Measure App Performance with Easy Profiler

You can use Easy Profiler ↗ to measure application performance. With You.i Platform, there are two ways to connect to Easy Profiler: directly, using C++, or from the Dev Panel. Coding in C++ is the preferred mechanism for profiling, and is the only way to profile application start up.

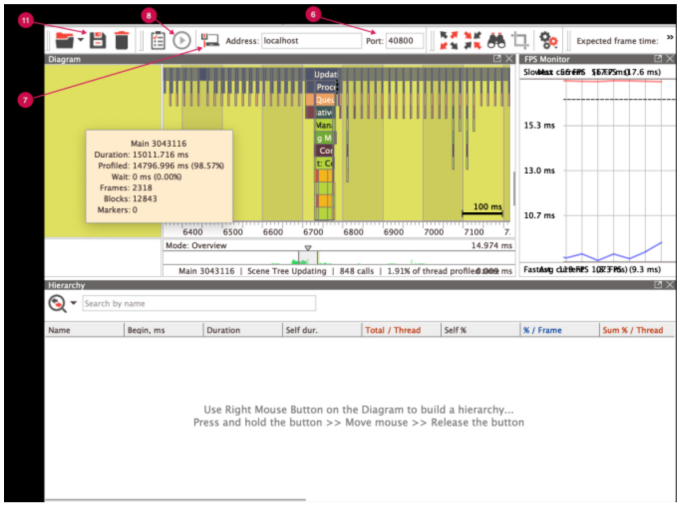
Connect to Easy Profiler and Capture Data via C++

To configure and use Easy Profiler through C++ code changes, do the following:

1. Add the following code to the top of the `App.cpp` file located at **App/src/App.cpp**: `#include <utility/YiEasyProfiler.h>`
2. Add `CYIEasyProfiler::StartListening(port_number);` to the `App::UserInit()` function of the `App.cpp` file as shown below:

```
bool App::UserInit()
{
  CYIEasyProfiler::StartListening(port_number);
}
```

   The default `port_number` for every target platform is 40800, except for LG webOS, where the default is 9930.
3. Save the changes made in `App.cpp`.
4. Generate and build the app using the instructions mentioned in Building Apps and launch the app.
5. Launch the Easy Profiler GUI executable file located at **youiengine/<versionInstalled>/tools/easy_profiler/<DevPlatform>/profiler_gui**. The following is a screenshot of the Easy Profiler GUI:



> **NOTE**
>
> If your app is running on a different system from Easy Profiler, be sure to use the IP address of the target platform instead of `localhost`.

6. Connect to the port using the `port_number` mentioned in `StartListening(port_number)`.
7. Click **Connect** in Easy Profiler.
8. Click the **Capture** icon in the GUI to capture the data.

> PRO TIP
>
> You can also call `CYIEasyProfiler::StartCapture` and `CYIEasyProfiler::StopCapture` in your application to programmatically initiate and end a data capture for a particular event, such as screen transition or button click. For more information, see `StartCapture()` and `StopCapture()` API documentation.

9. Perform the activity on the app that you want the data for.
10. Close the Capture dialog to stop capturing data.
11. Once the data is captured, you can save it as a `.prof` file.

> PRO TIP
>
> You can also use `CYIEasyProfiler::WriteToFile` with a writable path to save the data being captured using Easy Profiler.

Capture Data via React Native

When done performing the steps in Connect to Easy Profile and Capture Data via C++, implement the Systrace ↗ functionality to measure the performance for React Native. We use JavaScript's begin and end events to measure the performance for React Native:

1. Generate the project with the `--dev` option, for example: `youi-tv generate -p osx --dev`.
2. Import Systrace to the `index.youi.js` file. For example: `import { Systrace } from "react-native"`.
3. Call `Systrace.setEnabled(true)` in the code where you want to start measuring the performance in React Native. For example:

```
import React, { Component } from 'react';
import { createStackNavigator } from 'react-navigation';
import { AppRegistry, NativeModules, Systrace } from 'reac
Systrace.setEnabled(true); // Call setEnabled to turn on t

...

AppRegistry.registerComponent('YiReactApp', () => YiReactA
```

Once the above steps are done, you can see the JavaScript's events alongside the C++ events in the `MessageQueue` thread.

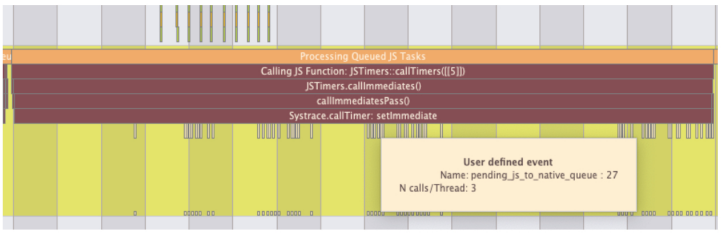
Capturing non-Timed JavaScript Events

Systrace ↗ allows you to mark JavaScript (JS) events with a tag and an integer value Capture the non-Timed JS events in EasyProfiler by adding the following code to the app code in addition to the steps mentioned in Capture Data via React Native:

```
Systrace.counterEvent("event_label", 10);
```

For more information, see Systrace counterEvent ↗ documentation.

