SEGGER SystemView

User Guide

Document: UM08027
Software Version: 2.50
Revision: 0
Date: May 3, 2017

A product of SEGGER Microcontroller GmbH & Co. KG

www.segger.com

Disclaimer

Specifications written in this document are believed to be accurate, but are not guaranteed to be entirely free of error. The information in this manual is subject to change for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, SEGGER Microcontroller GmbH & Co. KG (SEGGER) assumes no responsibility for any errors or omissions. SEGGER makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. SEGGER specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Copyright notice

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH & Co. KG

In den Weiden 11
D-40721 Hilden

Germany

Tel. +49 2103-2878-0
Fax. +49 2103-2878-28
E-mail: support@segger.com
Internet: www.segger.com

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please inform us and we will try to assist you as soon as possible. Contact us for further information on topics or functions that are not yet documented.

Print date: May 3, 2017

Software	Revision	Date	By	Description
2.50	0	170426	JL	• Setction “SystemView PRO” added. • Setction “Event filter” added.
2.42	1	170306	AG	• Chapter “Supported CPUs” updated.
2.42	0	170209	JL	• Section “GUI Controls” updated. • Section “Trigger Modes” added.
2.40	0	160728	JL	• Chapter “Getting started with SEGGER SystemView” updated. • Chapter “API reference” updated.
2.38	0	160624	JL	• Section “Renesas RX” to “Supported Devices” added.
2.36	0	160524	JL	• Section “Getting started with SEGGER SystemView” updated. • Section “Host Application - SystemViewer” updated. • Section “Supported OSes” updated. • Section “Integration guide” updated.
2.34	0	160401	JL	• Section “Optimizing SystemView” added. • Section “uC/OS-III” to “Supported OSes” added.
2.32	1	160322	JL	• Chapter “Performance and resource usage” added.
2.32	0	160310	JL	• Section Supported OSes added. • Post-Mortem mode added. • Chapters restructured by relevance. • Sample configuration for TI AM3350 Cortex-A8 added
2.30	0	160127	JL	• Section Using SystemViewer added. • Section Integrating SEGGER SystemView into an OS added. • Section Integrating SEGGER SystemView into a middleware module added. • Section Frequently asked questions added. • Section Supported CPUs added. • Section SEGGER SystemView API functions updated.
2.28	0	160114	JL	• Terminal Window description added. • Screenshots updated.
2.26	1	160106	JL	Configuration for embOS and FreeRTOS added.
2.26	0	151223	JL	• Printf functionality added.
2.24	0	151216	JL	• OS X and Linux version added.
2.22	0	151214	JL	• GUI and performance improvements.
2.20	1	151119	JL	• Screenshots updated. • Fixed defines in configuration.
2.20	0	151118	JL	• SystemViewer GUI elements restructured. • SystemView Config module added.
2.10	0	151106	JL	• Official Release.
2.09	0	151026	JL	

Software	Revision	Date	By	Description
				• Initial Pre-Release.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler).
- The C programming language.
- The target processor.
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend *The C Programming Language* by Kernighan and Richie (ISBN 0–13–1103628), which describes the standard in C programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

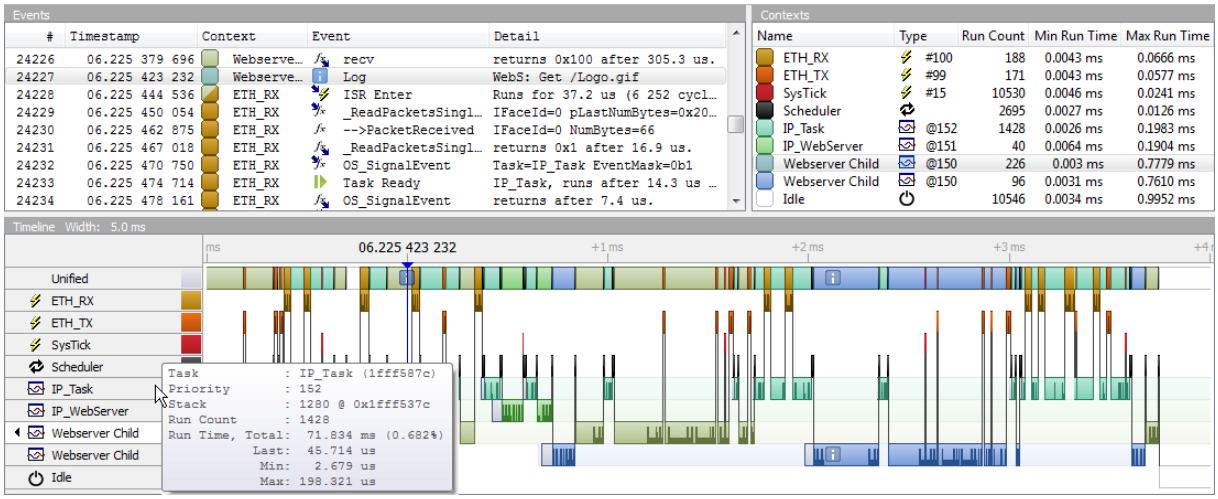
Style	Used for
Body	Body text.
`Keyword`	Text that you enter at the command prompt or that appears on the display (that is system functions, file- or pathnames).
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
`Sample comment`	Comments in program examples.
Reference	Reference to chapters, sections, tables and figures or other documents.
GUIElement	Buttons, dialog boxes, menu names, menu commands.
Emphasis	Very important sections.

Table of contents

Chapter 1

Overview

This section describes SEGGER SystemView in general.



1.1 What is SEGGER SystemView?

SystemView is a toolkit for visual analysis of any embedded system. SystemView gives complete insight into an application, to gain a deep understanding of the runtime behavior, going far beyond what a debugger is offering. This is particularly advantageous when developing and working in complex systems with multiple threads and events.

SystemView consists of two parts:

The PC visualization *SystemView App*, and some code, collecting information on the target system.

The SystemView host application allows analysis and profiling of the behavior of an embedded system. It records the monitor data which is generated in the embedded system and visualizes the information in different windows. The recording can be saved to a file for analysis at a later time or for documentation of the system.

The monitor data is recorded via the debug interface, meaning that no additional hardware (especially no extra pins) is required to use SystemView. It can be used on any system that allows debug access.

With a SEGGER J-Link and its *Real Time Transfer* (RTT) technology SystemView can continuously record data, and analyze and visualize it in real time.

SystemView makes it possible to analyze which interrupts, tasks and software timers have executed, how often, when exactly and how much time they have used. It sheds light on what exactly happened in which order, which interrupt has triggered which task switch, which interrupt and task has called which API function of the underlying RTOS.

Cycle-accurate profiling can be performed and even user functionality can be timed.

SystemView should be used to verify that the embedded system behaves as expected and can be used to find problems and inefficiencies, such as superfluous and spurious interrupts, and unexpected task changes. It can be used with any (RT)OS which is instrumented to call SystemView event functions, but also in systems without an instrumented RTOS or without any RTOS at all, to analyze interrupt execution and to time user functionality like time-critical subroutines.

How does it work?

On the target side a small software module, containing SYSTEMVIEW and RTT, needs to be included. The SYSTEMVIEW module collects and formats the monitor data and passes it to RTT. The RTT module stores the data in the target buffer, which allows continuous recording with a J-Link on supported systems, as well as single-shot recording and post-mortem analysis on any system.

The target system calls SYSTEMVIEW functions in certain situations, such as interrupt start and interrupt end, to monitor events. SystemView stores these events together with a configurable, high-accuracy timestamp, in the RTT target buffer. Timestamps can be as accurate as 1 CPU cycle, which equates to 5 ns on a 200 MHz CPU.

What resources are required on the target side?

The combined ROM size of RTT and the SYSTEMVIEW modules is less than 2 KByte. For typical systems, about 600 bytes of RAM are sufficient for continuous recording with J-Link. For system-triggered recording the buffer size is determined by the time to be recorded and the amount of events. No other hardware is required. The CPU needs less than 1 us for typical events (based on a 200 MHz Cortex-M4 CPU), which results in less than 1% overhead in a system with 10,000 events per second. Since the debug interface (JTAG, SWD, FINE,) is used to transfer the data, no additional pins are required.

On which CPUs can SystemView be used?

SystemView can be used on any CPU. Continuous real-time recording can be carried out on any system supported by J-Link RTT technology. RTT requires the ability to read memory via the debug interface during program execution which is generally supported in ARM Cortex-M0, M0+, M1, M3, M4 processors as well as all Renesas RX devices.

On systems which are not supported by the RTT technology the buffer content can be read manually when the system is halted, which allows single-shot recording until the buffer is filled and post-mortem analysis to capture the latest recorded data. Single-shot and post-mortem recording can be triggered by the system to be able to control when a recording starts and stops.

How much work is it to add it to a target system?

Not very much. A small number of files need to be added to the make file or project. If the operating system supports SystemView, then only one function needs to be called. In a system without RTOS or non-instrumented RTOS, two lines of code need to be added to every interrupt function which should be monitored. That’s all and should not take more than a few minutes.

1.2 The SEGGER SystemView package

The following sections describe how to install the SEGGER SystemView package and its contents.

1.2.1 Download and installation

The SEGGER SystemView package is available for Windows, OS X and Linux as an installer setup and a portable archive.

Download the latest package for your operation system from {https://www.segger.com/systemview.html}.

In order to do live recording the current J-Link Software and Documentation Package needs to be installed. Download and instructions are available at {https://www.segger.com/jlink-software.html}.

1.2.1.1 Windows

Installer

Download the latest setup from {http://www.segger.com/systemview.html} and execute it. The setup wizard guides through the installation.

After installation the package content can be accessed through the Windows *Start* menu or from the file explorer.

Portable zip

Download the latest zip from {http://www.segger.com/systemview.html} and extract it to any directory on the file system.

No installation is required, after extraction the package content can be used directly.

1.2.1.2 OS X

Installation package

Download the latest pkg installer from {http://www.segger.com/systemview.html} and execute it. The package installer guides through the installation.

After installation the SystemView App can be accessed through Launchpad.

1.2.1.3 Linux

Requirements

To run SystemView on Linux the Qt V4.8 libraries have to be installed on the system.

Installer

Download the latest DEB or RPM installer for your Linux from {http://www.segger.com/systemview.html} and execute it. The software installer guides through the installation.

Portable zip

Download the latest archive for your Linux from {http://www.segger.com/systemview.html} and extract it to any directory on the file system.

No installation is required, after extraction the package content can be used directly.

1.2.2 Package content

The SEGGER SystemView package includes everything needed for application tracing — the host PC visualization SystemView App and sample trace files for a quick and easy start.

The target sources to be included in the embedded application can be downloaded as an additional package.

Additional sources to interface with SEGGER software, such as embOS are included for a quick and easy start.

The following table lists the software package content.

SystemView package

File	Description
./SystemView.exe	The SystemView analysis and visualization tool.
./Doc/UM08027_SystemView.pdf	This documentation.
./Description/SYSVIEW_embOS.txt	SystemView API description file for SEGGER embOS.
./Description/SYSVIEW_FreeRTOS.txt	SystemView API description file for FreeRTOS.
./Sample/OS_IP_WebServer.SVDat	SystemView sample trace file of a web server application.
./Sample/OS_Start_LEDBlink.SVDat	SystemView sample trace file of a simple embOS application.
./Sample/uCOS_Start.SVDat	SystemView sample trace file of a simple uC/OS-III application.

