: The number of functions seen executing on the core.
  - **Threads seen**: The number of threads seen executing on the core.
  - **Max callstack depth**: The deepest the callstack got at any time while profiling. (Only visible if a callstack profile was taken.)
  - **Call tree node**: The total number of nodes in the **Call Tree** tab. (Only visible if a callstack profile was taken.)

11.5 Module Statistics

The **Module Statistics** section shows the various modules that were seen during the profile. This is split into **Recorded Modules** and **Resolved Modules**.

Recorded Modules are those relocatable modules that the profiler saw loaded and/or unloaded. For each recorded module there should an entry in the resolved modules list.

Resolved Modules are the actual modules that were loaded into the profiler in order to resolve symbol address and names. For static modules there will be an entry in this list, but not in the recorded modules list.

Each module listing may show the following:

- The name of the module.
- The path which the module was loaded from (only for **Resolved Modules**).
- **Base**: The address at which the module was loaded into memory.
- **Size**: The reported size of the module in memory (only for **Recorded Modules**).
- **Functions**: The number of functions loaded from the module (only for **Resolved Modules**).
- **Lifetime**: The times during which the module was loaded into memory in the format "[time loaded, time unloaded]".

11.6 Inferred Heartbeats

The **Inferred Heartbeats** section shows any inferred heartbeats that were found while analyzing the profile data. The following information is provided per inferred heartbeat:

- The core the inferred heartbeat was found on.
- **Period**: The period at which the function was found to execute.
- **Variance**: How close the function was to a regular period.
- **Thread**: The ID of the thread which the function was running on.
- The name of the function.

11.7 Application Information

The **Application Information** provides general data about the running application. The following items are recorded:

- **Average Frame Rate**: The average frame rate seen while profiling. To get a value here, frame marking must be enabled.
- **Build SDK**: The SDK that was used to build the application.
- **Build Type**: The type of build (Debug or Release).

11.8 Profiler Information

The Profiler Statistics section shows information about the profiler runtime that was used to record the data in the profile. While not directly useful most of the time, these values are useful when reporting problems seen with the profiler.

All Nintendo 3DS profiles should be able to see the following:

- **Profiler Header Version**: This is an internal marker used in the profiler which indicates a set of settings.

Recently added to the profiler are the following:

- **Profiler Version**: The version of the profiler used to record the data.
- **Profiler Build SDK**: The SDK used to build the profiler.

11.9 Notes

The **Notes** text box at the bottom of the **Info** tab allows you to store user notes with a profile. These notes are saved along with the profile when it is saved with the **Save** button.

If there are no notes, the **Notes** section will be collapsed with only the header showing. To expand the **Notes** section, either click the up arrow on the right side of the **Notes** header or drag the **Notes** header upward with the mouse.

12 Checkers Tab

The **Checkers** tab contains a list of checkers that have been run against the profile to detect potential problems.

Whenever a new profile is received from the dev kit or a profile is loaded, the checkers will be automatically run against the profile. As checkers complete, any checkers that found a problem show their results in the tab. Checkers that did not find any issues are not displayed.

12.1 Checkers Tab Toolbar

The following controls are on the **Checkers** tab toolbar.

12.1.1 Manage Checkers

Clicking on the **Manage Checkers** button will display all checkers in the **Checkers Tab**, including those disabled and those that did not have any results. This allows disabled checkers to be re-enabled as well as the ability to modify checker parameters for checkers that did not have any results.

When in this management mode, the state of each checker is displayed on the far right side of the checker title. If the checker would normally display a result on a standard run, no special label is shown on the right side. If the checker is Enabled but did not find anything, the text **No Results** is displayed. If the checker is Disabled the text **Checker Disabled** is displayed.

Note: A Disabled checker does not run while in management mode. To see potential results for the checker, it must be Enabled.

12.1.2 Status

To the right of the **Manage Checkers** button is a status area that will display the current status of the running checkers.

There are two different status types that might be displayed:

- `Waiting for ELF processing to finish`: This message indicates that there is still some background processing occurring on the ELF. This is most commonly attributed to the static analysis of the game, looking for static function calls in each function's code.
- `Running checker` *`checker_name`*: This message indicates which checker is currently being run on the profile data.

12.2 Right-Click Context Menu

A right-click in the tab will bring up a context menu containing a variety of options.

The context menu for each checker is generated depending on what kind of values are stored in the checker's results table. Any of the following items may be included in the context menu for an individual checker.

A context menu item will be disabled if the specific item in the checker results does not support the operation.

12.2.1 Select Function

The **Select Function** option results in the clicked function becoming selected in the **Functions** tab.

12.2.2 Show Details

The **Show Details** option will open a new Function Details Window, showing the details of the clicked function.

12.2.3 Show Assembly

The **Show Assembly** option will move focus to the Assembly Tab, showing the assembly of the clicked function.

12.2.4 Load per Frame Analysis

The **Load per Frame Analysis** option will select all relevant functions for this table entry and show the load per frame for the entire group in the **Sample Graph**. This will also cause the **Sample Graph** tab to become visible.

To reset the view, click the Reset button on the Analysis Tab.

12.3 Severity

The severity of each checker can be determined by its icon and its position in the list. The most severe issues appear toward the top of the list.

- Items with the purple chevron are the most severe items and are serious issues.
- Items with the red square are severe items and could indicate serious issues.
- Items with the orange diamond are major issues and should be investigated further.
- Items with the yellow triangle represent minor issues or issues that are not known to be serious.
- Items with the green circle are presented for informational purposes only.

12.4 Checker Results

When a checker finds items to report, an entry is added into the **Checkers** tab. Each entry contains a specific set of items.

12.4.1 Rerun Checker Button

The **Rerun Checker** button forces the checker to rerun with the current settings. This process will remove the existing checker results from the list, obtain new results, add the new results to the list and show you those results.

12.4.2 Copy Text Button

Clicking the **Copy Text** button will copy the **Summary**, **Explanation**, and **Suggestions** to the clipboard.

12.4.3 Copy Table Button

Clicking the **Copy Table** button will copy the **Results Table** to the clipboard. The copied text contains tab-separated values and can be paste directly into spreadsheet applications.

12.4.4 Disable Checker Button

The **Disable Checker** button keeps the checker from running and displaying in the **Checker Tab**. The Enabled/Disabled state of the checker is saved in the settings and will persist between profiler runs.

To re-enable the checker, use the Manage Checkers button in the **Checker Tab Toolbar**.

Note: Enabling or Disabling a checker will result in that checker being automatically rerun.

12.4.5 Summary

All checkers contain a brief **Summary** of the results that were found. If no results were found, this will be stated in the **Summary**.

If a checker takes longer than one second to run, the time will be reported at the bottom of the **Summary**.

12.4.6 Configurable Parameters

When a checker has settings that can be configured, spinner controls will appear beneath the **Summary**. Each spinner starts at the default value and has a maximum and minimum allowed value. Adjusting the parameter can result in better checker results for a specific application. Any changes to these parameters are saved for use in subsequent runs of the profiler. To reset the parameters to the default values (and overwrite any saved values), click the **Reset to Defaults** button.

12.4.7 Reset to Defaults Button

The **Reset to Defaults** button is a quick way to set all of the Checker Configurable Parameters back to their default values.

12.4.8 Results Table

Each checker creates its own **Results Table**. There are no set columns that are guaranteed to appear in a checker.

Columns that commonly appear in checkers include:

- **Self**: The amount of time attributed to this function in the profile results.
- **Function**: The name of the function found.

12.4.8.1 Sorting the Results Table

Each **Results Table** comes pre-sorted using a default sort determined to be the most generally useful for the checker results.

If the default sorting is insufficient, or to get a different view the results, it is possible to change the sorting used in the results. Click on any of the headers in the table to sort the results by that column. When this action is performed an arrow will appear in the column to indicate the sorted column and the sort direction. Clicking on the same column again will result in the sort direction changing.

12.4.9 Explanation

The details in the **Explanation** describe what the checker is looking for and why it is looking for it.

12.4.10 Suggestions

The details in the **Suggestions** provide guidance on how to change your application to correct the issues reported by the checker.

13 Call Tree Tab

This tab shows the discovered call tree based on the profiled sample data. This tab only appears if the profile was taken with **Callstack** data turned on (see Callstack Recording Options).

13.1 Inferred Call Tree

When a **Inferred Callstack** profile is taken, this tab will be renamed from **Call Tree** to **Inferred Call Tree**.

13.2 Call Tree Tab Toolbar

The following controls are on the **Call Tree** tab toolbar.

13.2.1 Prefix Selection

The prefix before each function name is determined by the three **Prefix** toggle buttons in the **Call Tree** toolbar. By default, the **Total** and **Self** prefixes are turned on. To toggle a prefix on and off, click on the prefix button. If all buttons are set to off, just the function name will be shown. The prefix choices are:

Total: This shows the time spent in each function and the functions it calls.

Self: This shows the time spent in each function, not including the functions it calls.

Sub: This shows the time spent in functions that are called by each function.

Note that the functions at each level of the tree are sorted based on the prefix. If **Total**, **Self**, or **Sub** are selected, the sorting is from highest to lowest of the leftmost prefix. If no prefix is selected, the sorting is alphabetical, based on the function name.

13.2.2 Expand All Button

The **Expand All** button expands all possible items in the view.

13.2.3 Collapse All Button

The **Collapse All** button collapses all possible items in the view.

13.2.4 Auto Expanding

By default, the **Call Tree** will not auto-expand, however you can turn on this behavior with the **Auto Expand** drop-down box. If auto expanding is turned on, the tree will expand and/or collapse based on which functions are selected with mouse clicks or highlighted as a result of the **Find** functionality (in either the **Functions** tab or **Call Tree** tab).

The **Auto Expand** drop down box has three choices:

- **Auto Expand Off**: The **Call Tree** will not expand or collapse based on functions being selected/deselected or highlighted by **Find**.
- **Auto Expand Only**: The **Call Tree** will expand (but not collapse) based on functions being selected or highlighted by **Find**.
- **Auto Expand/Collapse**: The **Call Tree** will expand and collapse based on functions being selected/deselected or highlighted by **Find**.

Additionally, you can quickly expand and collapse the entire tree with the **Expand All** and **Collapse All** buttons.

13.2.5 Copy Button

The **Copy** button will copy to the clipboard whatever is currently shown using the current sorting and filtering. You can then paste the content into an email, document, or Excel spreadsheet. Because the copied content uses tab delimiters, it will retain the column organization when pasted into an Excel spreadsheet.

The **Copy** button takes into account what is currently selected. If any items are selected at the time the **Copy** button is pressed, then only the selected items will be copied. If nothing is currently selected, everything will be copied.

13.2.6 Copy All Button

The **Copy All** button will copy to the clipboard whatever is currently shown using the current sorting and filtering. You can then paste the content into an email, document, or Excel spreadsheet. Because the copied content uses tab delimiters, it will retain the column organization when pasted into an Excel spreadsheet.

Unlike the **Copy** button, **Copy All** does not respect selected items. This allows for one to copy the entire visible call tree at any time.

13.3 Finding Functions

Type any string into the text box and a case-insensitive search is performed for the items that contain the string. Only items matching the search text will be displayed. To negate the results (show everything that does not match the filter), click on the exclamation point button to the right of the text box. If there are no matches for the search, the text box will be highlighted orange.