The following screen shot illustrates markers for the non-Timed JS events in the EasyProfiler GUI:



Capture Data at Initialization

You can also capture events that take place during the app initialization using the `YI_START_EASY_PROFILER_CAPTURE_AT_INIT` macro You can add the macro at the start of any `.cpp` file, outside of functions, such as `App.cpp` For example:

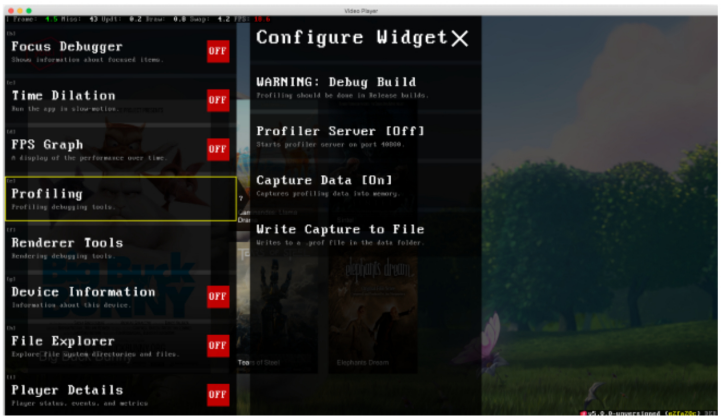
```
#include <utility/YiEasyProfiler.h>

YI_START_EASY_PROFILER_CAPTURE_AT_INIT;

HelloWorldApp::HelloWorldApp()
: m_pSpinAnimation(nullptr)
{

}
```

You can use `CYIEasyProfiler::StopCaptureAfterNextDraw` to trigger `CYIEasyProfiler::StopCapture()` when the app draws the next frame, it is used in conjunction with `YI_START_EASY_PROFILER_CAPTURE_AT_INIT`. The combination of these lets you capture everything up until the first draw, which is the bulk of any app's initialization time.

> NOTE
>
> Don't forget to add the following header file to the .cpp file when you use the `YI_START_EASY_PROFILER_CAPTURE_AT_INIT` macro:
> `#include <utility/YiEasyProfiler.h>`

Profile from the Dev Panel

The Dev Panel has a set of useful widgets to help you troubleshoot and debug your application - including direct access to the Profiler, without needing to modify your application.



As with the C++ method of connecting your app to Easy Profiler, you can launch and connect to a Profiler Server, or you can write profile data to a file and open it with Easy Profiler.

> IMPORTANT
>
> You.i TV recommends you perform profiling with Release builds To enable the Dev Panel for your Release build, add `InitializeDevPanel();` to `UserInit()` in `App.cpp` Remember to remove this when you are done profiling!

Option 1: Use the Profiler Server from the Dev Panel

1. Open the **Dev Panel** and go to **Profiling**.
2. Select **Profiler Server** to start the server.
3. Connect **Easy Profiler** to the server (see above for instructions).
4. Close the **Dev Panel** and exercise the desired app code.
5. When you're done, use the **Easy Profiler GUI** to stop the capture, which also transfers over the capture data.

Option 2: Write to a Profiler file from the Dev Panel

You can capture a file with your profile data that you open with Easy Profiler Note on some target platforms (such as 10ft devices and iOS) it's hard to download generated files Using the Profiler Server is the better choice for those platforms.

1. Open the **Dev Panel** and go to **Profiling**.
2. Select **Capture Data** to start data capture.
3. Close the **Dev Panel** and exercise the desired app code.
4. When you're done, open the **Dev Panel** again and select **Capture Data** to stop capturing.
5. Select **Write Capture to File**.
6. Find the resulting file in your app's **Data** folder and open it in Easy Profiler.

Best Practices for Using Easy Profiler

Remember the following points when you use Easy Profiler as performance tooling:

- You.i TV recommends using Easy Profiler with the `Release` configuration to get the correct data to measure performance of an app running on the target platform.
- Data captured with the `Debug` configuration provides detailed information on function calls through the JavaScript bridge, such as function name and type, which could result in enormous amounts of data consuming a lot of memory on low-end devices, while data captured with the `Release` configuration provides information on the function calls without divulging details about the name of the function or method being called.
- The App should run on the actual target platform to capture correct and valid data. If you are running an app on a target platform such as macOS that provides better performance tooling than Easy Profiler, You.i TV recommends that you use that performance tool instead of Easy Profiler, as it is not a sampling profiler.

> NOTE
>
> You.i TV does not recommend using Easy Profiler on simulators as it will result in unexpected behavior.

Color Legend to Understand Easy Profiler Data

The following table provides information on the color legend used in the Easy Profiler data:

Color Name	Hexadecimal	Associated Events	Notes

Congo Brown	5e3643	Creating Component, Measuring Component, Dispatching View Manager Command, Updating Component, Managing Children, Setting Children	
Porsche (a shade of Orange)	eea160	Processing JS Task, Processing Queued JS Tasks	
Goblin (a shade of green)	397b44	Destroying Registry, Destroying Subtree, Calculating and Applying Layout	
Spicy Mix (a shade of maroon)	7a444a	Initializing Bridge, Loading and Starting Bridge, Calling JS Function	
Antique Brass (a shade of brown)	bf7958	Loading Application Script, Invoking Callback, Calling JS Function, Setting Variable, Handling Memory Pressure	
Mineral Green	3c5956	Calling Native Module	
Sushi (a shade of green)	71aa34	Calling Method	
Matrix (a shade of Red)	a05b53	Expanding Argument	
Conifer (a shade of green)	b6d53c	Provides extra information related to data, type, component name, file name, calling function name	Only available with the Debug configuration