Target source package

File	Description
./Src/Config/Global.h	Global data types for SystemView.
./Src/Config/SEGGER_RTT_Conf.h	SEGGER Real Time Transfer (RTT) configuration file.
./Src/Config/SEGGER_SYSVIEW_Conf.h	SEGGER SYSTEMVIEW configuration file.
./Src/Sample/Config/SEGGER_SYSVIEW_Config_embOS.c	Sample initialization of SystemView with embOS.
./Src/Sample/Config/SEGGER_SYSVIEW_Config_embOS_CM0.c	Sample initialization of SystemView for Cortex-M0 targets with embOS.
./Src/Sample/Config/SEGGER_SYSVIEW_Config_embOS_RX.c	Sample initialization of SystemView for RX targets with embOS.
./Src/Sample/Config/SEGGER_SYSVIEW_Config_FreeRTOS.c	Sample initialization of SystemView with FreeRTOS.
./Src/Sample/Config/SEGGER_SYSVIEW_Config_NoOS.c	Sample initialization of SystemView with no OS.
./Src/Sample/Config/SEGGER_SYSVIEW_Config_NoOS_RX.c	Sample initialization of SystemView for RX targets with no OS.
./Src/Sample/Config/SEGGER_SYSVIEW_Config_uCOSIII.c	Sample initialization of SystemView with uC/OS-III.
./Src/Sample/Config/os_cfg_trace.h	Sample configuration for uC/OS-III Trace with SystemView.

File	Description
./Src/Sample/OS/SEGGER_SYSVIEW_embOS.c	Interface between SYSTEMVIEW and embOS.
./Src/Sample/OS/SEGGER_SYSVIEW_embOS.h	Interface header.
./Src/Sample/OS/SEGGER_SYSVIEW_FreeRTOS.c	Interface betweeen SYSTEMVIEW and FreeRTOS.
./Src/Sample/OS/SEGGER_SYSVIEW_FreeRTOS.h	Interface header.
./Src/Sample/OS/os_trace.h	Interface header for uC/OS-III.
./Src/Sample/OS/SEGGER_SYSVIEW_uCOSIII.c	Interface between SYSTEMVIEW and uC/OS-III.
./Src/Sample/Patch/FreeRTOSV8.2.3_Core.patch	FreeRTOS source patch for SystemView.
./Src/SEGGER/SEGGER.h	Global types & general purpose utility functions.
./Src/SEGGER/SEGGER_RTT.c	SEGGER RTT module source.
./Src/SEGGER/SEGGER_RTT.h	SEGGER RTT module header.
./Src/SEGGER/SEGGER_SYSVIEW.c	SEGGER SYSTEMVIEW module source.
./Src/SEGGER/SEGGER_SYSVIEW.h	SEGGER SYSTEMVIEW module header.
./Src/SEGGER/SEGGER_SYSVIEW_ConfDefault.h	SEGGER SYSTEMVIEW configuration fallback.
./Src/SEGGER/SEGGER_SYSVIEW_Int.h	SEGGER SYSTEMVIEW internal header.

1.3 SystemView PRO

SystemView can be used free of charge with any commercial or non-commercial target system. It includes all capabilities to fully analyze system behavior.

SystemView PRO extends these capabilities to provide even better means of system analysis. First it lifts the 1 million event limit and enables unlimited recording. Additionally it comes with new features, such as custom filters to easily search for events in the list.

1.3.1 Licensing

SystemView PRO licenses are available as single-user licenses.

The license can be stored on a J-Link, which is then acting like a USB dongle. This allows use of SystemView PRO on any of your computers, e.g. your PC at work and your notebook at home, by simply plugging in your J-Link.

A license can also be locked to one computer, to enable use of SystemView PRO, even when no J-Link is connected.

For more information about SystemView PRO and its licensing options, feel fre to contact us at info@segger.com.

Chapter 2

Getting started with SEGGER SystemView

This section describes how to get started with SEGGER SystemView. It explains how to analyze an application based on monitored data.

This chapter refers to the sample data file `OS_IP_WebServer.SVDat` which is part of the SEGGER SystemView package.

The sample data file shows the behavior of a target system running the embOS RTOS, the embOS/IP TCP/IP stack and a web server application.

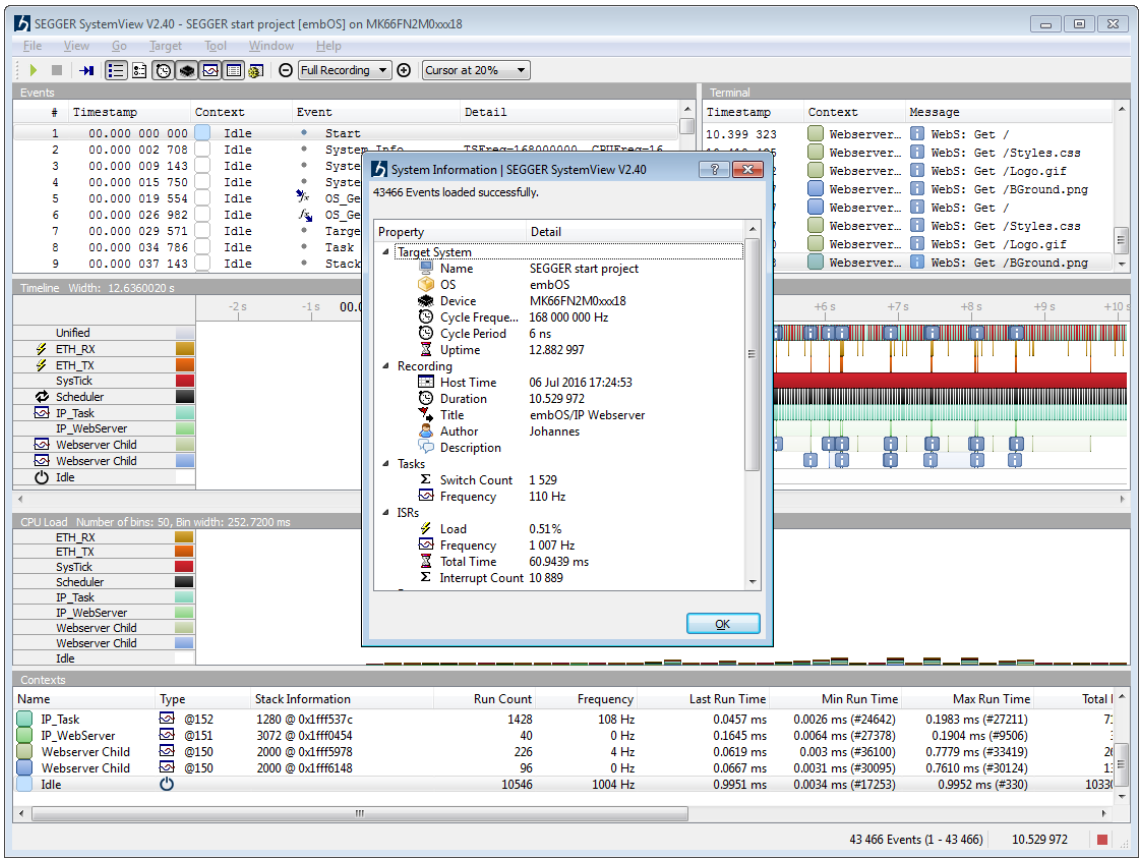
We are going to analyze what the application is doing with the information from SEGGER SystemView.

2.1 Starting SystemView and loading data

SystemView can monitor data live from the target application. The monitored data can be saved to a file for later work with it. Saved data can analyzed without a J-Link and even without the target hardware or the target application. This allows analysis of the system by developers who do not have physical access to it.

- Start the SystemView App (`SystemView.exe`) from the Windows *Start* menu or the installation directory.
- On the first start of SystemView it will prompt to open the sample recording. Click **Yes**.
- On further starts select **File →Sample Recordings → $PackageInstallationDir$/Sample/`OS_IP_WebServer`.SVdat**.

SystemView loads and analyzes the data, shows the system information of the loaded recording, and should now look like this:

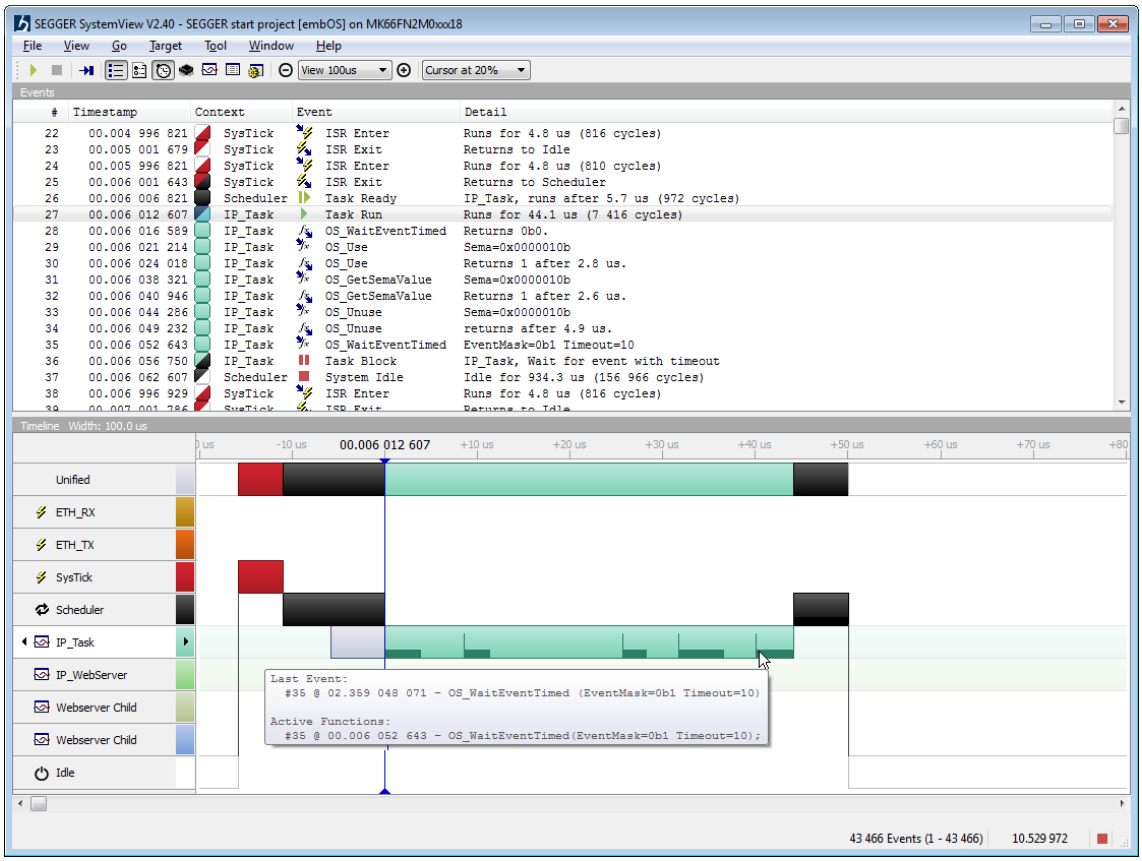


2.2 A first look at the system

We will take a first look at the data to get some information about the monitored system.

System Information

The System Information dialog, shown after loading the data, provides a first overview on the recording. It displays information about the target system, the recording and statistical information of tasks, interrupts, and events. The system information is sent by the application, therefore SystemView does not require any additional configuration to analyze and display the system behavior.



Timeline

The *Timeline* window shows the complete monitored data. In the *Events* list, scroll to the first item to get started.

The *Timeline* window visualizes the system activity by *context* (task, interrupt, scheduler and idle) over the system time. Each row refers to one context item and we can see all items which have been used in the application while it has been monitored.