Regular expressions are also supported (using .NET RegEx, which is Perl 5 compatible). To have the filter interpreted as a regular expression, click on the **RegEx** button to the right of the text box. Unlike basic filters, regular expressions are case-sensitive.

Helpful function filters can be found in the Helpful Simple Filters and Helpful Regular Expression Filters tables. Please see the Quick Regular Expression Primer for more information on regular expressions.

Note: The **Find** operation here is different than the **Filter** operations on the **Functions** and **Code Coverage** tabs. In the **Call Tree** view all functions are always visible. Instead, only the found results of the search are highlighted.

13.3.1 Saved Filters

Saved filters allow for using the same filters between profiling sessions, with any saved filters being loaded when the profiler is opened. By default, the **Filter Drop Down** box is loaded with example filters that you can customize. If you want to save your own filter in the drop down box, press the **Save** ("**+**") button after typing the filter (the **Negate** and **RegEx** button states will also be saved with the filter text). To delete a filter from the drop down box, select it first, then press the **Delete** ("**-**") button.

Note: The saved filters in the **Functions** tab will also be available in the **Call Tree** and **Code Coverage** tabs. However, please note that the **Filter Drop Down** box in the **Threads** tab and **Counters** tab are not shared since they are used to search for thread and counter names, not function names.

13.4 Function Selection

Clicking on a function in the **Call Tree** will highlight the function, similarly to how function selection works in the **Functions** tab (up to 45 functions can be highlighted). All instances of the highlighted function will be highlighted in the **Call Tree**. If auto-expanding is on, then the tree will be expanded to show all instances of the selected function.

13.5 Right-Click Context Menu

A right-click in the tab will bring up a context menu containing a variety of options.

13.5.1 Manual Expanding

The call tree works like a typical Windows tree control, where the tree can be expanded or collapsed at any node by clicking the plus or minus sign. Additionally, you can right click on a single node and choose to expand or collapse downward in the tree from that node.

13.5.2 Show Assembly

The **Show Assembly** option will move focus to the Assembly Tab, showing the assembly of the clicked function.

13.5.3 Show Details

The **Show Details** option will open a new Function Details Window, showing the details of the clicked function.

13.6 Partial Call Tree

When a function is selected, it will also appear in the **Partial Call Tree** at the bottom of the **Call Tree Tab**. There are three different views available in this sub-window which allow for differing ways to view the data.

The **Partial Call Tree** window can be resized by dragging the header up and down. It can also be collapsed by clicking on the down arrow on the right side of the header and similarly expanded by clicking on the up arrow.

Forward from Selected

This view treats the selected function as the root and expanding it successively allows you to travel down the call tree. This produces a similar view to that of the **Call Tree**. The difference, though, is that if there are multiple calling functions in the **Call Tree**, this view will combine the selected function into a single entry. This may allow for easier viewing of the calls made by the selected function.

The **Total**, **Self**, and **Sub** values in the view are calculated as the percentage of the profile where the specific tree view was present.

Reverse from Selected

This view treats the selected function as the root and expanding it successively allows you to travel up the call tree. However, this is not just a simple reversal of the call tree, but instead separates and groups functions based on how they are called in order to show relationships that would otherwise be hard to see. For example, a vector normalize function might appear 10 times in the **Call Tree**, but it might be called from only 3 different functions. In this case, the vector normalize function will be listed once, with a child node for calling function.

The **Total**, **Self**, and **Sub** values in the view are calculated as the percentage of the profile where the specific tree view was present.

Legacy Reverse

Similar to **Reverse from Selected**, only in this case, the vector normalize function will be listed 3 times, with each entry showing the total time called from each parent function. This is the legacy behavior from previous versions of the profiler.

The **Total**, **Self**, and **Sub** values in this view are calculated as the percentage of the whole profile.

13.7 Units

To change the units displayed before each item, use the **Unit** buttons in the **Home** tab of the ribbon bar (see Units).

13.8 UNKNOWN STACK DETAILS

Sometimes it is not possible for the profiler to fully traverse the stack when a sample is taken. When this occurs, the profiler will mark this undiscoverable section of the callstack with an **UNKNOWN STACK DETAILS** node in the **Call Tree**. These undiscoverable sections tend to be created in the following situations:

- The depth of the stack gets modified by a variable value.
- Stack depth exceeds 64 KB.
- `nngxWaitVSync` may appear under this header if Automatic Frame Detection is in use.

14 Sample Graph Tab

The **Sample Graph** graphically depicts selected functions, threads, counters, and instrumented functions/ code blocks. Click on each item within their respective tabs to draw them in the **Sample Graph**.

When **Sample by Time** is selected, functions and threads each will be depicted as a rectangle starting at the time it was sampled and ending at the following sample. As sampling time can vary, the widths of these rectangles may vary, too. Additionally, if the same function or thread is sampled multiple times in a row, it will appear as an elongated rectangle.

If one of the **Sample by Performance Counter** choices was used from the **Sampling Strategy** drop down box, then function and thread samples will be drawn as a single vertical line at the time it was recorded. **Sample by Performance Counter** sampling may record samples at an irregular rate with respect to time (with very small or very large gaps between some samples). This is expected behavior since performance counter events occur irregularly.

A **Load per Frame** graph will also be drawn for each selected function and thread if a logged heartbeat or inferred heartbeat is selected for that core. The **Load per Frame** allows you to quickly assess the character of a given function on a frame-by-frame basis. The dark line shows the actual data and the highlight shows the margin of error (which adaptively follows the line to give an indication of the data's average value). Additionally, the selected **Units** in the **Home** tab of the **Ribbon Bar** will control the vertical axis of the **Load per Frame** graph.

Note: The vertical scale of the **Load per Frame** graph can be controlled by a drop down selection in the **Sample Graph** toolbar.

For counters, a line graph will be drawn between each value recorded. Most counters listed under Types of Counters accumulate between samples and are cleared after the sample is recorded, so the value of the counter at the time of the sample is only correlated with the function sampled. Conversely, user logged counters from inside the code are recorded instantaneously and graphed as such.

For instrumented functions and code blocks, the graphs depend on the type of data selected. However, since the recorded data is a constant value between the start and end a function or code block, each recorded chunk is drawn as a rectangle at the height of the value. The left edge of the rectangle is the time the function or code block was entered and the right edge of the rectangle is the time the function or code block was exited (thus making the width of the rectangle equal to the duration of the of the function call or code block). For example, when graphing **Time per Call** for an instrumented function, the height of the rectangle is the time spent in that function call (and all functions it calls) and the width of the rectangle is also the time spent in that function call. When graphing **Function Calls** for an instrumented function, the height of the rectangle is always 1 (because it was called one time for a given function call) and the width is the duration. When graphing a performance counter the height of the rectangle is equal to the counter value for the function call or code block and the width is the duration.

The **Sample Graph** tab is drawn using DirectX 10. If there is a problem initializing DirectX within the **Sample Graph** tab, an error message will be output to the screen containing details of the problem. Some DirectX troubleshooting is covered in Troubleshooting.

14.1 Sample Graph Tab Toolbar

The following controls are in the **Sample Graph** tab toolbar.

14.1.1 Mouse Mode

The left section in the **Sample Graph** toolbar controls the mouse mode within the **Sample Graph**. There are four modes:

1. **Scroll Graph**: In this mode, left-click-and-drag will scroll the graph up, down, left, and right.

   Keyboard modifiers:

   - Holding the Ctrl key will temporarily engage the **Measure Time** mouse mode until Ctrl is released.
2. **Select Frames**: In this mode, left-click or left-click-and-drag will select heartbeat frames. To deselect frames, left-click or left-click-and-drag on an already selected frame.

   Keyboard modifiers:

   - Frames can also be deselected by holding the Alt key along with a left-click-and-drag.
   - Holding the Ctrl key will temporarily engage the **Scroll Graph** mouse mode until Ctrl is released.
3. **Select Time**: In this mode, left-click-and-drag to select regions of time. Note that times will be snapped to sample boundaries within the core that was initially clicked on. To deselect time, left click or left click and drag on an already selected time.

   Keyboard modifiers:

   - Time can also be deselected by holding the Alt key along with a left click and drag.
   - Holding the Ctrl key will temporarily engage the **Scroll Graph** mouse mode until Ctrl is released.
4. **Measure Time**: In this mode, left click and drag to measure regions of time. When the left mouse button is released, the region will disappear.

   Keyboard modifiers:

   - Holding the Ctrl key will temporarily engage the **Scroll Graph** mouse mode until Ctrl is released.

In the **Select Frames** and **Select Time** modes, after selecting an area the **Functions**, **Threads**, **Call Tree**, and **Counters** tabs will recalculate based off of this selection.

Note: Some controls work in any mouse mode.

- The left and right arrow keys can be used to page left or right and the up and down arrow keys to zoom in and out.
- A function can be double-clicked to expand its direct children. This option is also available in the right-click context menu.

Caution: When selecting individual frames or times, the amount of statistical data is greatly reduced and the margin of error will dramatically rise. Double check the margin of error by enabling the Error button on the **Home** tab of the ribbon bar.

14.1.2 Deselect All Frames Button

To quickly deselect all selected frames or selected periods of time, click on the **Deselect All** button.

14.1.3 Icicle Graph

The **Icicle Graph** is a visualization of the call stack at every recorded sample. To best view the graph, select at least one function on a core, zoom into a single frame or set of frames, and check the **Icicle Graph** toggle button. It is recommended to zoom into a single frame or small set of frames since the graph is extremely detailed. The name **Icicle Graph** comes from the look of the graph, with root functions at the top and callstack leaves toward the bottom at the tips.

As a visualization of the callstack, the **Icicle Graph** shows the calling behavior from the root function down to the leaf of each callstack. To see individual function names and statistics (depth, total, and self), use the mouse to hover over the segment you are interested in. The color of each function is based on the automatic system classification performed by the profiler, with the color key located above each core header label. The automatic system classification might be incorrect for some functions, but on average it performs well and is helpful in understanding which systems are executing.

Icicle Graph rendering for multiple cores can be slow and will make the profiler feel sluggish to zoom or scroll. To keep rendering fast, it is recommended that you only draw an **Icicle Graph** for one core at a time (by not selecting functions/threads/counters on multiple cores). Since the **Icicle Graph** only draws on cores where a function, thread, or counter are selected, just restrict selection to one core.

When looking at individual frames, it is often the case that the **Load per Frame** graph is not helpful at this zoom level. To quickly hide the **Load per Frame** graph, deselect the **Load Graph** button in the **Sample Graph** toolbar.

14.1.4 Icicle Selected Graph

The **Icicle Selected** graph is similar to the **Icicle Graph**. However, instead of displaying colors for every function, only the currently selected functions are colored in the **Icicle Graph**. This has the benefit of making it easy to see where in a callstack a specific function was called at a particular time.

14.1.5 Stack Graph

The **Stack Graph** provides a quick and easy way to track the depth of the stack across a profile. The color of each bar is based on the leaf function's classification, with the color key located above each core header label.

If the profiler was unable to obtain the stack depth, the graph values will be fixed at 0.

14.1.6 Load Graph

This button provides a quick way to hide or show the **Load per Frame Graph** in the **Sample Graph**.

14.1.7 System Load

The **System Load** button provides a quick way to hide or show the **System Load** graph in the **Sample Graph**. The **System Load** drop-down box allows you to select which systems will be shown in the graph.