At the beginning we can see that there are two tasks, `IP_Task` and `IP_WebServer`, indicated by the light background in the context row.

Zoom in to a timeline width of 2.0 ms and double-click on the vertical line below '+1000 us' to center and select the item. (Use the mouse wheel, the toolbar items, [Ctrl] + [+]/[-] keys, or **View →Zoom In**, **View →Zoom Out** to zoom.)

There is some system activity every millisecond from the SysTick interrupt.

Move the mouse over a context name to get more information about the context type and run time information.

Click on the right arrow button of the `IP_Task` context to jump to its next execution.

Zoom in or out to show the activity in detail.

We can see the SysTick interrupt returned to the OS Scheduler, which makes the `IP_Task` ready, indicated by the grey bar in the `IP_Task`'s row, and lets it run. The `IP_Task` returns from the embOS API function `OS_WaitEventTimed` with return value 0, which indicates that no event has been signaled in time.

The `IP_Task` calls three other embOS API functions which quickly return and `OS_WaitEventTimed`, which activates the scheduler, deactivates the task, and puts the system into idle. `IP_Task` will be activated again when the event (EventMask = 1) occurs or after the timeout of 10 ms.

Recorded function calls are visualized in the timeline as small bars in the context row. The vertical peak line indicates the call of a function, the bar shows the length of the call. Stacked bars visualize nested function calls.

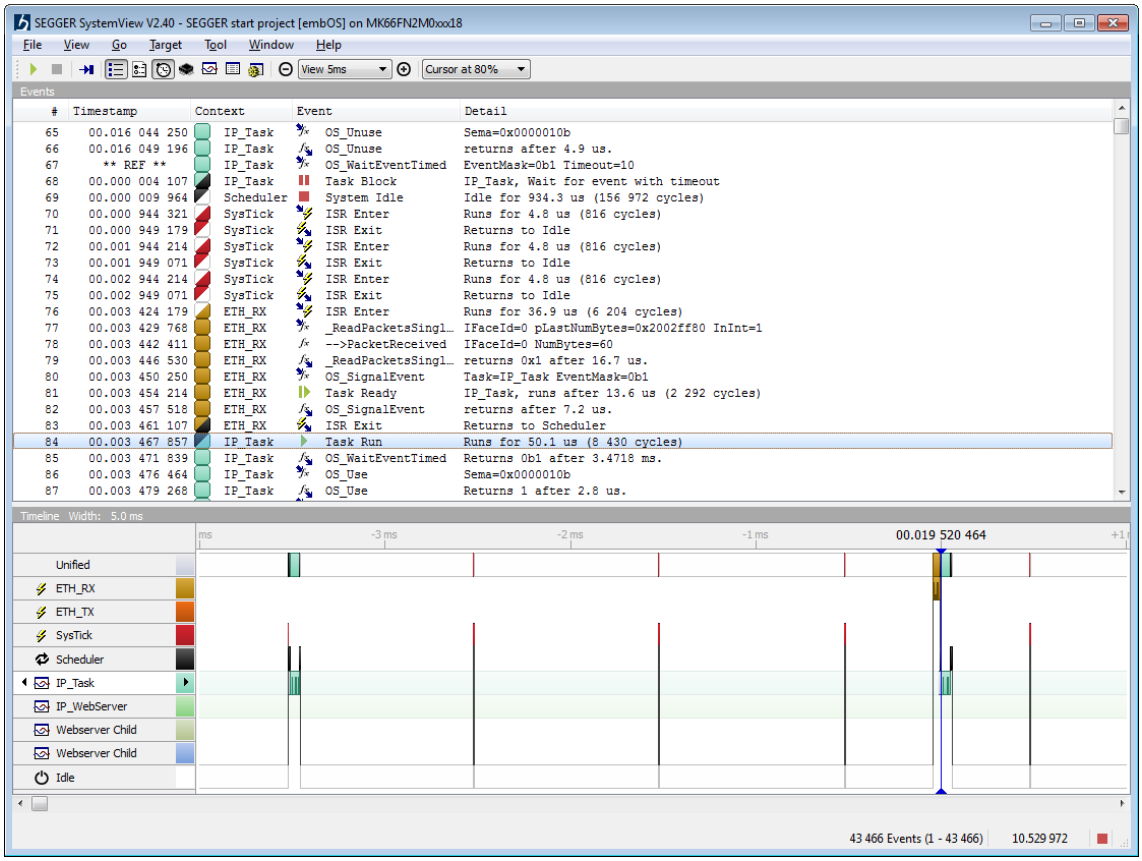
Move the mouse over the context activity to get more information about context runtime, events and function calls.

Conclusion

We have got some first information about the monitored system. From the Timeline we know which tasks and interrupts are used by the application, that it is controlled by the 1 kHz SysTick interrupt, and the `IP_Task` is activated at least every 10 ms.

2.3 Analysing system activity

After getting some information of the system we will analyze how the system is activated.



Events list

The Events list shows all events as they are sent from the system and displays their information, including timestamp of the event, active context, type of event and event details. It is synchronized with the Timeline.

We have seen that every millisecond the SysTick ISR enters and exits and that it activates the `IP_Task` every 10 ms because its timeout occurred.

Go to event #67 with **View →Go to Event...** (Keyboard shortcut: **Ctrl+G**). It is a call of `OS_WaitEventTimed` with a timeout of 10 ms from the `IP_Task` at 00.016 052 607. The timeout would happen at 00.026 052 607.

Set a time reference on the event (**View →Events →Toggle Reference**, Right-Click → **Toggle Reference**, or (Keyboard shortcut **R**). All following timestamps in the events list are measured from the latest reference.

To now see whether the `IP_Task` runs because of the timeout or because of the event it waits for, go to the next activity of `IP_Task` with **Go →Forward** (Keyboard shortcur: **F**).

The timestamp is 00.003 467 857, so 3 ms after the last reference and clearly before the 10 ms timeout. So the task has been activated by the event it waited for.

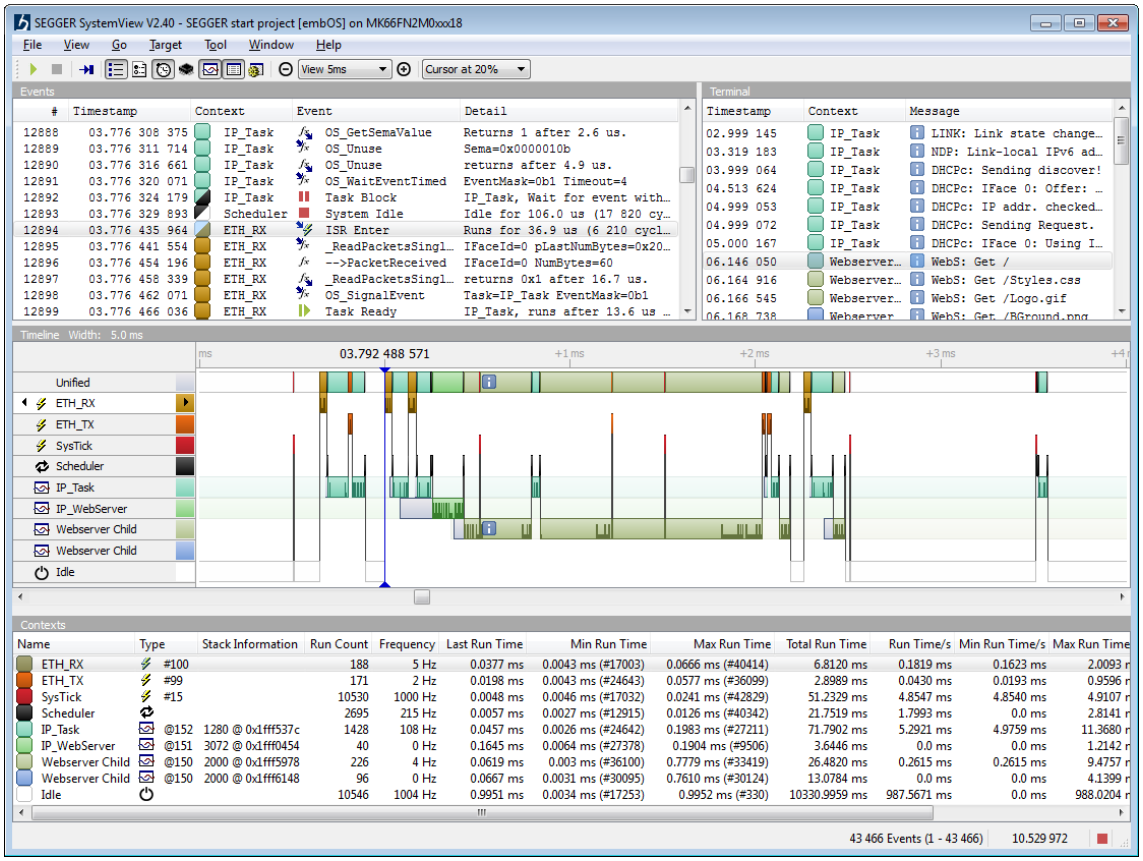
We can see the `ETH_Rx` interrupt happened before. We received a packet via ethernet (60 Bytes on interface 0). Therefore the `ETH_Rx` interrupt signaled the event, which marked the task as ready as indicated in the timeline. The `ETH_Rx` interrupt returns to the Scheduler. `IP_Task` runs and returns from `OS_WaitEventTimed` with return value 0b1, indicating that this event happened.

Conclusion

Going further through the events, we can see that the `IP_Task` is activated after the 10 ms timeout occurred or after we received something and the `ETH_Rx` interrupt occurred.

2.4 Further analysis of the application core

We now know that the system is mainly controlled by the ETH_Rx interrupt. The next step is to see what the system does when it is more active.



Timeline, Events list, Terminal and Contexts window

The windows of SystemView are synchronized and provide the best possibilities for system analysis when used together.

The application creates a web server which can be accessed by the browser to view the embOS/IP demo web page. The sample data has been gathered while the web server was running and the browser loaded the web page multiple times.

Log output has also been sent through SystemView and is displayed in the Terminal window along with the timestamp it has been logged and the active context.

Select a message in the Terminal to also select it in the Events window and the Timeline. The Timeline also indicates all Terminal output.

Go through the messages to see the system initialization when the Ethernet connection is established and select “WebS: Get /”, which is the request from the browser to get the root index webpage.

Go to event #12894, right before the message for detailed analysis.

Here we see that an ETH_Rx interrupt occurred, which calls the embOS/IP function _ReadPacketsSingleIF and receives the packet. Upon reception the embOS event is signaled as seen before, and the interrupt exits into the scheduler which activates the IP_Task.

The IP_Task sets the system event which signals the IP_WebServer Task to become ready. Another packet is received immediately and handled by the IP_Task. When IP_WebServer starts running it is in accept() which calls some OS functions and then returns. It then checks if the Webserver Child task exists and creates it since it did not.

On creation of the task it is added to the contexts and marked with a light background in the timeline while it is not active.

IP_WebServer waits for another connection in accept() and the Webserver Child handles the received http request and serves the webpage. While Webserver Child is active, it may be interrupted by other ETH_Rx interrupts, which cause a preemptive task switch to the IP_Task, because the IP_Task has a higher priority than the Webserver Child.

Note: Tasks are ordered by priority in the Timeline, the exact task priority can be seen in the Contexts window.

2.4.1 Analysis conclusion

We analyzed what a system does without insight into the application code. With the application source we can check with SEGGER SystemView that the system does what it is supposed to do.

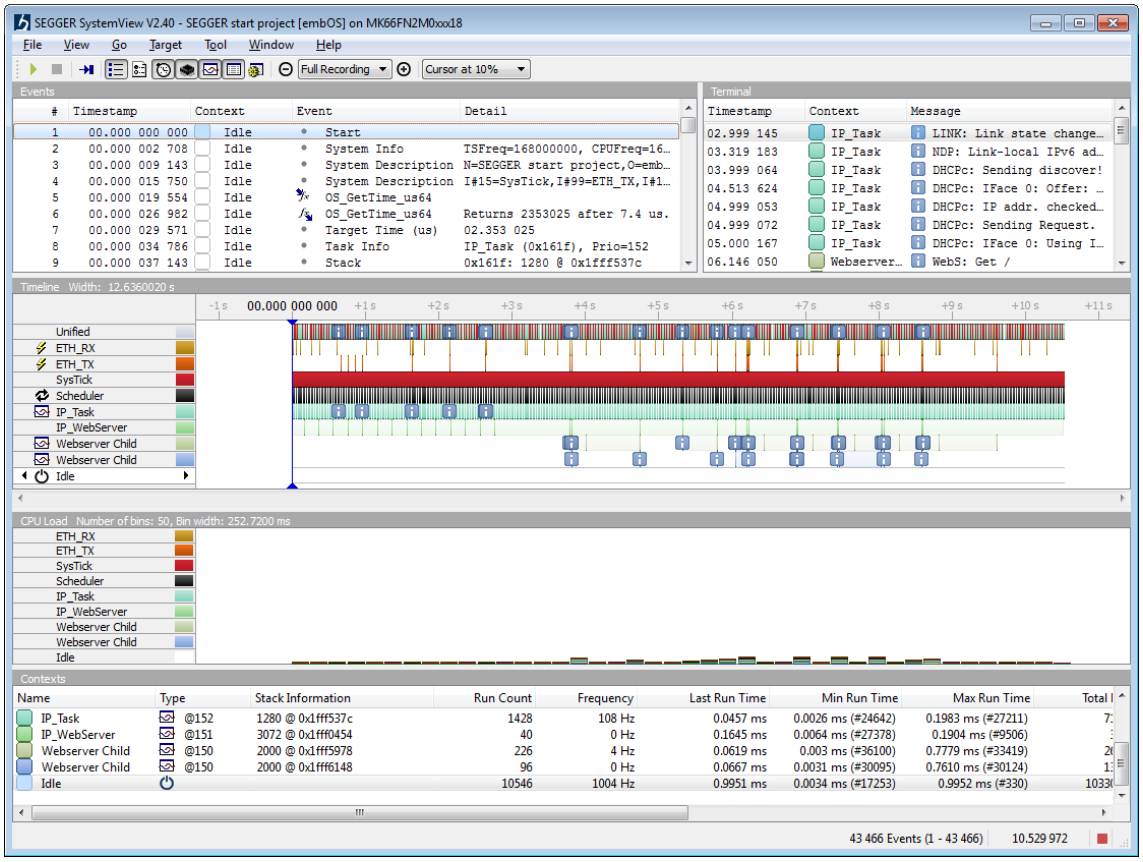
SEGGER SystemView can actively help developing applications, since it not only shows what the system does, but also allows exact time measurement and visualizes the influence of interrupts and events on the application flow. This provides advanced possibilities to find problems and to improve the system.

Chapter 3

Host application - SystemView App

This section describes the SystemView analysis and visualization tool.

3.1 Introduction



The SystemView App is the host PC visualization tool for SEGGER SystemView. It connects via a J-Link to the target application, controls the application tracing and reads its data. The monitored data is analyzed on runtime and visualized in the different windows of SystemView. After tracing has stopped, the data can be saved to a file which allows later analysis of the application trace.

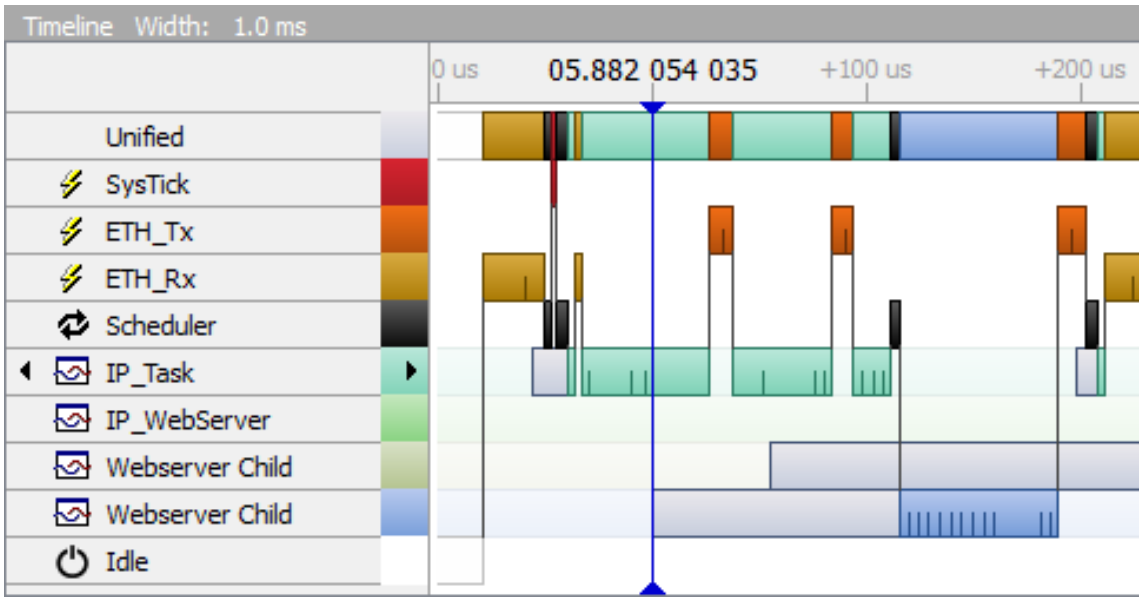
To get started with SystemView please refer to the previous chapter.

SystemView provides different windows to visualize the execution in the system, measure timing and analyze the CPU load. All windows are synchronized to always get all information of the currently selected state.

For a description of the application windows please refer to the following sections.

SystemView allows going through the monitored data and keeping track of what happened in the system at any time.

3.2 Timeline window



The *Timeline* window gathers all system information within one view. It shows the system activity by *context* (task, interrupt, scheduler, timer and idle) over the system time. Each row refers to one context item to show all context items which have been used in the application while it has been monitored.

A mouse-over tooltip on the context items reveals more details and run time information about the context.

A mouse-over tooltip on active context shows the details of the current event and the currently active functions if available.

A ruler on the context marks the current activity span.

An existing task is marked with a light background from its first occurrence to its termination to provide a quick overview which tasks exist at any time.

Switches between contexts are displayed as connection lines to easily identify which events cause context switches and when they occurred.

Tasks which are marked ready for execution are displayed with a light grey bar until their execution starts.

Contexts are ordered by priority. The first row displays all activity in a unified context. Interrupts are top of the list, ordered by Id. Followed by the Scheduler and software timers, if they are used in the system. Below the Scheduler (and timer) the tasks are ordered by priority. The bottom context displays idle time, when no other context is active.

The Timeline is synchronized with the Events list. The event under the cursor (the blue line) is the selected event in the list.

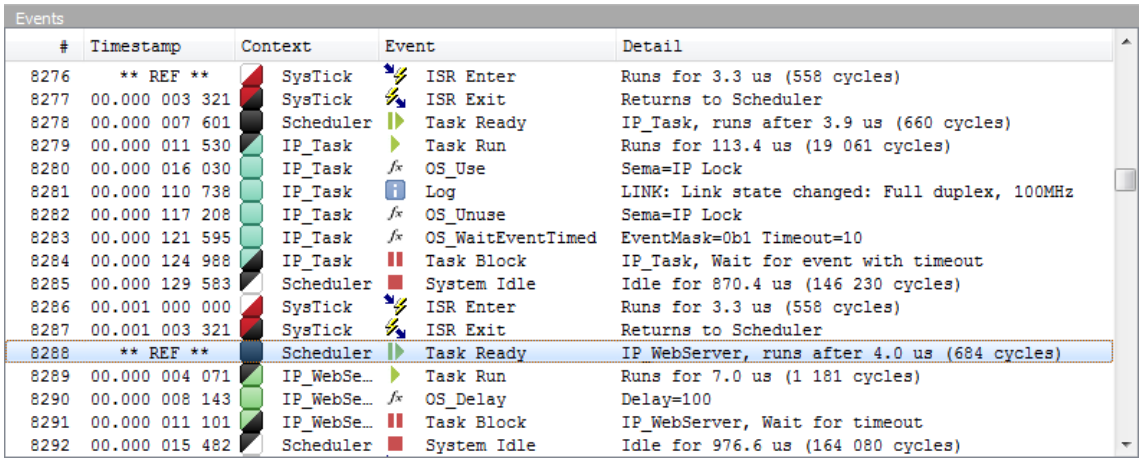
The cursor can be fixed at 10% to 90% of the window and update the selection in the list when scrolling through the timeline.

An event can be dragged under the cursor to select the corresponding event in the events list and vice-versa.

To get an overview of the whole system or to see the exact duration of an event the Timeline view can be zoomed in or out.

To jump to the next or previous activity of a context, the labels include left and right buttons on mouse-over.

3.3 Events window


Events

#	Timestamp	Context	Event	Detail
8276	** REF **	SysTick	ISR Enter	Runs for 3.3 us (558 cycles)
8277	00.000 003 321	SysTick	ISR Exit	Returns to Scheduler
8278	00.000 007 601	Scheduler	Task Ready	IP_Task, runs after 3.9 us (660 cycles)
8279	00.000 011 530	IP_Task	Task Run	Runs for 113.4 us (19 061 cycles)
8280	00.000 016 030	IP_Task	OS_Use	Sema=IP Lock
8281	00.000 110 738	IP_Task	Log	LINK: Link state changed: Full duplex, 100MHz
8282	00.000 117 208	IP_Task	OS_Unuse	Sema=IP Lock
8283	00.000 121 595	IP_Task	OS_WaitEventTimed	EventMask=0b1 Timeout=10
8284	00.000 124 988	IP_Task	Task Block	IP_Task, Wait for event with timeout
8285	00.000 129 583	Scheduler	System Idle	Idle for 870.4 us (146 230 cycles)
8286	00.001 000 000	SysTick	ISR Enter	Runs for 3.3 us (558 cycles)
8287	00.001 003 321	SysTick	ISR Exit	Returns to Scheduler
8288	** REF **	Scheduler	Task Ready	IP_WebServer, runs after 4.0 us (684 cycles)
8289	00.000 004 071	IP_WebSe…	Task Run	Runs for 7.0 us (1 181 cycles)
8290	00.000 008 143	IP_WebSe…	OS_Delay	Delay=100
8291	00.000 011 101	IP_WebSe…	Task Block	IP_WebServer, Wait for timeout
8292	00.000 015 482	Scheduler	System Idle	Idle for 976.6 us (164 080 cycles)

The Events window shows all events as they are sent by the system and displays their information. Every event has the following items:

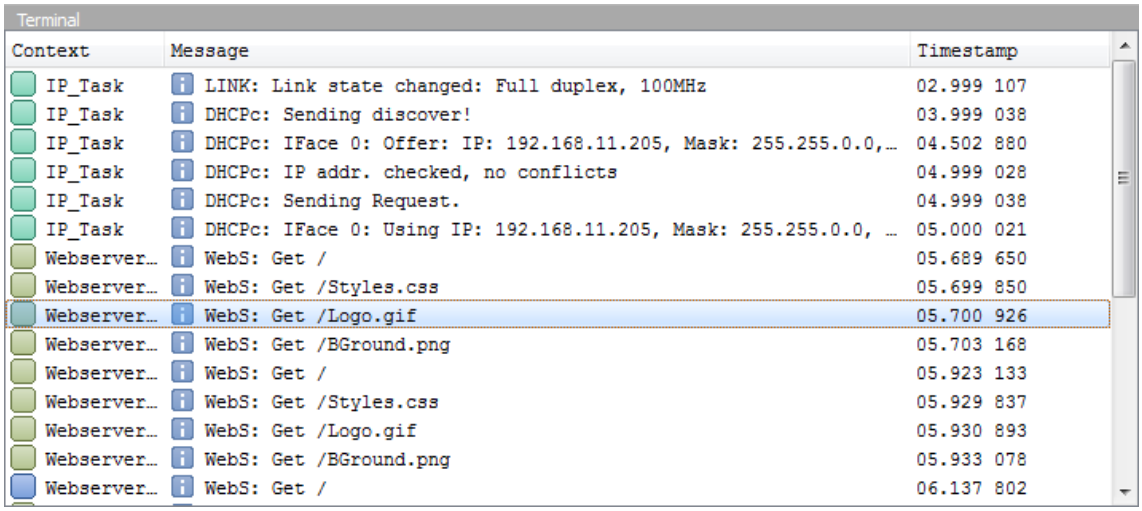
- A timestamp in target time or recording time, which can be displayed with microsecond or nanosecond resolution.
- A context from which it has been created, i.e. the task which is running.
- An event description, displayed with the type of event (IRS enter and exit, task activity, API call).
- Event details describing the parameters of the event, i.e. the API call parameters.
- An ID to locate events in the list.