14.1.8 Heatmap Mode

Heatmap Mode stacks all frames on top of each other, drawing each sample faintly so that repetitive trends can be visualized. This mode is most helpful when sampling at a low rate (below 1000 samples per frame), since within a single frame there won't be enough data to see the true behavior. However, by stacking the samples on top of each other, you can leverage dozens or hundreds of frames to build up a visualization of how your game performs within a frame.

Since **Heatmap Mode** stacks frames, cores that do not have a heartbeat frame selected will be hidden in this mode. To force a heartbeat framing across all cores, go to the **Heartbeats** tab in the ribbon bar, find a desirable heartbeat on any core, then click **Apply to All Cores**.

The vertical black line that appears in **Heatmap Mode** represents the end of each stacked frame. The line is drawn faintly so that many stacked frames will accumulate to make it darker as they overlap each other. This is a good way to judge if frames are generally taking the same amount of time (by stacking directly on top of each other), whether they fall into several clusters of frame rates, or whether they are wildly scattered.

Often a profile contains a mix of frame rates which can make the **Heatmap Mode** difficult to understand. This is where you should leverage the **Group Selection** options in the **Heartbeats** tab. By selecting only singular frame rates (like **20Hz**), you can stack only these slow frames, thus eliminating noise from irrelevant frames. Additionally, this is a really good time to also use the **Story Analysis** in the **Analysis** tab only select important functions within these particular frames.

In **Heatmap Mode**, function names can get in the way of seeing the data. One option is to select **Short Names** or **Hide Names** in the **Sample Graph** ribbon bar.

14.1.9 Vertical Graph Scale

The vertical size or scale of graphs can be set using the **Vertical Graph Scale** drop-down in the **Sample Graph** toolbar. All changes are saved in the settings for the next time the profiler is loaded.

Practical vertical scales will be in the range of **1x** to **5x**. Use the larger vertical scales to precisely measure a particular value, since the y-axis will contain more resolution and unit labels.

14.1.10 Graph Cores

The **Graph Cores** drop-down box provides control over when complex graphs are displayed in the **Sample Graph** tab. When set to **Graph Cores as Selected**, graphs are only displayed for a particular core when a function or thread is selected on that core. In order to draw graphs for all cores regardless of function or thread selection, choose **Graph All Cores** from the drop-down box.

14.1.11 Self Samples / Total Samples / Mixed Samples

This drop down selection only appears for **Callstack** profiles that contain callstack data. With this drop down you can select between graphing samples by **Self Samples**, **Total Samples**, or **Mixed Samples**.

- Functions graphed as **Self Samples** will show a sample marker each time the function was sampled as a leaf function (when the program was stopped and sampled, execution was in that function).
- Functions graphed as **Total Samples** will show a sample marker each time the function was sampled as part of a callstack, regardless of whether it was a leaf function.
- Functions graphed as **Mixed Samples** will show the same samples as **Total Samples**, but function leaf samples (**Self Samples**) are drawn in a darker color.

For example, the function `nnMain` might have very few sample markers graphed when viewed with **Self Samples**, but it might have a very large number of sample markers graphed when viewed with **Total Samples** or **Mixed Samples**.

14.1.12 Adjusting Name Widths

When a function or thread is graphed, its name appears on the left side of the **Sample Graph**. The **Names** drop-down in the **Sample Graph** toolbar contains four settings that control how function name or thread name are displayed:

- **Hide Names**: This setting will hide all function names and thread names by not drawing them.
- **Short Names**: This setting will display shortened function names (limiting each to 100 pixels wide). For thread names, only the thread's ID will be shown.
- **Long Names**: This setting will display the function name without arguments. Thread names will be shown in full (ID and name).
- **Full Names**: This setting will display the entire function name including arguments. Thread names will be shown in full (ID and name).

14.1.13 Graph Tool Tips

The **Graph Tool Tips** drop down allows you to specify the type of tool tips you want to see when hovering over a function row, thread row, or **Icicle Graph** function.

- **Hide Tool Tips**: No tool tips are shown.
- **Show Name Only**: Only the name is shown.
- **Show Brief Stats**: The name, sample statistics, and frame number are shown.
- **Show Full Stats**: The name, thread, system classification, spike detection, periodicity, sample statistics, frame number, and exact time within the profile are shown.
- **Show Fn at Time**: The function or callstack will be shown at the exact time pointed at. This ignores the function or thread row and instead focuses on the horizontal position of the mouse within the duration of the profile.

14.1.14 View Control

The **Sample Graph** toolbar contains buttons to control the view:

- **Hard Scroll Left (left pointing arrow)**: Scrolls the view all the way to the left.
- **Scale to Fit (left and right arrows)**: Shows the entire profile, scaled horizontally to fit the window.
- **Hard Scroll Right (right pointing arrow)**: Scrolls the view all the way to the right.

14.2 Timeline

At the bottom of the **Sample Graph** is the **Timeline**. This shows the entire zoomed out profile as a frame rate graph. Overlaid on the **Timeline** is a window representing your current zoomed in area of the graph above. You can click and drag the window on the timeline as if it was a scrollbar. Also, clicking once on either side of the window will move it one window width to the left or right. The mouse wheel also works within the **Timeline** area to zoom in and out.

If the frame rate graph is not being displayed in the **Timeline**, it is because the game is not marked for frames properly. See Frame Marking on Nintendo 3DS for more information on how to do this. Alternatively, you can go to the **Heartbeats** tab and for a given heartbeat on a given core, you can choose **Use as Frame Rate** to have that heartbeat be drawn as the frame rate in the **Timeline**.

14.3 Zoom Control

The zoom (horizontal scale) can be controlled using the mouse wheel. The current level of zoom is displayed in the **Sample Graph** toolbar. If you want to zoom all the way out, the easiest method is to click the **Scale to Fit** button in the **Sample Graph** toolbar.

An alternative control for the zoom is the up and down arrow keys. This can be a good way to quickly zoom into the graph.

14.4 Right-Click Context Menu

A right-click in the tab will bring up a context menu containing a variety of options.

When the menu is up, there will be a highlight placed on the current function (orange horizontal rectangle) and the current time (orange vertical line). Particular choices in the context menu will affect either the current function or current time, so these are marked for reference.

14.4.1 Expand Direct Parents

The **Expand Direct Parents** option will select the parents of the currently selected function and place them above the currently selected function. Since a function can be called from many different parents, all parents are selected.

14.4.2 Expand Direct Children

The **Expand Direct Children** option will select the children of the currently selected function and place them below the currently selected function. A quicker way to accomplish this is to simply double click on the any function in the **Sample Graph**.

14.4.3 Deselect Direct Children

The **Deselect Direct Children** option is like an undo for the **Expand Direct Children** option.

14.4.4 Reorder

There are several **Reorder** options:

- **Chronologically**: Reorders all selected functions based on their first occurrence within the frame that right clicked. This is the same ordering used in the **Story Analysis**.
- **By Magnitude**: Reorders all selected functions based on how often each function was sampled. This is similar to the order in the **Functions** tab when it is sorted by **Total**.
- **By Periodicity**: Reorders all selected functions based on how periodic they are. Functions that are very periodic will be placed toward the top and functions that are sampled very irregularly will be placed toward the bottom.
- **By Function Name**: Reorders all selected functions alphabetically by function name. This might help to locate a particular function if you know the name and there are many functions displayed in the **Sample Graph**.

14.4.5 Deselect

There are several **Deselect** options:

- **This Function**: Deselects the currently highlighted function.
- **All Other Functions**: Deselects all functions other than the currently highlighted function.
- **All Functions On This Core**: Deselects all functions on the core of the currently highlighted function.
- **All Functions On Other Cores**: Deselects all functions on other cores of the currently highlighted function.
- **Above This Function**: Deselects all functions above the currently highlighted function.
- **Below This Function**: Deselects all functions below the currently highlighted function.

14.4.6 Move

There are several **Move** options:

- **Move Function Up**: Moves the currently highlighted function up one row.
- **Move Function Down**: Moves the currently highlighted function down one row.

14.4.7 Show Assembly

The **Show Assembly** option will move focus to the Assembly Tab, showing the assembly of the clicked function.

14.4.8 Show Details

The **Show Details** option will open a new Function Details Window, showing the details of the clicked function.

14.4.9 This Core At This Time

There are several **This Core At This Time** options:

- **Find Sampled Leaf Function**: Selects the sampled leaf function on this core at the currently highlighted time. The function will be placed at the top of the currently shown functions.
- **Find All Sampled Functions**: Selects all sampled functions (the callstack) on this core at the currently highlighted time. The functions will be placed at the top of the currently shown functions.

14.4.10 All Cores At This Time

There are several **All Cores At This Time** options:

- **Find Sampled Leaf Functions**: Selects all sampled leaf functions on all cores at the currently highlighted time. The functions will be placed at the top of the currently shown functions.

- **Find All Sampled Functions**: Selects all sampled functions (the callstack) on all cores at the currently highlighted time. The functions will be placed at the top of the currently shown functions.

14.4.11 Select Only This Heartbeat Frame

The **Select Only This Heartbeat Frame** option will deselect all frames/times and then select the current frame that is being hovered over with the mouse. This can be useful when you want to quickly focus on a specific frame for **Story Analysis** or **Spike Detection** (select the frame first and then activate the analysis).

If you want to select more than just one frame, change the mouse mode to **Select Frames** in the **Sample Graph** toolbar and use the left mouse button to select or deselect frames.

15 Code Coverage Tab

The **Code Coverage** tab allows you to track which functions have been seen and which have not been seen in a given profile. This can be helpful when trying to determine coverage for testing or for discovering unanticipated behavior.

Please note that loading a saved profile will clear the lists and they will be recalculated based on the saved profile. The lists are not saved with a profile but are instead generated based on the profile.

15.1 Functions Seen During Profiling

This window contains the names of functions that have been seen during profiling. The number of functions in the list is reported in the heading along with the combined code size. The combined code size is only visible when there is no filter applied.

15.1.1 Copy Button

The **Copy** button will copy to the clipboard whatever is currently shown using the current sorting and filtering. You can then paste the content into an email, document, or Excel spreadsheet. Because the copied content uses tab delimiters, it will retain the column organization when pasted into an Excel spreadsheet.

The **Copy** button takes into account what is currently selected. If any items are selected at the time the **Copy** button is pressed, then only the selected items will be copied. If nothing is currently selected, everything will be copied.

15.2 Functions Not Seen During Profiling

This window contains the names of functions that have not been seen during profiling. The number of functions in the list is reported in the heading along with the combined code size. The combined code size is only visible when there is no filter applied.

15.2.1 Copy Button

The **Copy** button will copy to the clipboard whatever is currently shown using the current sorting and filtering. You can then paste the content into an email, document, or Excel spreadsheet. Because the copied content uses tab delimiters, it will retain the column organization when pasted into an Excel spreadsheet.

The **Copy** button takes into account what is currently selected. If any items are selected at the time the **Copy** button is pressed, then only the selected items will be copied. If nothing is currently selected, everything will be copied.

15.3 Filtering Functions

Type any string into the text box and a case-insensitive search is performed for the items that contain the string. Only items matching the search text will be displayed. To negate the results (show everything that does not match the filter), click on the exclamation point button to the right of the text box. If there are no matches for the search, the text box will be highlighted orange.

Regular expressions are also supported (using .NET RegEx, which is Perl 5 compatible). To have the filter interpreted as a regular expression, click on the **RegEx** button to the right of the text box. Unlike basic filters, regular expressions are case-sensitive.

Helpful function filters can be found in the Helpful Simple Filters and Helpful Regular Expression Filters tables. Please see the Quick Regular Expression Primer for more information on regular expressions.

15.3.1 Saved Filters

Saved filters allow for using the same filters between profiling sessions, with any saved filters being loaded when the profiler is opened. By default, the **Filter Drop Down** box is loaded with example filters that you can customize. If you want to save your own filter in the drop down box, press the **Save** ("**+**") button after typing the filter (the **Negate** and **RegEx** button states will also be saved with the filter text). To delete a filter from the drop down box, select it first, then press the **Delete** ("**-**") button.

Note: The saved filters in the **Functions** tab will also be available in the **Call Tree** and **Code Coverage** tabs. However, please note that the **Filter Drop Down** box in the **Threads** tab and **Counters** tab are not shared since they are used to search for thread and counter names, not function names.

15.4 Right-Click Context Menu

A right-click in the tab will bring up a context menu containing a variety of options.

15.4.1 Copy Function Name to Clipboard

This will copy the full function name with arguments to the clipboard.

15.4.2 Select Function

The **Select Function** option results in the clicked function becoming selected in the **Functions** tab.

15.4.3 Show Assembly

The **Show Assembly** option will move focus to the Assembly Tab, showing the assembly of the clicked function.

15.4.4 Show Details

The **Show Details** option will open a new Function Details Window, showing the details of the clicked function.

16 Assembly Tab

The **Assembly** tab allows you to see where in a function samples occurred. This can be helpful in tracking down specific problem spots within a function.

16.1 Assembly Tab Toolbar

The following controls are in the **Assembly** tab toolbar.

16.1.1 Copy Button

The **Copy** button will copy to the clipboard whatever is currently shown using the current sorting and filtering. You can then paste the content into an email, document, or Excel spreadsheet. Because the copied content uses tab delimiters, it will retain the column organization when pasted into an Excel spreadsheet.

The **Copy** button takes into account what is currently selected. If any items are selected at the time the **Copy** button is pressed, then only the selected items will be copied. If nothing is currently selected, everything will be copied.

16.1.2 View Controls

In order to more readily facilitate looking through large functions with few samples, a **Previous** and **Next** button are provided. These will scroll to the previous or next off-screen line of assembly, respectively, that has non-zero count in either self or sub.

16.1.3 Show Source

It is possible to view interleaved source code with the assembly. This information is obtained from the ELF files stored on your hard drive. As such, there are a few limitations on viewing source code.

This button works as a toggle. On each profiler startup it will default to being disabled. If the button is enabled, selecting a new function for viewing will automatically attempt to pull in source code for the function.

Note: Clicking on **Show Source** will force the sorting to **Address** and then interleave the source code.

16.1.3.1 Limitations on Viewing Interleaved Source Code

Provided here is a list of limitations on viewing interleaved source code.

- The Cygwin tools package `binutils` must be installed on the computer. The profiler makes of the `addr2line` tool in the package to obtains which source code line pertains to each assembly instruction. If the `binutils` package cannot be located, the message "`Unable to find addr2line in Cygwin installation. Please install binutils package.`" will be displayed.
- The original ELF files must exist on your computer in the same locations in which they were previously found when the profile was initially taken. If the required ELF cannot be located, the message "`Unable to locate` *`ElfName`*" will be displayed.
- The ELF on the computer must have a date that is older than the date/time at which the profile was taken. If the date on the ELF is newer, than it is not possible that the ELF corresponds with the current profile data. If this situation occurs, the message "`ELF modified date/time is newer than the profile date/time`" will be displayed.

- Individual source files are not checked for date/time stamps. As such, it is possible that the source file has been modified and line numbers changed since the ELF was created. In this case, incorrect lines will be displayed in the assembly view. Be mindful of this limitation while using the interleaved source code view.

16.2 Selecting a Function to View

When the **Assembly** tab is open, click on a function in the **Functions** tab to show its data in the **Assembly** tab. The currently selected function is shown to the right of the text **Selected Function** in the toolbar area of the **Assembly** tab. Please note that the selected function will not be highlighted in the **Functions** tab.

Functions that are RPL internal functions are not viewable. If one of these functions is selected, a message will display in the assembly window along with **Self** and **Sub** counts for the function. These should match the counts visible in the **Functions** tab.

16.3 Assembly View

This is the area of the tab which actually displays the assembly source code. Each line of source code contains the following information:

- **Address**: The address at which the line of assembly appears in memory.
- **Self**: The number of samples taken when the program counter was at that address, executing that line of assembly.
- **Sub**: The number of samples taken where this function was above the sampled instruction in the **Call Tree**. As such, this number will only appear on branch-with-linking instructions in **Callstack** profiles.
- **Assembly**: The assembly instruction that is stored at that address.

The Nintendo 3DS CPU Profiler uses a different disassembler than the one provided by the provided compiler. As such, a direct comparison of the assembly against the compiler tools version will result in slightly different text. However, the difference should be in the mnemonic that is used (or not used as the case may be).

The columns can be resized and reordered as much as desired. However, when the **Copy** function is used, the default column ordering and sizes are used.

16.3.1 Highlighting

In order to make it easier to visually assess which areas of the assembly are seen more often in profiling, the line with the most **Self** samples is colored the darkest red color. Lighter shades are used for lessening **Self** hit counts. If there are more **Sub** samples than there are **Self** samples, than the color will change varying shades of a yellow-green color.

16.3.2 Sorting

Rudimentary sorting is available for the assembly view. The default sorting is by ascending **Address**. Any time a new function is selected to view the sorting mode is set back to the default.

To change the sorting, click on the column header to sort by that column. Each column can only be sorted a single direction. **Address** and **Assembly** both use an ascending sort while **Self** and **Sub** both use a descending sort.

Note: When any of the column headers are clicked **Show Source** is automatically disabled.

16.4 Right-Click Context Menu

A right-click in the tab will bring up a context menu containing a variety of options.

Options that target a specific function are only enabled when the assembly instruction is a branch.

Options that target data are only enabled when there is a literal pool in the function.

16.4.1 Select Target Function

The **Select Target Function** option results in the clicked function becoming selected in the **Functions** tab.

This option is only enabled if the target function was seen in the profile. If the target function was not seen then the option will be disabled.

16.4.2 Jump to Target

Selecting the **Jump to Target** option will result in the **Assembly View** showing the target address. If the jump occurs within the same function the **Assembly View** scrolls to bring the target address to the top of the window. If the jump is to another function then the **Assembly View** changes the selected function to the target function.

16.4.3 Show Target Details

The **Show Target Details** option will open a new Function Details Window, showing the details of the target function.

16.4.4 Copy Target Function Name

The **Copy Target Function Name** option will copy the full function name with arguments to the clipboard.

16.4.5 View Data As

When a function contains a literal pool the profiler looks at the data in the literal pool and attempts to automatically determine the type of that data. The **View Data As** options allow for overriding the automatically assigned type. The following types are currently allowed for assignment:

- **Int32**: A 32-bit signed integer
- **UInt32**: A 32-bit unsigned integer
- **Single**: A 32-bit floating-point value
- **String**: A 4 byte array of extended ASCII (ISO 8859-1) characters
- **Function**: Treat the data as if it is a function address

17 Console Tab

The **Console** tab can be used to show debug output written from the dev kit. Output is only available when Synced to a dev kit with the **Enable** button toggled to the on position.

The Console tab acts like a console window, with new output appearing at the bottom of the window. Approximately 1000 lines can be buffered before beginning to lose previous lines output.

17.1 Console Tab Toolbar

The following controls are in the **Console** tab toolbar.

17.1.1 Enable Button

By default the **Console** does not display debug output from the dev kit. In order to get debug output, you must toggle the **Enable** button. To stop receiving debug output, toggle the **Enable** button again.

17.1.2 Clear Button

The **Clear** button clears everything from the window.

17.1.3 Copy Button

The **Copy** button will copy to the clipboard whatever is currently shown using the current sorting and filtering. You can then paste the content into an email, document, or Excel spreadsheet. Because the copied content uses tab delimiters, it will retain the column organization when pasted into an Excel spreadsheet.

The **Copy** button takes into account what is currently selected. If any items are selected at the time the **Copy** button is pressed, then only the selected items will be copied. If nothing is currently selected, everything will be copied.

17.1.4 Text Format Selection

The **Text Format** drop-down allows for changing the format used to decode text output from the dev kit. This rule is applied as text is received and is not retroactively applied.

Valid selections are:

- **UTF-8**
- **Shift-JIS**

18 Function Details Window

The **Function Details** window provides additional information about a selected function. The following information is displayed:

- The name of the function.
- The module the function was found in.
- The address of the function.
- The size, in bytes, of the function.
- The number of instructions in the function.
- The functions that this function calls.
- The functions that call this function.

Additionally, if the function was seen during a profile, the following details are also displayed.

- The minimum **Total** % of the function (only displayed for **Callstack** profiles; found by finding highest **Total** % in the call tree for all threads and cores).
- The combined **Self** % of the function (found by summing all **Self** % together across all threads and cores).
- The threads on which the function was seen.

18.1 Selecting a Function to Display

Selecting a function to display in the **Function Details** window occurs by using the right-click context menu. Right-click on the function you wish to view details for and select **Show Details**. The **Functions Details** window will then be displayed. It must be closed before continuing to interact with the rest of the profiler.

The following tabs in the profiler contain a context menu option to show function details:

- Functions Tab
- Checkers Tab (some, but not all of the Results Tables)
- Checkers Tab
- Call Tree Tab (both regular and reverse call tree)
- Sample Graph Tab
- Code Coverage Tab

18.2 Function Details Operations

The following operations are supported from the **Function Details** window.

18.2.1 Show Assembly

The **Show Assembly** option will move focus to the Assembly Tab, showing the assembly of the clicked function.

18.2.2 Select Function

The **Select Function** option results in the clicked function becoming selected in the **Functions** tab.

18.2.3 Traversing Function Calls

To change which function the **Function Details** window is looking at, it is possible to double-click on a function in either the **Calls Functions** or **Called by Functions** lists. This will switch to displaying the details for the selected function. This process can be continually repeated.

19 System Classifications

Every function is automatically classified as to what system it belongs to (Physics, Graphics, Audio, etc). This is primarily accomplished with string matching using commonly accepted terms for each system (for example, "broadphase" for Physics or "matrix" for Math). It is fully expected that some functions will be classified incorrectly, but since it is mostly accurate, it is still immensely helpful in aiding understanding.

Selected functions will be color coded to their system classification when selected. This is reflected in the **Functions** tab, **Call Tree** tab, and **Sample Graph** tab. The key for what each color means is displayed above each core in the **Sample Graph** tab

Functions can be filtered by their system classification by using the **System Classification** drop-down box on the **Functions** tab toolbar.

20 Performance Counter Groups

The CPU contains two performance counters that can be configured to count events inside the CPU. Since there are several individual performance counters, the profiler has grouped similar or complementary performance counters into **Performance Counter Groups**. The profiler allows you to record a particular performance counter group (per-core) each time a sample is taken. This section goes into more detail as which performance counter groups exist, and what kind of data they record.

20.1 Performance Counters Disabled

This is the default selection. When this option is selected, no performance counter data will be recorded with the profile data.