The Events window allows going through the list, jumping to the next or previous context, or to the next or previous similar event. The Timeline and CPU Load windows are synchronized to show the currently selected event.

The timestamp in the events list can be relative to the start of recording or the target system time if it has been sent by the system. Events can be set as time reference for following events to allow easy measurement of when an event occurred after another one.

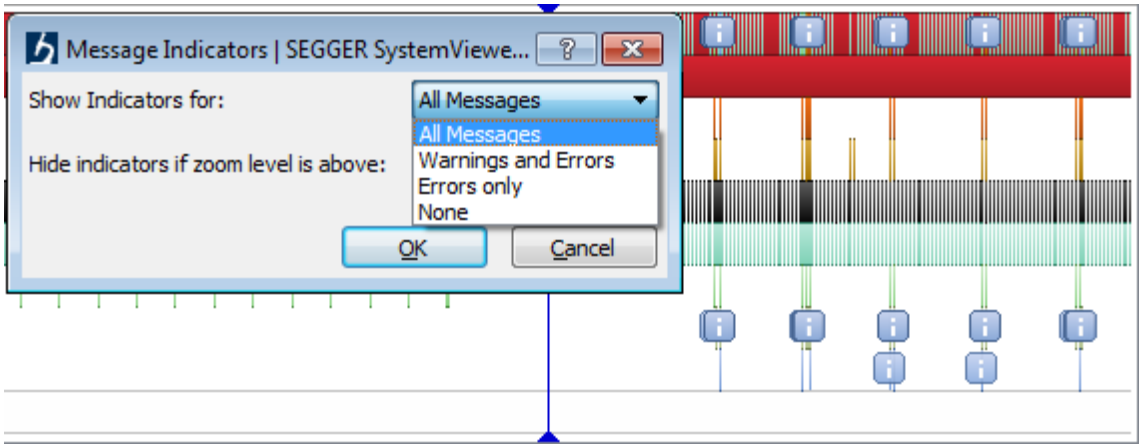
SystemView include an event filter to select show or hide APIs, ISRs, System Information, Messages, Tasks, and User Events.

3.4 Terminal Window



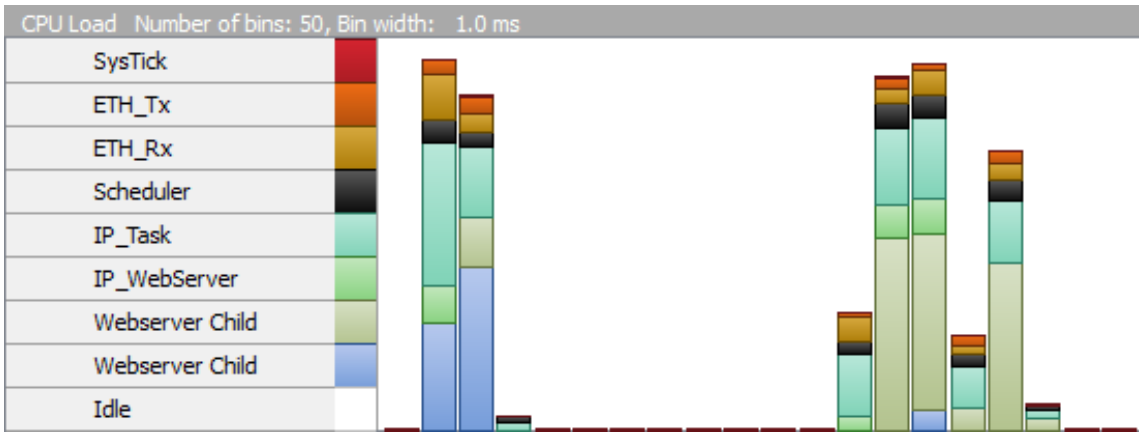
The *Terminal* window shows printf output from the target application alongside the task context from which the output has been sent and the timestamp when the message has been sent. Double-click on a message to show it with all information in the events list.

The Timeline window shows indicators for output, ordered by level - Errors are always on top.The log level for which indicators are shown can be configured via **View → Message Indicators...**.



SystemView printf output can be sent formatted by the application or unformatted with all parameters and is formatted by the SystemView App.

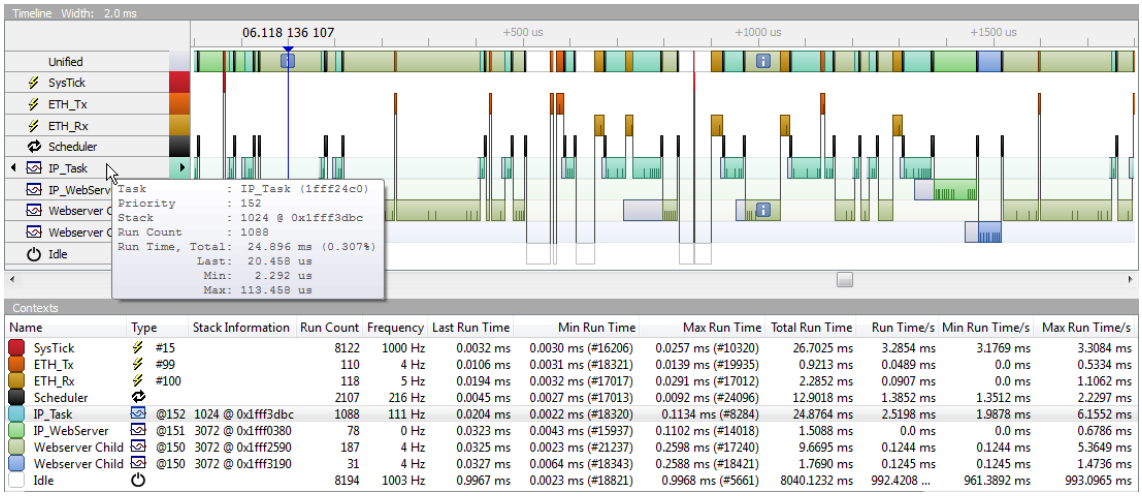
3.5 CPU Load window



The CPU Load window displays the used CPU time of contexts for a period. The CPU Load is measured over the width of a bin with the current resolution of the Timeline and is therefore synchronized with the zoom level.

The number of bins can be selected to measure the load over a shorter or longer period. With a single bin the CPU load is measured over the whole visible Timeline.

3.6 Contexts window



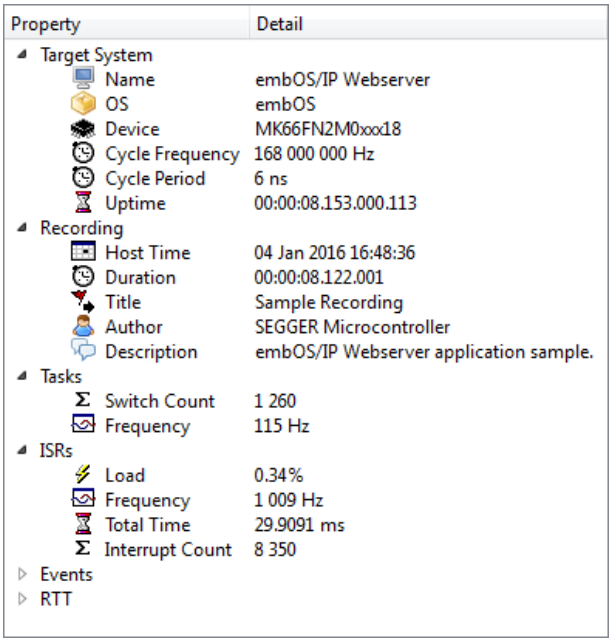
The *Contexts* window shows statistical information of the contexts (Tasks, Interrupts, Scheduler, and Idle). Each context item can be identified by its Name and Type. The Type includes the priority for tasks and the ID for interrupts. (i.e. The Cortex-M SysTick is interrupt ID #15.)

The Contexts window information include following items:

- The context name and type.
- Stack information for tasks, if available.
- Activation count of the context.
- Activation frequency.
- The total and last run-time.
- The current, minimum and maximum run time per second in ms and %.

The Contexts window is updated while the recording, the current context is indicated by selection of the row.

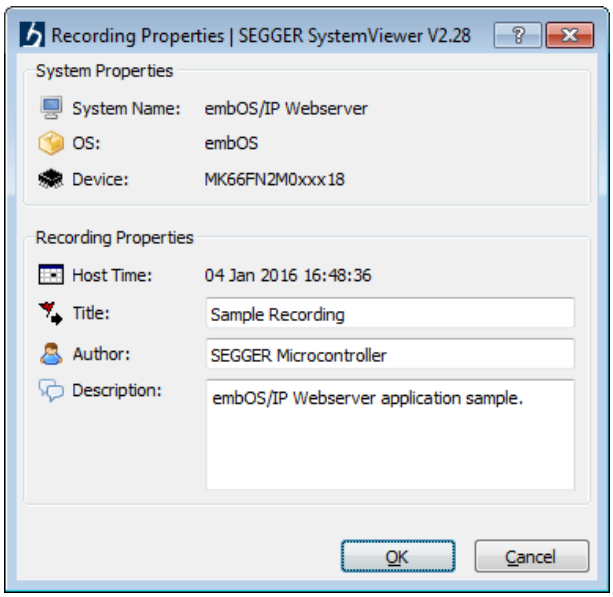
3.7 System information



The window on the top right displays:

- Information about the system, which has been sent by the application to identify it.
- Recording properties, which can be set by the user.
- Statistics about tasks, interrupts, SystemView events and recording throughput.

The System information includes the application name, the used OS, the target device, and the timing information. Additional information about task switches and interrupt frequency provides a quick overview of the system.



The recording properties can be set by the user to be stored with the record when it is saved and allow identifying a record when it is loaded for later analysis.

3.8 Event Filter

The Events window features filtering of events. This can be useful for example to hide interrupt events or to show only task execution.