20.2 Instruction Misses and Bus Contention

This metric helps determine if your code is thrashing the cache. Additionally, if cycles per missed instruction is greater than 32, this indicates the bus is extremely busy (due to GFX, audio, etc). Two graphs are produced from these counters: **Instruction Cache Misses** and **Cycles per Missed Instruction**.

Analyzing Instruction Cache Misses

The more instruction cache misses, the more the code is thrashing in the cache. However, this graph may or may not indicate a problem. Please note that this graph is highly sensitive to the number of samples taken, since the number of instruction cache misses accumulate between samples (this is the only performance counter graph that is sensitive to the sampling rate). Because of this sampling rate sensitivity, it is not possible to claim that something like 5K instruction cache misses is good or bad without looking at the overall graph and considering how 5K compares to the min, max, or average. Generally, higher values in the graph mean more thrashing (the working set of the code might be too big for the cache). If you see high values, consider how you might reduce the size of the working set for the code at that point in time. Solutions to consider: use Thumb code, optimize for size (instead of optimize for speed), reuse common subroutines, and/or disable or reduce inlining.

Analyzing Cycles per Missed Instruction

This graph counts the number of CPU cycles waiting for an instruction cache line to be filled. On CTR, the minimum will be 32 cycles for one cache line. Unlike the top graph, this graph is "normalized" and the numbers are consistent regardless of the sampling rate (32 is the minimum and is considered really good, and higher than 80 or 100 is considered bad). Actually, this graph shows bus activity (so this metric is a proxy metric for bus activity since we can't measure bus activity directly). As a result, this graph gives you an idea of how busy the bus is at different points in the frame. If the bus is busy, it's due to other clients such as GPU, DSP/audio, network, card, LCD, etc. For example, there is a noticeable difference in activity when the display buffers are in FCRAM as opposed to VRAM (longer bus wait time when using FCRAM).

20.3 Instruction and I-Cache Efficiency

This metric helps determine the amount of time waiting for instruction cache lines to be filled. Additionally, it shows the instructions per cycle (IPC), which measures overall system effectiveness. Two graphs are produced from these counters: **Percent Cycles Stalled Waiting for Instruction** and **Instructions per Cycle**.

Analyzing Percent Cycles Stalled Waiting for Instruction

Typically for CTR, it seems that about 30% of CPU cycles are wasted waiting for instructions to be fetched into the cache (less than 30% indicates the code is in a tight loop, such as memcpy). As the bus traffic from other systems increases, this metric gets worse (larger) since the CPU must wait longer to get instructions from main memory. This graph is "normalized" so it is consistent regardless of sampling rate. While the "Cycles per Missed Instruction" graph tells you the actual amount of time you were waiting for each instruction to come into the cache, this graph lets you understand the relation between the waiting time and the overall amount of time available to the CPU; this relation is a measure of the efficiency of the CPU + cache + bus + FCRAM and how well they work in harmony.

Analyzing Instructions per Cycle (IPC)

This graph also gives a measure of overall CPU efficiency. On CTR, the ideal is 1.0 or higher but typically the value is around 0.2 (based on games so far - the max ever seen was 0.55). Many things can contribute to a low IPC, such as waiting for memory, waiting for previous computations to complete, or waiting for a co-processor (such as the VFP). Therefore, use this graph to find the problem areas of your frame, then investigate the cause with the other graphs.

20.4 Memory and Compute Performance

This metric helps determine the amount of time the CPU is stalled waiting for instructions or data. Data stalls occur when loading data from memory or from waiting for results to be computed. Three graphs are produced from these counters: **Percent Stalled due to Instruction Cache Misses**, **Percent Stalled due to Data Hazards**, and **Percent Stalled due to Unavailable Data/Instructions (lower bound)**. This last graph is the max of the previous two graphs (unfortunately it is not possible to compute the exact percent stalled due to all issues, given the existing CPU performance counters. Only the lower bound can be computed.).

Analyzing Percent Cycles Stalled Waiting for Instruction

Typically for CTR, it seems that about 30% of CPU cycles are wasted waiting for instructions to be fetched into the cache (less than 30% indicates the code is in a tight loop, such as memcpy). As the bus traffic from other systems increases, this metric gets worse (larger) since the CPU must wait longer to get instructions from main memory. This graph is "normalized" so it is consistent regardless of sampling rate. While the "Cycles per Missed Instruction" graph tells you the actual amount of time you were waiting for each instruction to come into the cache, this graph lets you understand the relation between the waiting time and the overall amount of time available to the CPU; this relation is a measure of the efficiency of the CPU + cache + bus + FCRAM and how well they work in harmony.

Analyzing Percent Cycles Stalled Waiting for Data Hazards

This graph shows the percentage of cycles stalled in the CPU due to a data hazard (waiting for data from main memory or waiting for results to be computed). Large spikes indicate areas that might benefit from rearranging data in memory, reducing the size of data, or performing other simultaneous work while computing or retrieving data.

Analyzing Percent Cycles Stalled Waiting due to Unavailable Data/Instructions (lower bound)

This graph is the maximum of the two previous graphs. This represents the lower bound of how much the CPU is stalled due to instruction cache misses and data hazards. Since an instruction cache miss and a data hazard could happen simultaneously, it would be wrong to add the two above graphs. However, by taking the max of the two above graphs, this represents the minimum the CPU is waiting for any reason (but it is likely waiting longer than what this graph shows).

20.5 Data Cache Read Performance

This metric helps determine the percentage of reads that miss in the data cache, resulting in loads from main memory. One graph is produced from these counters: **Data Cache Read Miss Percentage**.

Analyzing Data Cache Read Miss Percentage

This graph helps determine the percentage of reads that miss in the data cache, resulting in loads from main memory. Ideally the percentage should remain very low. If the percentage spikes, it means that the working data set is too large fit in the cache, the data set has poor spatial locality (ping-pongs around memory), or that the data set is being loaded for a single pass through the data (and not reused).

20.6 Data Cache Write Performance

This metric helps determine the percentage of writes that miss in the data cache, resulting in loads from main memory. One graph is produced from these counters: **Data Cache Write Miss Percentage**.

Analyzing Data Cache Write Miss Percentage

This graph helps determine the percentage of writes that miss in the data cache, resulting in loads from main memory (a cache line must be read in before it can be altered). Ideally the percentage should remain very low. If the percentage spikes, it means that the data set has poor spatial locality (ping-pongs around memory) or that the data set is being written for a single pass through the data (and not rewritten).

20.7 Pressure on Load/Store Queue

This metric helps determine if the number of load/store operations is too high (queue full) in a given part of the code. One graph is produced from these counters: **Percent Cycles Stalled Due to Load Store Queue Full**.

Analyzing Percent Cycles Stalled Due to Load Store Queue Full

This metric helps determine if the number of load/store operations is too high (queue full) in a given part of the code. This indicates that code is trying to load/store too many things at once and should maybe space it out a little to better avoid this particular stall. In practical terms this helps identify functions or parts of the code with high memory traffic.

20.8 Mispredicted Branches

This metric helps determine if branches are being overly mispredicted. This indicates that performance may be gained by reversing certain logic tests. One graph is produced from these counters: **Percent of Branches Mispredicted**.

Analyzing Percent of Branches Mispredicted

This metric helps determine if branches are being overly mispredicted. This indicates that performance may be gained by reversing certain logic tests. Unfortunately, the percentages of mispredicted branches is much higher than this graph indicates (a high of 12% could actually be 80%), but the relative shape is accurate. The scale of the mispredicted branches is artificially low because all branches are being considered whether they are conditional branches or not (mispredicted branches / all branches). An accurate graph cannot be calculated because there is no performance counter or combination of performance counters to produce: (mispredicted branches / all conditional branches). The current graph is the best that can be computed with respect to mispredicted branches.

20.9 Correlation Between Sampled Functions and Performance Counters

There is only a very weak correlation between the sampled functions and performance counter data, so be careful not to attribute too much meaning to any coinciding data. For example, if there is a large percentage of data cache write misses at the same time the function `strcmp` was sampled, it is unlikely that the function `strcmp` caused the misses. This is because the performance counter was accumulating data between samples (during which many functions were called) and then at the time of the sample, the program counter happened to be in the `strcmp` function when the performance counter was recorded and cleared. The higher the rate of sampling, the stronger the correlation, but unfortunately the profiler recording code will heavily taint the results (causing extra instruction and data cache misses as well as interfering with the other metrics).

While you might not be able to identify a particular function responsible for a performance counter problem, it might be possible to attribute performance counter data to a particular system or area or the code. For example, if there is a large percentage of data cache read misses right before rendering, you can perhaps attribute the misses to code that executed before rendering, based on knowledge of your own game and the clues given by what functions were called. For example, you might determine that the animation system was responsible for the large percentage of data cache read misses because that particular system executed right before rendering.

With the use of the **Heatmap** mode, the correlation between sampled functions and performance counters is much stronger and might be somewhat more reliable.

21 Command Line Options

In order to provide more control over some of the behavior of the profiler GUI, some command-line options have been provided. The format of the command line should be as follows:

21.1 Command Line Format

```
"Nintendo CPU Profiler.exe" [--options] [NPROF] [ELF] [Executable]
```

[--options]

Zero or more of the options listed in Available Options.

[NPROF]

Enter in a filename of a previously saved NPROF (`C:\path\file.nprof`) to have it loaded automatically when the profiler starts.

[ELF]

Enter in the path to an AXF file that you would like to use for syncing (C:\path\example.axf). This provides the same functionality as the --loadelf option. This tells the GUI that it should perform a **Sync** with the specified file as soon as the **Sync** option becomes available. If Executable is not specified the profiler will attempt to find it automatically.

[Executable]

Enter the path to either a CCI or CCL file to be used for syncing (C:\path\example.cci). This tells the GUI that it should perform a **Sync** with the specified file as soon as the **Sync** option becomes available. If ELF is not specified the profiler will attempt to find it automatically.

Important: The NPROF parameter should not be specified with the ELF or Executable parameters.

21.2 Available Options

--autosave[=filename]

This tells the GUI that it should save out the results of the profile immediately, rather than requiring a button press. Only the NPROF will be saved if this option is used.

The filename in the command line is optional. If not present, the filename will be based on the currently loaded ELF and will be saved into the same folder as the ELF. The final filenames used will be in the form: `filename-YYYYMMDD-hhmmss.nprof`

--loadelf=filename

(Deprecated) This option has been replaced by just specifying the file directly on the command line. This tells the GUI that it should perform a **Sync** with the specified file as soon as the **Sync** option becomes available.

--saveonerror

(Deprecated) Specifying this option will save an NPROF even if there was a problem in parsing the data. This is primarily intended as a feature to enable sending data to the profiler team in case of a problem in parsing what is believed to be valid data. (It was determined that this feature was too useful to leave as an optional setting. It is now always enabled.)

--script=filename

When the profiler starts up it will automatically run the specified script. This operation will be performed after a specified NPROF is loaded, but may occur before --loadelf has completed.

--shutdown

This option will shutdown the profiler GUI after a profile has been received. It is recommended that this option only be used with the --autosave option.

22 Scripting

The profiler supports loading in Windows PowerShell script files in order to perform automated GUI control. This can be useful if you want to take the same suite of profiles multiple times.

22.1 Windows PowerShell

Windows PowerShell scripting features allow for fairly complex behavior. It is beyond the scope of this document to go into a background of this scripting language. You can find more details on Windows PowerShell scripting from Microsoft.