In SystemView events can be filtered by different groups:

- APIs - OS or module generated events.
- ISRs - Interrupt enter and exit.
- Messages - Terminal Output.
- System Events - System and Task information.
- Tasks - Task execution.
- User Events - User event start and stop.

SystemView PRO additionally features custom filters, which allows selection of any event to be displayed or hidden. With curstom filters all system events and registerd OS and middleware events can be selected individually.

3.9 Trigger Modes

While SystemView is recording in continuous mode, the recorded events are analyzed and displayed in real time. The Trigger Modes enable automatic selection of specific events when they occur.

The Trigger Mode can be seleced in the toolbar or in the right-click context menu of the tasks and interrupts in the Timeline Window.

In Manual Scroll Mode, the selection is not automatically updated and the user can scroll through the events and analyze the system while it is recorded.

In Auto Scroll Mode, the selection is synced to a 100 ms timesamp. The event at the last recorded 100 ms timestamp is selected.

In Continuous Trigger Mode, the user can configure on which context (Taks or Interrupt) and which event to trigger. SystemView then always selects the last occurence of an event which satisfies the configured condition.

In Single Trigger Mode, SystemView selects an event which satisfies the configured condition only once and switches back to Manual Scroll Mode.

3.10 GUI controls

SystemView can be controlled with mouse and keyboard, and via menus. The most important controls are also accessible in the toolbar.

The following table describes the controls of SystemView.

Action	Menu	Shortcut
Recording data		
Start recording on the target.	**Target →Start Recording**	**F5**
Stop recording.	**Target →Stop Recording**	**F9**
Read post-mortem or single-shot data from the system.	**Target →Read Recorded Data**	**Ctrl+F5**
Save recorded data to a file.	**File →Save Data**	**Ctrl+S**, **F2**
Load a record file.	**File →Load Data**	**Ctrl+O**, **F3**
Load a recently used file.	**File →Recent Files**	**none**
Load a sample recording.	**File →Sample Recordings**	**none**
View/Edit recording properties.	**File →Recording Properties...**	**Ctrl+Shift+R**
View		
Set/clear the current event as time reference.	**View →Toggle Reference**	**R**
Remove all time references.	**View →Clear References**	**Ctrl+Shift+R**
Display timestamps in application target time.	**View →Display Target Time**	**None**
Display timestamps in time since start of recording.	**View →Display Recording Time**	**None**
Set the timestamp resolution to 1 us.	**View →Resolution: 1 us**	**None**
Set the timestamp resolution to 100 ns.	**View →Resolution: 100 ns**	**None**
Set the timestamp resolution to 10 ns.	**View →Resolution: 10 ns**	**None**
Set the timestamp resolution to 1 ns.	**View →Resolution: 1 ns**	**None**
Zoom in.	**View →Zoom →Zoom In**	**Ctrl++**, **Scroll up**
Zoom out.	**View →Zoom →Zoom Out**	**Ctrl+-**, **Scroll down**
Show 10 us across the timeline.	**View →Zoom →10us Window**	**None**
Show 20 us across the timeline.	**View →Zoom →20us Window**	**None**
Show 50 us across the timeline.	**View →Zoom →50us Window**	**None**
Show 100 us across the timeline.	**View →Zoom →100us Window**	**None**
Show 200 us across the timeline.	**View →Zoom →200us Window**	**None**
Show 500 us across the timeline.	**View →Zoom →500us Window**	**None**
Show 1 ms across the timeline.	**View →Zoom →1ms Window**	**None**
Show 2 ms across the timeline.	**View →Zoom →2ms Window**	**None**
Show 5 ms across the timeline.	**View →Zoom →5ms Window**	**None**
Show 10 ms across the timeline.	**View →Zoom →10ms Window**	**None**
Show 20 ms across the timeline.	**View →Zoom →20ms Window**	**None**

Action	Menu	Shortcut
Show 50 ms across the timeline.	**View →Zoom →50ms Window**	**None**
Show 100 ms across the timeline.	**View →Zoom →100ms Window**	**None**
Show 200 ms across the timeline.	**View →Zoom →200ms Window**	**None**
Show 500 ms across the timeline.	**View →Zoom →500ms Window**	**None**
Show 1 s across the timeline.	**View →Zoom →1s Window**	**None**
Show 2 s across the timeline.	**View →Zoom →2s Window**	**None**
Show 5 s across the timeline.	**View →Zoom →5s Window**	**None**
Show 10 s across the timeline.	**View →Zoom →10s Window**	**None**
Show 30 s across the timeline.	**View →Zoom →30s Window**	**None**
Show 60 s across the timeline.	**View →Zoom →60s Window**	**None**
Show the full recording in the timeline.	**View →Zoom →Full Recording**	**None**
Set the cursor to 10% of the timeline.	**View →Cursor →Cursor at 10%**	**1**
Set the cursor to 20% of the timeline.	**View →Cursor →Cursor at 20%**	**2**
Set the cursor to 30% of the timeline.	**View →Cursor →Cursor at 30%**	**3**
Set the cursor to 40% of the timeline.	**View →Cursor →Cursor at 40%**	**4**
Set the cursor to 50% of the timeline.	**View →Cursor →Cursor at 50%**	**5**
Set the cursor to 60% of the timeline.	**View →Cursor →Cursor at 60%**	**6**
Set the cursor to 70% of the timeline.	**View →Cursor →Cursor at 70%**	**7**
Set the cursor to 80% of the timeline.	**View →Cursor →Cursor at 80%**	**8**
Set the cursor to 90% of the timeline.	**View →Cursor →Cursor at 90%**	**9**
Measure the visible CPU load in a single bin.	**View →CPU Load →Single Bin**	**None**
Measure the visible CPU load in 10 bins.	**View →CPU Load →10 Bins**	**None**
Measure the visible CPU load in 50 bins.	**View →CPU Load →50 Bins**	**None**
Measure the visible CPU load in 100 bins.	**View →CPU Load →100 Bins**	**None**
Measure the visible CPU load in 200 bins.	**View →CPU Load →200 Bins**	**None**
Show/Hide API calls in the events list.	**View →Event Filter →Show APIs**	**Shift+A**
Show/Hide ISR Enter/Exit in the events list.	**View →Event Filter →Show ISRs**	**Shift+I**
Show/Hide Messages in the events list.	**View →Event Filter →Show Messages**	**Shift+M**
Show/Hide System events in events list.	**View →Event Filter →Show System Events**	**Shift+S**

Action	Menu	Shortcut
Show/Hide Task activity in events list.	**View →Event Filter →Show Tasks**	**Shift+T**
Show/Hide User events in events list.	**View →Event Filter →Show User Events**	**Shift+U**
Show only API calls in the events list.	**View →Event Filter →Show APIs only**	**Ctrl+Shift+A**
Show only ISR Enter/Exit in the events list.	**View →Event Filter →Show ISRs only**	**Ctrl+Shift+I**
Show only Messages in the events list.	**View →Event Filter →Show Messages only**	**Ctrl+Shift +M**
Show only System events in events list.	**View →Event Filter →Show System Events only**	**Ctrl+Shift+S**
Show only Task activity in events list.	**View →Event Filter →Show Tasks only**	**Ctrl+Shift+T**
Show only User events in events list.	**View →Event Filter →Show User Events only**	**Ctrl+Shift+U**
Reset all event filters.	**View →Event Filter →Reset all Filters**	**Ctrl+Shift +Space**
Toggle automatic scroll to last item on new events.	**View →Auto Scroll**	**None**
Select the message indicators to be shown in the timeline.	**View →Message Indicators...**	**Ctrl+M**
Manually scroll the events while recording.	**Toolbar**	**Ctrl+1**
Automatically scroll the events while recording.	**Toolbar**	**Ctrl+2**
Continuously trigger and scroll on a condition while recording.	**Toolbar**	**Ctrl+3**
Trigger and scroll on a condition once while recording.	**Toolbar**	**Ctrl+4**
Configure the trigger condition.	**Toolbar**	**Ctrl+T**
Go		
Jump to the next context switch.	**Go →Forward**	**F**
Jump to the previous context switch.	**Go →Back**	**B**
Jump to the next similar event.	**Go →Next [Event]**	**N**
Jump to the previous similar event.	**Go →Previous [Event]**	**P**
Jump to the next similar event with the same context.	**Go →Next [Event] in [Context]**	**Shift+N**
Jump to the previous similar event with the same context.	**Go →Previous [Event] in [Context]**	**Shift+P**
Open dialog to go to an event by Id.	**Go →Go to Event...**	**Ctrl+G**
Open dialog to go to an event by timestamp.	**Go →Go to Timestamp...**	**Ctrl+Shift+G**
Scroll forward.	**Go →Scroll Forward**	**Left**, **Ctrl +Scroll up**, **Click&Drag**

Action	Menu	Shortcut
Scroll back.	**Go →Scroll Back**	**Right**, **Ctrl +Scroll down**, **Click&Drag**
Window		
Show/hide the Events window.	**Window →Events View**	**E**
Show/hide the Timeline window.	**Window →Time View**	**T**
Show/hide the CPU Load window.	**Window →CPU Load View**	**L**
Show/hide the Contexts window.	**Window →Context View**	**C**
Show/hide the Terminal window.	**Window →Terminal View**	**M**
Show/hide the System information window.	**Window →System View**	**S**
Show/hide the Log window.	**Window →Log View**	**O**
Show/Hide the Status bar	**Window →Show/Hide Status Bar**	**None**
Show/Hide the Tool bar	**Window →Show/Hide Tool Bar**	**None**
Open application preferences dialog.	**Tool →Preferences**	**Alt+,**
Help		
Open this SystemView Manual.	**Help →SystemView Manual**	**F11**
Show SystemView information.	**Help →About SystemView**	**F12**

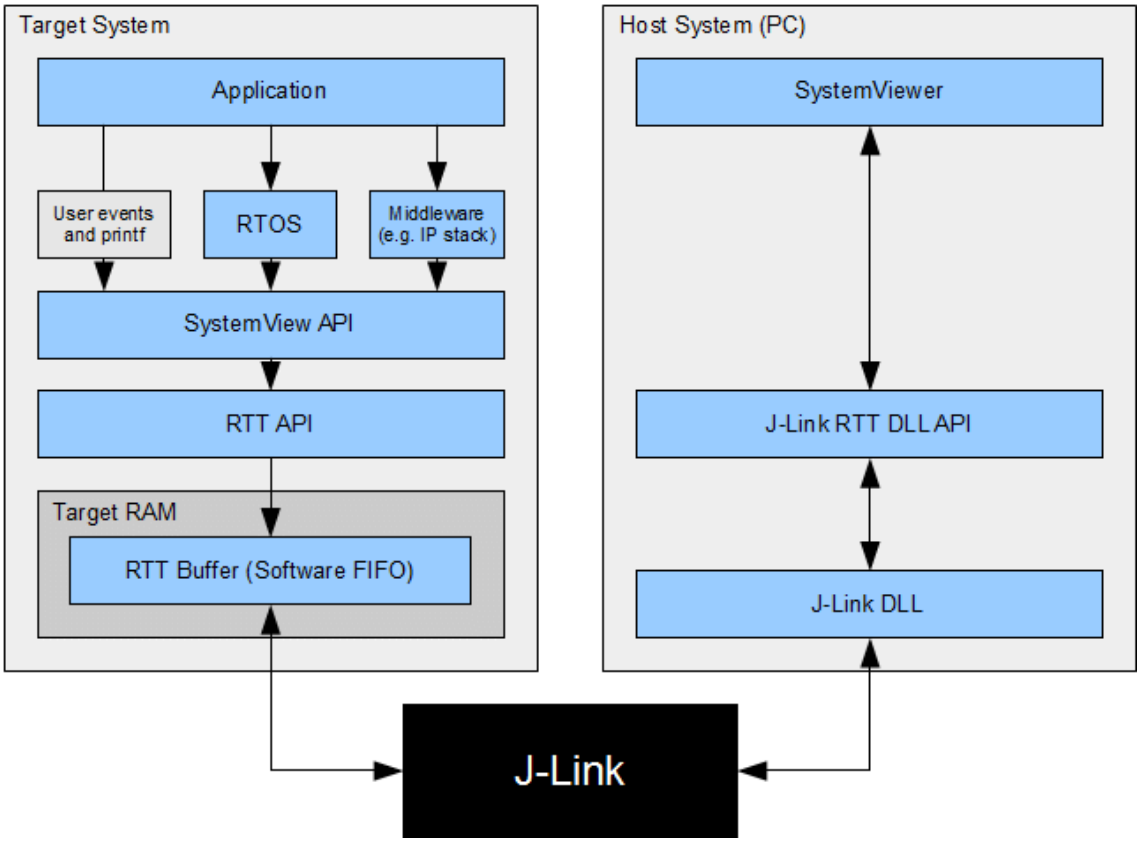
Chapter 4

Recording with SystemView

This section describes how to use the SystemView App for continuous recording and how to do manual single-shot recording with a debugger.

4.1 Continuous recording

With a J-Link debug probe and the SEGGER Real Time Transfer technology (RTT), SystemView can continuously record target execution in real time, while the target is running. RTT requires the ability of reading memory via the debug interface during program execution. This especially includes ARM Cortex-M0, M0+, M1, M3, M4 and M7 processors as well as all Renesas RX devices.



Start recording

To start recording with SystemView, connect the J-Link and target and click **Target →Start Recording**.

Enter or select the device name. The drop-down lists the most recently used devices. If the current device is not part of the list, it can be entered manually. A list of supported device names is available at {https://www.segger.com/`jlink_supported_devices`.html}.

> **Note**
>
> For

RTT Control Block Auto Detection, as well as for some devices the exact device has to be known. It is recommended to not only select a core.

If necessary configure the J-Link connection, when connecting to a specific J-Link or to a J-Link, which is connected via TCP/IP.

Select the target interface and target interface speed for the connected device.

Configure the RTT Control Block Detection. In most cases Auto Detection can be used. If the RTT Control Block can not be detected, select a Search Range in which the RTT Control Block can be, or enter the exact Address directly.

Click OK to connect to the target and start recording.

> **Note**
>
> SystemView

can be used parallel to a debugger. In this case recording can be done while the debugger is running. Make sure all required configuration is done in the debugger.

SystemView will continuously read the recorded data form the target and update its windows with the data while running.

Stop recording

To stop recording select **Target**→**Stop Recording**.

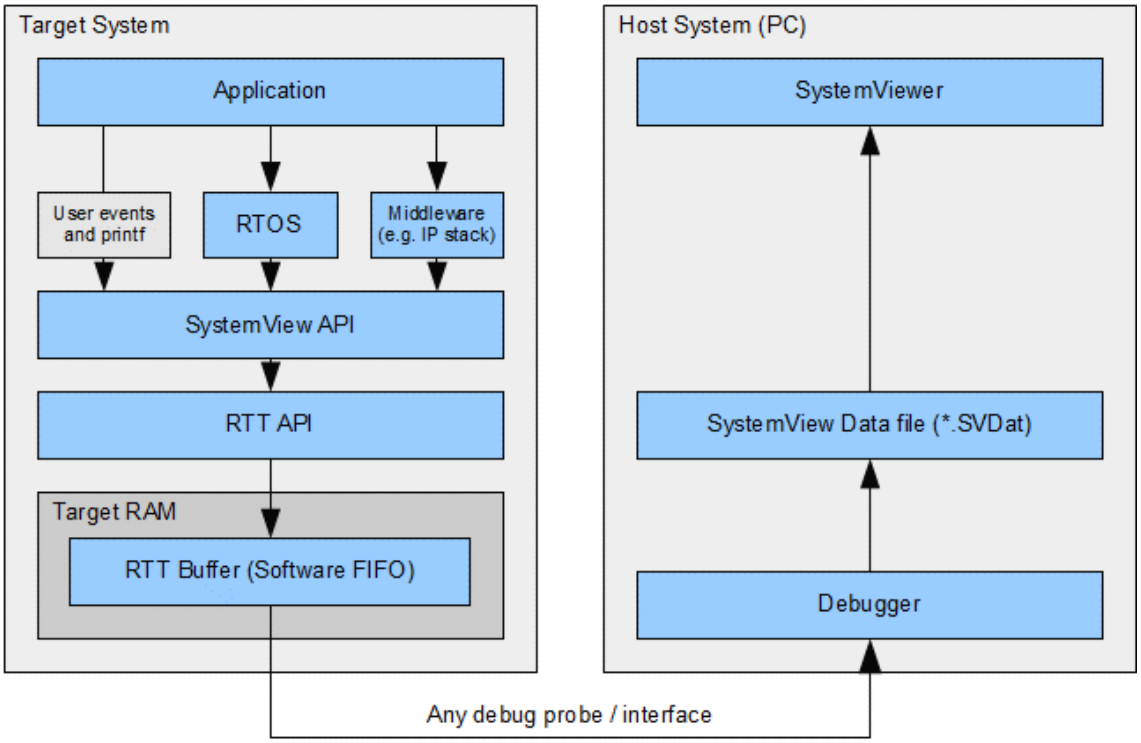
SystemView is limited to 1 000 000 events and automatically stops afterwards.

4.2 Single-shot recording

When the target device does not support RTT or when no J-Link is used, SEGGER SystemView can be used to record data until its target buffer is filled.

In single-shot mode the recording is started manually in the application, which allows recording only specific parts, which are of interest.

As a usual application generates about 5 to 15 kByte recording data per second and peaks only to higher rates at full load, even a small buffer in the internal RAM can be used to record data for analysis of critical parts. When using external RAM SystemView can record for a long time, even in single-shot mode.



Get single-shot data from the system

To get the data which has been recorded in single-shot mode, the SystemView buffer has to be read via the SystemView App or an external debugger.

- Connect a debugger and load the target application.
- Configure and initialize SystemView from the application (`SEGGER_SYSVIEW_Conf()` or `SEGGER_SYSVIEW_Init`).
- Start recording in the application from where it should be analyzed (`SEGGER_SYSVIEW_Start`).

With a J-Link SystemView can automatically read single-shot data from the target.

- Start the SystemView App and select **Target→Read Recorded Data**.

Without a J-Link or without SystemView the data can be read using following steps:

- Halt the application in the debugger when the buffer is full or after recording has been done.
- Get the SystemView RTT buffer address and the number of bytes used (Normally `_SEGGER_RTT.aUp[1].pBuffer` and `_SEGGER_RTT.aUp[1].WrOff`).
- Save the number of bytes from the buffer to a file with .SVDat extension.
- Open the file with the SystemView App.

To be able to record more than once, the buffer write offset (`_SEGGER_RTT.aUp[1].WrOff`) can be set to 0 when the data has been read. To prevent SystemView overflow events to happen, the application should be halted as soon as the buffer is filled and cannot hold another SystemView event.

4.3 Post-mortem analysis

Post-mortem analysis is similar to single-shot recording, with one difference: SystemView events are continuously recorded and the SystemView buffer wraps around to overwrite older events when the buffer is filled. When reading out the buffer, the newest events are available.

Post-mortem analysis can be useful when a system runs for a long time and suddenly crashes. In this case the SystemView buffer can be read from the target and SystemView can show what happened in the system immediately before the crash.

> **Note**
>
> Post

-mortem analysis requires the debugger or debug probe to be able to connect to the target system without resetting it or modifying the RAM.

To get as much useful data for analysis as possible it is recommended to use a large buffer for SystemView, 8 kByte or more. External RAM can be used for the SystemView buffer.

To configure the target system for post-mortem mode, please refer to `SEGGER_SYSVIEW_POST_MORTEM_MODE` and `SEGGER_SYSVIEW_SYNC_PERIOD_SHIFT` in chapter *Target configuration* on page 55.

Get post-mortem data from the system

To get the data which has been recorded in post-mortem mode, the SystemView buffer has to be read via the SystemView App or an external debugger.

- Configure and initialize SystemView from the application (`SEGGER_SYSVIEW_Conf()` or `SEGGER_SYSVIEW_Init`).
- Start recording in the application from where it should be analyzed (`SEGGER_SYSVIEW_Start`).
- Connect a debugger, load the target application, and let the system run.

With a J-Link SystemView can automatically read post-mortem data from the target.

- Start SystemView and select **Target**→**Read Recorded Data**.

Without a J-Link or without SystemView the data can be read using following steps:

Since the SystemView buffer is a ring buffer, the data might have to be read in two chunks to start reading at the beginning and save as much data as possible.

- Configure and initialize SystemView from the application (`SEGGER_SYSVIEW_Conf()` or `SEGGER_SYSVIEW_Init`).
- Start recording in the application from where it should be analyzed (`SEGGER_SYSVIEW_Start`).
- Connect a debugger, load the target application, and let the system run.
- when the system crashed or all tests are done, attach with a debugger to the system and halt it.
- Get the SystemView RTT buffer (Usually `_SEGGER_RTT`.aUp[1].pBuffer).
- Save the data from pBuffer + WrOff until the end of the buffer to a file.
- Append the data from pBuffer until pBuffer + RdOff - 1 to the file.
- Save the file as *.SVdat or *.bin.
- Open the file with the SystemView App.

4.4 Save and load recordings

When recording is stopped, the recorded data can be saved to a file for later analysis and documentation. Select **File→Save Data**. The Recording Properties Dialog pops up, which allows saving a title, author, and description with the data file. Click **OK**. Select where to save the data and click **Save**.

Saved data can be opened via **File→Load Data**. The most recently used data files are available via the menu at **File→Recent Files**, too. SystemView can open *.bin and *.SVDat files.

Chapter 5

Target implementation - SYSTEMVIEW modules

This section describes the instrumentation of a target application to use SEGGER SystemView.

5.1 Prerequisites

To use SEGGER SystemView the target source modules of the SystemView package need to be added to the application make file or project.

The SEGGER SystemView package is available at {https://www.segger.com/systemview.html}.

To use continuous real-time tracing the following additional items are required:

- An ARM Cortex-M or Renesas RX target.
- A J-Link Debug Probe.

For an easy start it is recommended to use SEGGER embOS V4.12 or later, which already includes the SystemView integration.

5.1.1 SEGGER SystemView target implementation modules

The following files are part of the SEGGER SystemView target implementation. We recommend to copy all files into the application project and keep the given directory structure.

File	Description
`/Config/Global.h`	Global type definitions for SEGGER code.
`/Config/SEGGER_RTT_Conf.h`	SEGGER Real Time Transfer (RTT) configuration file.
`/Config/SEGGER_SYSVIEW_Conf.h`	SEGGER SYSTEMVIEW configuration file.
`/Config/SEGGER_SYSVIEW_Config_[SYSTEM].c`	Initialization of SystemView for [SYSTEM].
`/OS/SEGGER_SYSVIEW_[OS].c`	Interface between SYSTEMVIEW and [OS].
`/OS/SEGGER_SYSVIEW_[OS].h`	Interface header.
`/SEGGER/SEGGER.h`	Global header for SEGGER global types and general purpose utility functions.
`/SEGGER/SEGGER_RTT.c`	SEGGER RTT module source.
`/SEGGER/SEGGER_RTT.h`	SEGGER RTT module header.
`/SEGGER/SEGGER_SYSVIEW.c`	SEGGER SYSTEMVIEW module source.
`/SEGGER/SEGGER_SYSVIEW.h`	SEGGER SYSTEMVIEW module header.
`/SEGGER/SEGGER_SYSVIEW_ConfDefault.h`	SEGGER SYSTEMVIEW configuration fallback.
`/SEGGER/SEGGER_SYSVIEW_Int.h`	SEGGER SYSTEMVIEW internal header.