22.2 Cmdlets

The profiler allows for scripting by creating new cmdlets to be used. All built-in cmdlets should also be available. Extended cmdlets that have been installed separately may not be available.

All of the profiler's cmdlets start with the prefix 'PROFILER-'.

Note: In Windows PowerShell, cmdlets are not case-sensitive.

Profiler cmdlet arguments are space separated and text not enclosed in quotes is converted to upper-case for internal comparisons. This will cause a problem with function names. As such, all function names will need to be enclosed in quotation marks (""). Quotation marks will also allow for C++ function names with multiple arguments and/or template information to contain spaces if needed.

If the argument name is surrounded by [], passing in the name is optional. If the argument name is omitted than the order of the arguments must be passed in the order shown.

22.2.1 PROFILER-ANALYZE

Asks the profiler to perform an analysis of the profile data.

Story Analysis

This option has the profiler analyze the data using one of the Story Analysis modes discussed in Story Analysis.

```
PROFILER-ANALYZE -Story <StoryType>
```

-Story <StoryType>

One of the types of Story Analysis options. Options listed in Story Types.

Spike Detection

This option has the profiler analyze the data using one the Spike Detection modes discussed in Spike Detection.

```
PROFILER-ANALYZE -SpikeDetection <SpikeDetectionType>
```

-SpikeDetection <SpikeDetectionType>

One of the types of Spike Detection options. Options lised in Spike Detection Types.

Examples

```
PROFILER-ANALYZE -Story Full
PROFILER-ANALYZE -SpikeDetection Rare
```

22.2.2 PROFILER-ASSEMBLY

Controls the options for interacting with the **Assembly** tab.

Set Function

`PROFILER-ASMFUNCTION -FunctionName <String>`

-FunctionName <String>

The string provided must match the function name that is displayed in the profiler exactly. If using C++ this will include the arguments and template information.

Show Source

`PROFILER-ASSEMBLY -ShowSource <Boolean>`

-ShowSource <Boolean>

Specifies if source code should be shown.

Examples

```
PROFILER-ASSEMBLY -FunctionName "TestFunction"
PROFILER-ASSEMBLY -ShowSource $true
```

22.2.3 PROFILER-CALLSTACKS

Sets whether to record callstacks in a Sampled profile or not.

`PROFILER-CALLSTACKS [-ShowCallstacks] <Boolean>`

[-ShowCallstacks] <Boolean>

Specifies if callstacks should be recorded.

Example

```
PROFILER-CALLSTACKS $false
```

22.2.4 PROFILER-COPY

Tells the profiler to copy the contents of a specified tab to the clipboard.

```
PROFILER-COPY [-Tab] <ScriptingTabs> [-ExtendedParam
<ScriptingCopyExtensions>]
```

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

[-ExtendedParam <ScriptingCopyExtensions>]

Optional. Certain tabs have more than one copy option. The extended options are listed in Copy Extensions.

Example

```
PROFILER-COPY Functions
PROFILER-COPY CallTree Selected
PROFILER-COPY CodeCoverage -ExtendedParam Seen
```

22.2.5 PROFILER-FILTER

Sets a filter for the specified tab.

There are a few different forms to this cmdlet. Not all forms are compatible with all tabs.

By Name

This option is the default form of PROFILER-FILTER. It will result in the Filter/Find box on the tab being modified.

```
PROFILER-FILTER [-Tab] <ScriptingTabs> [-Negate] <Boolean> [-RegEx] <Boolean>
[-FilterName] <String>
```

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

[-Negate] <Boolean>

Specifies if the results should be negated.

[-RegEx] <Boolean>

Specifies if the filter is a regular expression.

[-FilterName] <String>

The text to use for the filter. It is recommended that this text be placed in quotation marks, especially if this is a regular expression filter.

Limit

This option will either Limit the results in the Function Tab or allow all results in the Function Tab to be shown.

```
PROFILER-FILTER [-Tab] <ScriptingTabs> -Limit <Boolean>
```

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

Limit <Boolean>

Whether or not to limit the visible results.

Show Selected

This option will set the Function Tab to only show the selected functions.

```
PROFILER-FILTER [-Tab] <ScriptingTabs> -ShowSelected <Boolean>
```

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

ShowSelected <Boolean>

Whether or not to show only the selected items.

By Core

This option will filter the currently visible results by only showing the results from the specified core.

```
PROFILER-FILTER [-Tab] <ScriptingTabs> -Core <Object>
```

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

Core <Object>

Specify either the Integer core number to view, or the string "All" to show all cores.

By Thread

This option will filter the currently visible results by only showing the results from the specified thread.

```
PROFILER-FILTER [-Tab] <ScriptingTabs> -Thread <String>
```

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

Thread <String>

The name of the thread to filter. Specify "All" to see all threads.

By Group

```
PROFILER-FILTER [-Tab] <ScriptingTabs> -Group <String>
```

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

Group <String>

The name of the group to filter. Specify "All" to see all groups.

By System

```
PROFILER-FILTER [-Tab] <ScriptingTabs> -System <String>
```

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

System <String>

The name of the system to filter. The name is the same as what is displayed in the System Classification Filter. Any value that does not match one in the filter results in defaulting back to the **All Systems** option.

Examples

```
PROFILER-FILTER Functions $false $true "^OS"
PROFILER-FILTER Functions -Negate $false -RegEx $true -FilterName "^OS"
PROFILER-FILTER Functions -ShowSelected $true
PROFILER-FILTER Threads -Core 0
PROFILER-FILTER Functions -Thread "ThreadName"
PROFILER-FILTER Counters -Group "GroupName"
PROFILER-FILTER Functions -System "Graphics"
```

22.2.6 PROFILER-INSTRUMENT

Sets the function you wish to instrument based on the specified function name.

To set a function, the symbol must be known by the profiler GUI. This is most easily accomplished by taking a quick Sampled profile before attempting to take an Instrumented profile.

`PROFILER-INSTRUMENT [-FunctionName] <String>`

[-FunctionName] <String>

This must match the function name that is displayed in the profiler exactly. If using C++ this will include the arguments and template information.

Example

```
PROFILER-INSTRUMENT "TestFunction"
```

22.2.7 PROFILER-MARKED

Sets whether Marked code should be recorded with the profile data.

Although the GUI allows this value to be set independently on each tab, this cmdlet will set the flag to record Marked code for both Sampled and Instrumented profiles.

`PROFILER-MARKED [-ShowMarkedCode] <Boolean>`

[-ShowMarkedCode] <Boolean>

Specifies if marked code should be recorded in the profile data.

Example

```
PROFILER-MARKED $false
```

22.2.8 PROFILER-OPEN

Opens an NPROF file.

`PROFILER-OPEN [-FileName] <String>`

[-FileName] <String>

The full path to the NPROF file to open. It is recommended that this path name be enclosed in quotation marks.

Example

```
PROFILER-OPEN "C:/Profiles/SavedProfile1.nprof"
```

22.2.9 PROFILER-PERFCOUNTERS

Sets which performance counters to record with both Sampled and Instrumented profiles.

Although the GUI allows for setting these independently, this cmdlet will set both at the same time.

```
PROFILER-PERFCOUNTERS [-PerfCounters] <ScriptingPerfCounters>
```

[-PerfCounters] <ScriptingPerfCounters>

The Performance Counter values are detailed in Scripting Perf Counter.

Example

```
PROFILER-PERFCOUNTERS Disabled
```

22.2.10 PROFILER-SAMPLE

Sets both the Sample Strategy and Sample Rate for Sampled profiles.

```
PROFILER-SAMPLE [-Strategy] <ScriptingSampleStrategy> [-Rate] <Integer> [-
NoWarning]
```

[-Strategy] <ScriptingSampleStrategy>

Strategy values are detailed in Scripting Sample Strategy.

[-Rate] <Integer>

Rate values are detailed in Rate.

[-NoWarning]

This flag keeps a warning from being generated when an invalid Strategy and Rate are combined.

Returns

A Boolean value indicating whether the Strategy and Rate were set correctly.

Example

```
PROFILER-SAMPLE Time 1000
$success = PROFILER-SAMPLE -NoWarning -Strategy Time -Rate 100
```

22.2.11 PROFILER-SAMPLEGRAPH

Controls the Sample Graph options.

ShowCombined

Control whether the combined line should be visible.

```
PROFILER-SAMPLEGRAPH -ShowCombined <Scripting.SampleGraph.CombinedLineDisplay>
```

-ShowCombined <Scripting.SampleGraph.CombinedLineDisplay>

Sets how the combined line is shown. Combined line display options are detailed in Scripting.SampleGraph.CombinedLineDisplay.

SampleDisplay

Control how samples are displayed.

```
PROFILER-SAMPLEGRAPH -SampleDisplay <Scripting.SampleGraph.SampleDisplay>
```

-SampleDisplay <Scripting.SampleGraph.SampleDisplay>

Sets how samples are displayed. Sample display options are detailed in Scripting.SampleGraph.SampleDisplay.

NameDisplay

Control how names are displayed.

```
PROFILER-SAMPLEGRAPH -NameDisplay <Scripting.SampleGraph.FunctionNameDisplay>
```

-NameDisplay <Scripting.SampleGraph.FunctionNameDisplay>

Sets how function names are displayed. Function name display options are detailed in Scripting.SampleGraph.NameDisplay.

Example

```
PROFILER-SAMPLEGRAPH -ShowCombined Only
PROFILER-SAMPLEGRAPH -SampleDisplay Mixed
PROFILER-SAMPLEGRAPH -NameDisplay Short
```

22.2.12 PROFILER-SAVE

Saves the current profile data.

```
PROFILER-SAVE [[-FileName] <String>]
```

[[-FileName] <String>]

Optional. The full path of the file to save. It is recommended that this path name be enclosed in quotation marks. If no value is passed in, the file will be saved to the current directory with the date and time for its name.

Example

```
PROFILER-SAVE
PROFILER-SAVE "C:/Profiles/SavedProfile1.nprof"
```

22.2.13 PROFILER-SELECT

Selects items in the specified tab.

TopGroup

Selects groups of 45 elements at once.

`PROFILER-SELECT [-Tab] <ScriptingTabs> -Top [[-TopGroup] <Integer>]`

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

-Top

Specifies that this will be selecting groups of 45 elements at once.

[[-TopGroup] <Integer>]

Which group will be selected. If unspecified, the first 45 elements are selected.

List

Selects elements at the specified indices from the tab (indices run from top to bottom, with the top being 0).

`PROFILER-SELECT [-Tab] <ScriptingTabs> -List <Integer[]>`

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

-List <Integer[]>

A list of the indices to select.

DeselectAll

Deselects all currently selected elements.

`PROFILER-SELECT [-Tab] <ScriptingTabs> -DeselectAll`

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

-DeselectAll

Specified to deselect all selected elements on the tab.

Example

```
PROFILER-SELECT Functions -Top
PROFILER-SELECT Functions -Top 3
PROFILER-SELECT Functions -List 0 1 4 5
PROFILER-SELECT Functions -DeselectAll

[int[]]items = 0, 3
PROFILER-SELECT Functions -List @items
```

22.2.14 PROFILER-SHOWTAB

Brings a tab into view.

`PROFILER-SHOWTAB [-Tab] <ScriptingTabs>`

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

Example

```
PROFILER-SHOWTAB SampleGraph
```

22.2.15 PROFILER-START

Starts taking a profile of the specified type.

```
PROFILER-START [-ProfileType] <ScriptingProfileType>
```

[-ProfileType] <ScriptingProfileType>

The Profile Type values are specified in Scripting Profile Type.

Example

```
PROFILER-START Sampled
PROFILER-START Instrumented
```

22.2.16 PROFILER-STOP

Stops the profile session.

```
PROFILER-STOP
```

Example

```
PROFILER-STOP
```

22.2.17 PROFILER-SYNC

Syncs to the runtime application using the specified ELF.

Rather than use this for automation, it is generally recommended that you use the Command Line Options to specify an ELF to load automatically.

```
PROFILER-SYNC [[-FileName] <String>]
```

[[-FileName] <String>]

Optional. The full path to the ELF to load. It is recommended that this path name be enclosed in quotation marks. If no ELF is specifically specified, the GUI will attempt to determine what ELF is currently in use to load that.

Returns

A Boolean value if the Sync was successful or not.

Example

```
PROFILER-SYNC
$success = PROFILER-SYNC "C:/Game/bin/NDEBUG/game.elf"
```

22.2.18 PROFILER-TREECONTROL

Tells the profiler to perform actions on the tree present on the specified tab.

```
PROFILER-TREECONTROL [-Tab] <ScriptingTabs> [-Action]
<ScriptingTreeControlActions>
```

[-Tab] <ScriptingTabs>

Tab names are listed in Scripting Tabs.

[-Action] <ScriptingTreeControlActions>

Specifies the action to apply on the tree. Available actions are listed in Scripting Tree Control Actions.

Example

```
PROFILER-TREECONTROL CallTree ExpandAll
PROFILER-TREECONTROL -Tab Info -Action CollapseAll
```

22.2.19 PROFILER-UNITS

Specifies which units should be displayed in the GUI.

```
PROFILER-UNITS [-UnitType] <ScriptingUnitType> [-ShowError] <Boolean>
```

[-UnitType] <ScriptingUnitType>

The Unit Type values are specified in Scripting Unit Type.

[-ShowError] <Boolean>

Whether the error value should be visible.

Example

```
PROFILER-UNITS Percent $false
```

22.2.20 PROFILER-UNSYNC

Disconnects the GUI from the runtime application.

```
PROFILER-UNSYNC
```

Example

```
PROFILER-UNSYNC
```

22.2.21 PROFILER-WAIT

Causes the profiler to wait for the specified number of seconds.

Note: This cmdlet is functionally equivalent to the built-in 'SLEEP' command.

```
PROFILER-WAIT [-Seconds] <Double>
```

[-Seconds] <Double>

The number of seconds to wait. This value can be fractional.

Example

```
PROFILER-WAIT 0.5
```

22.2.22 PROFILER-WAITFOR

Causes the profiler to wait for a specific event before continuing execution.

`PROFILER-WAITFOR [-WaitEvent] <ProfilerWaitEvents>`

[-WaitEvent] <ProfilerWaitEvents>

The Event values are specified in Profiler Wait Events.

Example

```
PROFILER-WAITFOR ReadyToProfile
```

22.2.23 PROFILER-CTR-CORES

Tells the profiler which cores should be recorded in a Sampled profile.

`PROFILER-CTR-CORES <Integer[]>`

<Integer[]>

Space separated list of core numbers to record. If a core number is not present in the list, it is disabled for recording. Each core number to record should be specified only once. The order of the numbers does not matter.

Example

```
PROFILER-CTR-CORES 0 1

[int[]]cores = 0, 1
PROFILER-CTR-CORES @cores
```

22.2.24 PROFILER-CTR-INFERRED

Tells the profiler to attempt to take an **Inferred** profile.

`PROFILER-CTR-INFERRED`

If it is not possible to take an **Inferred** profile, this will instead set the profiler to take a **Callstack** profile.

Example

```
PROFILER-CTR-INFERRED
```

22.3 Argument Values

The values that can be specified in certain cmdlet's arguments are specified below.

22.3.1 Copy Extensions

The Call Tree and Code Coverage tabs have more than one copy functionality.

After each option below, compatible tabs will be listed.

- `All` - Tabs: `CallTree`, `CodeCoverage`. Default value. Used to copy all data on the tab.
- `NotSeen` - Tabs: `CodeCoverage`. Used to copy only the unseen functions list.
- `Seen` - Tabs: `CodeCoverage`. Used to copy only the seen functions list.
- `Selected` - Tabs: `CallTree`. Used to copy only the selected functions from the Call Tree.

22.3.2 Profiler Wait Events

Events that can be waited for in the scripting engine.

- `CanSync`
- `ReadyToProfile`

22.3.3 Rate

Not all rate values can be specified for all Sample Strategies. To see which rates are compatible with which strategies, please use the GUI. If '0' is specified the value will be set to its default recording value for the strategy chosen.

22.3.4 Scripting Perf Counter

The strings used to denote performance counter groups in a script.

These values correspond with the performance counter groups specified in Performance Counter Groups.

- `Disabled`
- `InstructionMissAndBussContention`
- `InstructionICacheEfficiency`
- `MemComputePerf`
- `CacheMemPerf`
- `DCacheReadPerf`
- `DCacheWritePerf`
- `PressureLoadStoreQueue`
- `MispredictedBranch`

22.3.5 Scripting Profile Type

The different types of profiles that can be started.

- `Sampled`
- `Instrumented`

22.3.6 Scripting Sample Strategy

These strings denote sampling strategies in a script.

These values correspond with the strategies presented in Sampling Strategy and Rate Buttons.

- `Time`
- `MissICache`
- `MissDCacheRead`
- `MissDCacheWrite`
- `BranchMispredict`
- `StalledInstruction`

- `StalledDHazard`
- `StalledLSUFull`

22.3.7 Scripting Tabs

A list of the tabs that can be targeted using certain scripting commands.

- `Assembly`
- `CallTree`
- `CodeCoverage`
- `Counters`
- `Functions`
- `Info`
- `Instrumented`
- `Last`
- `Threads`

22.3.8 Scripting Tree Control Actions

A list of the actions that can be performed on trees.

- `CollapseAll`
- `ExpandAll`

22.3.9 Scripting Unit Type

The type of units to display in the profiler.

- `AvgTime`
- `Percent`
- `Samples`
- `Time`

22.3.10 Scripting.SampleGraph.CombinedLineDisplay

Controls how the sample graph shows the combined line.

- `Hide`
- `Show`
- `Only`

22.3.11 Scripting.SampleGraph.NameDisplay

Controls how function and counter names are displayed in the sample graph.

- `Hide`
- `Short`
- `Long`
- `Full`

22.3.12 Scripting.SampleGraph.SampleDisplay

Controls how Callstack samples are displayed.

- `Self`
- `Total`
- `Mixed`

22.3.13 Spike Detection Types

List of Spike Detection options described in Spike Detection.

- `Rare`
- `Burst`
- `Flutter`
- `BadFrame`

22.3.14 Story Types

List of story analysis options described in Story Analysis.

- `Brief`
- `Full`
- `File`
- `Top`

22.4 Other Useful Commands

To maximize the usefulness of the scripting, here are some additional properties, commands, and function calls that may prove useful.

- `[Environment]::CurrentDirectory`

  Gets the current directory of the profiler application. This will usually be the folder from which the EXE was launched.
- `[Nintendo.CPU.Profiler.Scripting.ScriptingCore]::NProfFilename`

  The entire filename of the currently loaded NPROF file. If no NPROF is loaded, or if this is a new profile that has not yet been saved, this will contain an empty string.
- `[Nintendo.CPU.Profiler.Scripting.ScriptingCore]::ScriptDirectory`

  The directory from which the currently executing script was loaded.
- `[Nintendo.CPU.Profiler.Scripting.ScriptingGUI]::ELFDirectory`

  The directory of the ELF that is current synced.

23 Nintendo 3DS CPU Profiler Game API

The Nintendo 3DS CPU Profiler Game API provides a way for a profiled game/application to enhance the information in a profile. Most importantly, a game can mark the beginning of each frame so that the profiler can perform frame-based analysis.

The API is written in C. Using any of these functions will enable the taking of an **Instrumented Profile**.

The API documentation can be found in the supplied man pages.

24 Troubleshooting

The following is a list of known problems and solutions.

24.1 Profiler crashes

In the case of a crash, a log file is saved out in the same location as the Nintendo CPU Profiler executable. When this happens, please send this log file along with any other important information to Nintendo and we will quickly resolve the issue. Please maintain a version of the application that caused the crash (it may be needed to help diagnose the problem).

24.2 Sample Graph draws very slow and has overlapping frame marker

If you accidentally record heartbeats too fast (faster than every 1ms or so), the **Sample Graph** will draw all of them (causing drawing to become very slow). If this occurs, find the error in your code where you are recording them too often.

24.3 Sample Graph does not appear

If the Sample Graph does not appear, but does display an error message, try following the advice of the error message. Details about DirectX requirements can be found in Requirements.

24.4 Sample Graph appears to be a frame behind

This problem has been reported when either the incorrect graphics drivers are in use, or the graphics drivers are out of date. Please ensure that you have the correct graphics drivers installed for the graphics card and your computer, and that the drivers are up-to-date.

24.5 Problem starting profiler on devkit

There are times when the profiler will report back that there was a problem starting the profiler on the devkit. If this happens continuously but the application is starting properly than please try closing all open profiler applications. Re-running this should then allow for proper connection to the dev kit upon the next sync. If the problem continues, try doing this again, but this time also close out of the debugger software and power-cycle the dev kit.

Finally, if you are making a standard application and using a SNAKE dev kit, ensure that the System Mode is not set to dev1. The System Mode for a standard application on a SNAKE dev kit should be set to either dev2 or prod in order to successfully profile.

Note: The profiler operates on a 20 second timeout. If your application is taking longer than this to start please let Nintendo know as we may need to look at increasing the timeout value.

24.6 Profiler starts but the game to profile does not

This issue will typically present itself as continuing to show the Test Menu screen. Please double-check Limitations and Resources to be sure that the requirements and restrictions there are being met.

24.7 Unable to take Instrumented Profile

When instrumentation is not possible, the message "`Instrumented profiling unavailable because necessary API references were not found. Check Manual for more information.`" will be displayed in the Instrumented Function text box. In order to take an **Instrumented Profile**, at least one function from the Nintendo 3DS CPU Profiler Game API must be used. Doing so will include the necessary API references for instrumentation to become possible.

24.8 Receiving error "Unable to locate LoadRun library."

Full error output is "`Unable to locate LoadRun library. Syncing for Nintendo 3DS is not available.`"

Please reinstall the debugger software to fix this problem. The profiler relies on either the environment variable *KMC_PARTNER_CTRS* or *IS_CTR_DIR* being setup and pointing to the correct location in the debugger install folder in order to find the loadrun_ctr.dll. If this file is missing or the environment variable is not set correctly, reinstalling the debugger will restore this environment variable and ensure that the dll is present.

24.9 Profile results don't make sense

If the results don't look realistic, the likely cause is that specified AXF is out-of-date from the CCI currently loaded into Emulation Memory. Make sure that two files are in sync with each other and then **Unsync** and **Sync** again and take another profile.

However, because of the ABI for ARM chips, it is extremely difficult to generate an accurate callstack. The profiler does the best that it can, but it can still have problems (such as UNKNOWN STACK DETAILS). If you run into an excessive number of these issues, please contact your local Nintendo support group so that they can investigate the situation.

24.10 A thread takes more time the higher the sample rate

Due to profiler overhead, threads that go off after a set amount of time will appear to take more and more time the higher the sample rate chosen. This is to be expected. If this is a problem, either reduce the sampling rate or disable the thread when profiling.

24.11 HIO Daemon interfering with PCCOM

Sometimes the HIO-Daemon will interfere with the library's ability to communicate with the dev kit. To fix this problem, restart the HIO-Daemon.

24.12 Function names are not demangled

Please (re)install the ARMCC compiler tools to fix this problem. The profiler depends on certain environment variables which are setup when the ARMCC compiler tools are installed and makes use of a compiler provided library to demangle names. If the compiler tools are not installed on the same computer as the profiler executable, the function names will not be demangled.

25 Known Issues

The following are known issues.

25.1 Intermittent Freezing

If .NET 4.6 is installed, a garbage collection bug within .NET 4.6 causes intermittent freezing of the GUI between 2 to 10 seconds. Visual Studio 2015 and Windows 10 both automatically upgrade your system to .NET 4.6. Microsoft is aware of this bug and has issued a hotfix for Windows 7 https://support.microsoft.com/en-us/kb/3088957, Windows 8 https://support.microsoft.com/en-us/kb/3088955, and Windows 8.1 https://support.microsoft.com/en-us/kb/3088956. The "Cumulative Update for Windows 10 for x64-based Systems (KB3093266)" https://support.microsoft.com/en-us/kb/3093266 will address the problem on Windows 10. Eventually, all of these Microsoft fixes will be available through Windows Update.

25.2 Call Tree cores disappear after Collapse All

In the **Call Tree**, if you expand all, then scroll down a ways, and then collapse all, cores 0 and 1 may disappear from the **Call Tree**. Of those still visible, they will be offset lower than expected with a gap between them and the top of the screen. To resolve the situation, either expand the visible core or use the mouse scroll wheel to attempt to scroll down.

25.3 Functions containing a '$' do not demangle

There is a known limitation in the demangler provided with ARMCC 4.1 and 5.04 that does not allow function names containing a '$' from demangling. As such, these function names will still appear mangled in the profiler.

26 Glossary

Buffer

The memory on the dev kit used to store the profile data.

Call Tree

A tree constructed from individual sample callstacks, showing the calling structure of the game during the time it was profiled. This is slightly different from a call graph, since there are no cycles. In a call tree, the same function can appear multiple times, based on where it was called.

Callstack

The trace of function calls discovered at runtime by walking the stack.

Counter

Data from a CPU counter (performance counters) or any other counter described in Types of Counters.

Heartbeat

A periodic event on a particular core that can be manually tracked by using functions built into the platform's API. A heartbeat is similar to a frame marker, but more generic in meaning since it can represent any type of periodic event. Heartbeat markers can be viewed in the **Sample Graph**.

Icicle Graph

An icicle graph is a hierarchical display of the sampled callstack over time, with the root of the callstack at the top growing downwards.

Jitter

Offsets in the sampling times in order to introduce variance in what sections of code are sampled. Without proper jittering there would be a disproportional number of samples in certain sections of the code due to the fixed sampling rates.

Sample by Performance Counter

This is a way to sample functions using a particular performance counter. For example, you could choose to **Sample by L1 Data Cache Miss** at a rate of **Every 500 misses**. For every 500 misses, the profiler will sample what function your game was currently executing.

Sample by Time

This is a way to sample functions based on the specified rate.

Sampling

Periodically stopping the game and recording the current function or callstack.

Sampling Rate

The rate at which samples are taken.

Sampling Strategy

The strategy used to determine when a sample will be taken.

Self

The time spent exclusively in a function, but not any functions it calls.

Sub

The time spent in the functions called by this function (Sub = Total - Self).

Sync

Connecting with the dev kit and loading the ELF file. Once the profiler is synced, a profile can be taken.

System Classification

Functions are classified as belonging to a particular game system, such as animation, graphics, physics, and so on.

Total

The time spent in a function and all of the functions it calls.

Units

The units used to show how much time a function has taken. This can be either **Percent**, **Time**, or **Samples**.

Unsync

Disconnecting from the dev kit.

27 Revision History

Version	Revision Date	Description
4.05	2015/10/23	Added System Classification Filter to Functions tab. Changed Reverse Call Tree to Partial Call Tree. Added section on System Classification filter to PROFILER-FILTER command. Added section Graph Cores to Sample Graph. Added section System Load to Sample Graph. Removed Quick Filters.
4.04	2015/04/22	Added Last Functions tab. Added Diff feature on Functions tab. Added script commands for new features. Added ability to take Inferred profile. Made installation steps more explicit.
4.03.2	2015/02/13	Renamed the **CTR Back Compat** button to **Forced Compat**.
4.03	2014/10/22	Added script command PROFILER-TREECONTROL. Added details on the Quick Ribbon Bar. Added section on Function Details window operations. Added information on Assembly Tab right-click context menu. Removed option to Run Checkers as this is always performed after a profile is loaded or taken. Renamed 'Debug Output' tab to 'Console' tab. Renamed some options on the Sample Graph right-click context menu. Replaced notices on 'SYSTEM WAITING' with 'System Idle Thread'. Updated Resources section.
4.02.1	2014/08/06	Added configurable checkers. Added Text Format Selection to Debug Output. Added Select Function context menu option to Code Coverage.
4.02.0	2014/07/30	Added Checkers. Added Function Details. Added Stack Depth graph. Added Icicle Selected graph. Added more context menus.

Version	Revision Date	Description
		Added ability to sort on Assembly Tab.
		Added note about using workflows on Nintendo 3DS. Updated operation requirements. Fixed locations where the wrong tools were referenced.
4.01.1	2014/05/06	Added Debug Output Tab documentation.
4.01.0	2014/04/16	Standardized section layouts. Added Inferred Heartbeats section to Info Tab. Updated Cmdlet list. Updated Glossary. Replaced 1st, 2nd, and 3rd buttons with Select Top button. Fixed some locations where the wrong platform details were referenced. Removed Quick Filter listings of regular expressions.
		Removed Known Issue where CCL files did not work with Command Line option --loadelf.
4.00.0	2014/02/28	Switched to new format. Added Heartbeats Tab. Updated Quick Filter buttons.
		Moved the API documentation to its own set of man pages. Removed the ability to record into a ring buffer.
3.01.8	2013/12/17	Removed statement 'and click "Download"' from .NET 4.0 download link. Added Known Issue about CCL files and --loadaxf command line option.
3.01.7	2013/12/13	Updated the link to .NET 4.0 download page.
3.01.6	2013/02/26	Removed the limitation item (Section 1.4) regarding the profiler's need for exclusive control over HIO.
3.01.5	2013/02/19	Updated SDK requirements.
3.01.4	2012/10/29	Updated the firmware requirements.
3.01.3	2012/09/06	Removed '14.4 Devkit locks up while profiling' as this issue referenced a known issue that no longer exists.
3.01.2	2012/08/24	Removed old Known Issues that are no longer issues.
3.01.1	2012/08/23	Added new function reference for selecting which core to profile.
3.01.0	2012/08/14	Revision History moved to the end of the manual. Known Issue about CCL files was removed as it is no longer valid.

Version	Revision Date	Description
		Added section 3.2.5 : Information with regards to the Core Selection. Added a new chapter "Profiling Tips"
3.0.0	2012/05/07	Removed known issue of inability to profile applications that make use of the nn::ro library (e.g. application that make use of DLLs). Updated the known issue in regards to CCL File support on IS-CTR DEBUGGERs. Updated 14.11 to reflect recent changes to dependencies required to demangle function names. Update Section 1.3 to better reflect the DevKits currently in use.
2.01.0	2012/01/24	Added IS-CTR specific instructions. Added known issue regarding IS-CTR and CLL files. Updated 2.3 for loading CCI/CCL files instead of AFX files.
2.00.0	2011/11/30	Updated firmware version for SDK-2_X. Added supported information for SDK-3_X. Removed references to the PCCOM Server. Removed troubleshooting issue dealing with the Profiler and the PCCOM Server. Added known issue of Profiler crash on applications that make use of the DLL library.
1.01.0	2011/10/19	Updated Version. SDK-1_X references removed.
1.00.0	2011/06/22	Updated product name to Nintendo 3DS CPU Profiler. Added section on the Manual button. Fixed error with memory usage in system resources requirements. Added cross-references to Limitations and Troubleshooting sections in the Taking a Profile section.
0.14.2	2011/04/27	Updated notes on the limitations of which functions can be instrumented. Updated "Devkit locks up while profiling" known issue.
0.14.1	2011/03/18	Added more notes on the limitations of which functions can be instrumented. Changed documentation to account for multiple SDK versions. Removed Plotting Game API functions. Removed Known Issue "Access Violation occurs when profiling is stopped".
0.14	2011/02/25	Added section on Reverse Call Trees. Added section on graph rendering time in the Sample Graph.

Version	Revision Date	Description
		Added section on Profiler resource usage. Removed notice of Thread limitation due to SDK limitations.
0.13.1	2011/01/28	Updated information on resizing the buffer. Added Alpha description to Sample Graph section.
0.13.0	2011/01/21	Updated Instrumented tab explanation. Added Instrumented Graph tab explanation. Added CodeBlocks. Added information on usage with Home Menu. Updated information on resizing the buffer. Changed restrictions on function selection for Instrumented Profiles. Removed BufferSize API function.
0.12.1	2010/12/20	Added information on requirements to take Instrumented Profile.
0.12.0	2010/12/09	Added ARM instrumentation. Modified GUI layout. Added section Performance Counters. Updated Troubleshooting and Known Bugs.
0.11.0	2010/11/17	Added Error button explanation to Units description. Added 1st, 2nd, 3rd, 4th button explanations. Removed Top 50 button explanation. Added additional information for filtering of Functions tab. Updated Time button explanation to reflect new behavior. Added mention of vblank red vertical line in Sample Graph. Added frame rate button explanation in Sample Graph. Refined frame rate button explanations in Heatmap mode. Added backup automatic frame marking (not recommended). Changes to the Profiler Game API. Updated Requirements.
0.10.1	2010/10/14	Fixed problems with API declarations. Added Enable/Disable Profiling API functions. Removed references to C++ API.
0.10.0	2010/10/07	Added Heatmap feature description. Added sampling by performance counter feature description. Added Runtime Control API function reference. Added command line reference.

Version	Revision Date	Description
		Updated Sampling Rate description.
0.09.1	2010/09/10	Updated installation steps. Required PARTNER Debugger software now 5.61-091-20100901. Removed bugs and notes about multiple dev kits. Updated causes of UNKNOWN STACK DETAILS. Updated troubleshooting.
0.09.0	2010/09/03	Added Performance Counter Tab explanation. Added section on Limitations. Changed steps to start profiling. Changed Count button to Samples button. Added accuracy warnings to Time explanation.
0.08.0	2010/08/06	Added explanation on how to associate .nprof files with the profiler in installation instructions. Added Thread Selection explanation of Active. Added Top 50 button explanation. Added Threads button and Thread header explanation. Added SYSTEM WAITING explanation. Added mention of new items in Info tab's Profile Statistics. Minor clarification in Filtering section.
0.07.0	2010/07/27	Initial version.