5.2 Including SEGGER SystemView in the application

To use SEGGER SystemView, the target implementation modules must be added to the target application. Copy the sources from `/Config/` and `/SEGGER/`, as well as the `SEGGER_SYSVIEW_Config_[SYSTEM].c` and `SEGGER_SYSVIEW_[OS].c/.h` which match your system and OS to the application source and include them in the project.

Example

For a system with embOS on a Cortex-M3 include `SEGGER_SYSVIEW_Config_embOS.c`, `SEGGER_SYSVIEW_embOS.c` and `SEGGER_SYSVIEW_embOS.h`.

For a system with no OS or no instrumented OS on a Cortex-M3 include `SEGGER_SYSVIEW_Config_NoOS.c` only.

5.3 Initializing SystemView

The system information are sent by the application. This information can be configured via defines in SEGGER_SYSVIEW_Config_[SYSTEM].c. Add a call to SEGGER_SYSVIEW_Conf() in the main function to initialize SystemView.

```
#include "SEGGER_SYSVIEW.h"

/*********************************************************************
*
*       main()
*
* Function description
*   Application entry point
*/
int main(void) {
  OS_IncDI();                      /* Initially disable interrupts         */
  OS_InitKern();                   /* Initialize OS                        */
  OS_InitHW();                     /* Initialize Hardware for OS           */
  BSP_Init();                      /* Initialize BSP module                */

  SEGGER_SYSVIEW_Conf();           /* Configure and initialize SystemView  */

  /* You need to create at least one task before calling OS_Start() */
  OS_CREATETASK(&TCB0, "MainTask", MainTask, 100, Stack0);
  OS_Start();                      /* Start multitasking                   */
  return 0;
}
```

The generic part of SEGGER SystemView is now ready to monitor the application.

When using embOS V4.12 or later with profiling enabled, SystemView events for ISRs, Task, and API calls are generated. When not using embOS, appropriate events must be generated by the application.

Download the application to the target and let it run. As long as the SystemView App is not connected, and SEGGER_SYSVIEW_Start is not called, the application will not generate SystemView events. When SystemView is connected or SEGGER_SYSVIEW_Start is called it will activate recording SystemView events.

5.4 Start and stop recording

When the data is read continuously with SystemView, the recording is started and stopped automatically by SystemView. While SystemView is not recording the target system will not generate SystemView events.

For single-shot recording `SEGGER_SYSVIEW_Start` has to be called in the application to activate recording SystemView events. Events are recorded until the SystemView buffer is filled or `SEGGER_SYSVIEW_Stop` is called.

For post-mortem analysis `SEGGER_SYSVIEW_Start` has to be called in the application to activate recording SystemView events. Events are recorded until `SEGGER_SYSVIEW_Stop` is called. Older events are overwritten when the SystemView buffer is filled.

5.5 The SystemView system information config

The included files SEGGER_SYSVIEW_Config_[SYSTEM].c provide the configuration of SystemView and can in most cases be used without modification.

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
*                       The Embedded Experts                         *
*                          www.segger.com                            *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File    : SEGGER_SYSVIEW_Config_embOS.c
Purpose : Sample setup configuration of SystemView with embOS.
Revision: $Rev: 3734 $
*/
#include "RTOS.h"
#include "SEGGER_SYSVIEW.h"
#include "SEGGER_SYSVIEW_embOS.h"

//
// SystemcoreClock can be used in most CMSIS compatible projects.
// In non-CMSIS projects define SYSVIEW_CPU_FREQ.
//
extern unsigned int SystemCoreClock;

/*********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/
// The application name to be displayed in SystemViewer
#ifndef   SYSVIEW_APP_NAME
  #define SYSVIEW_APP_NAME        "Demo Application"
#endif

// The target device name
#ifndef   SYSVIEW_DEVICE_NAME
  #define SYSVIEW_DEVICE_NAME     "Cortex-M4"
#endif

// Frequency of the timestamp. Must match SEGGER_SYSVIEW_Conf.h
#ifndef   SYSVIEW_TIMESTAMP_FREQ
  #define SYSVIEW_TIMESTAMP_FREQ  (SystemCoreClock)
#endif

// System Frequency. SystemcoreClock is used in most CMSIS compatible projects.
#ifndef   SYSVIEW_CPU_FREQ
  #define SYSVIEW_CPU_FREQ        (SystemCoreClock)
#endif

// The lowest RAM address used for IDs (pointers)
#ifndef   SYSVIEW_RAM_BASE
  #define SYSVIEW_RAM_BASE        (0x20000000)
#endif

#ifndef   SYSVIEW_SYSDESC0
  #define SYSVIEW_SYSDESC0        "I#15=SysTick"
#endif

// Define as
 1 if the Cortex-M cycle counter is used as SystemView timestamp. Must match SEGGER_SYSVIEW_Conf.h
#ifndef   USE_CYCCNT_TIMESTAMP
  #define USE_CYCCNT_TIMESTAMP    1
#endif
```



```
//#ifndef   SYSVIEW_SYSDESC1
//  #define SYSVIEW_SYSDESC1      ""
//#endif

//#ifndef   SYSVIEW_SYSDESC2
//  #define SYSVIEW_SYSDESC2      ""
//#endif

/*********************************************************************
*
*       Defines, fixed
*
**********************************************************************
*/
#define DWT_CTRL                  (*(volatile OS_U32*) (0xE0001000uL))
  // DWT Control Register
#define NOCYCCNT_BIT              (1uL << 25)
  // Cycle counter support bit
#define CYCCNTENA_BIT             (1uL << 0)
  // Cycle counter enable bit

/*********************************************************************
*
*       _cbSendSystemDesc()
*
*  Function description
*    Sends SystemView description strings.
*/
static void _cbSendSystemDesc(void) {
  SEGGER_SYSVIEW_SendSysDesc("N="SYSVIEW_APP_NAME",O=embOS,D="SYSVIEW_DEVICE_NAME);
#ifdef SYSVIEW_SYSDESC0
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC0);
#endif
#ifdef SYSVIEW_SYSDESC1
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC1);
#endif
#ifdef SYSVIEW_SYSDESC2
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC2);
#endif
}

/*********************************************************************
*
*       Global functions
*
**********************************************************************
*/
void SEGGER_SYSVIEW_Conf(void) {
#if USE_CYCCNT_TIMESTAMP
  //
  //  The cycle counter must be activated in order
  //  to use time related functions.
  //
  if ((DWT_CTRL & NOCYCCNT_BIT) == 0) {       // Cycle counter supported?
    if ((DWT_CTRL & CYCCNTENA_BIT) == 0) {    // Cycle counter not enabled?
      DWT_CTRL |= CYCCNTENA_BIT;              // Enable Cycle counter
    }
  }
#endif
  SEGGER_SYSVIEW_Init(SYSVIEW_TIMESTAMP_FREQ, SYSVIEW_CPU_FREQ,
                      &SYSVIEW_X_OS_TraceAPI, _cbSendSystemDesc);
  SEGGER_SYSVIEW_SetRAMBase(SYSVIEW_RAM_BASE);
  OS_SetTraceAPI(&embOS_TraceAPI_SYSVIEW);    // Configure embOS to use SYSVIEW.
}

/************************** End of file ****************************/
```


Chapter 6

Target configuration

SEGGER SystemView is configurable to match the target device and application. The default compile-time configuration flags are preconfigured with valid values, to match the requirements of most systems and normally do not require modification.

The default configuration of SystemView can be changed via compile-time flags which can be added to `SEGGER_SYSVIEW_Conf.h`.

6.1 System-specific configuration

The following compile-time configuration is required to match the target system. The sample configuration in SEGGER_SYSVIEW_Conf.h defines the configuration to match most systems (i.e. Cortex-M devices with Embedded Studio, GCC, IAR or Keil ARM). If the sample configuration does not include the used system, the configuration has to be adapted accordingly.

For a detailed description of the system-specific configuration, refer to *Supported CPUs* on page 65.

6.1.1 SEGGER_SYSVIEW_GET_TIMESTAMP()

Function macro to retrieve the system timestamp for SystemView events.

On Cortex-M3/4 devices the Cortex-M cycle counter can be used as system timestamp.

Default on Cortex-M3/4: `(*(U32 *)(0xE0001004))`

On most other devices the system timestamp has to be generated by a timer. With the default configuration the system timestamp is retrieved via the user-provided function SEGGER_SYSVIEW_X_GetTimestamp().

Default on other cores: `SEGGER_SYSVIEW_X_GetTimestamp()`

For an example, please refer to SEGGER_SYSVIEW_Config_embOS_CM0.c or SEGGER_SYSVIEW_Config_NoOS_RX.c

> **Note**
>
> The

frequency of the system timestamp has to be provided in SEGGER_SYSVIEW_Init.

6.1.2 SEGGER_SYSVIEW_TIMESTAMP_BITS

Number of valid low-order bits delivered by clock source as system timestamp.

If an unmodified clock source is used as system timestamp, the number of valid bits is the bit-width of the clock source (i.e. 32 or 16 bit).

Default: 32 (32-bit clock source used)

Example to save bandwidth

As SystemView packets use a variable-length encoding, shifting timestamps can save both buffer space and bandwidth.

A 32-bit clock source, i.e. the Cortex-M cycle counter on Cortex-M3/4 can be shifted by 4, resulting in the number of valid timestamp bits to be 28 and the timestamp frequency, as used in SEGGER_SYSVIEW_Init, to be the core clock frequency divided by 16.

```
#define SEGGER_SYSVIEW_GET_TIMESTAMP() ((*(U32 *)(0xE0001004)) >> 4)
```

```
#define SEGGER_SYSVIEW_TIMESTAMP_BITS 28.
```

6.1.3 SEGGER_SYSVIEW_GET_INTERRUPT_ID()

Function macro to get the currently active interrupt.

On Cortex-M devices the active vector can be read from the ICSR.

Default on Cortex-M3/4: `((*(U32 *)(0xE000ED04)) & 0x1FF)`

Default on Cortex-M0/1: `((*(U32 *)(0xE000ED04)) & 0x3F)`

On other devices the active interrupt can either be retrieved from the interrupt controller directly, can be saved in a variable in the generic interrupt handler, or has to be assigned manually in each interrupt routine.

By default this can be done with the user-provided function SEGGER_SYSVIEW_X_GetInterruptId() or by replacing the macro definition.

For an example refer to SEGGER_SYSVIEW_Config_embOS_RX.c or *Cortex-A/R Interrupt ID* on page 78.

6.1.4 SEGGER_SYSVIEW_LOCK()

Function macro to recursively lock SystemView transfers from being interrupted. I.e. disable interrupts.

SEGGER_SYSVIEW_LOCK() must preserve the previous lock state to be restored in SEGGER_SYSVIEW_UNLOCK().

Recording a SystemView event must not be interrupted by recording another event. Therefore all interrupts which are recorded by SystemView (call SEGGER_SYSVIEW_RecordEnterISR / SEGGER_SYSVIEW_RecordExitISR), call an instrumented function (i.e. an OS API function), cause an immediate context switch, or possibly create any other SystemView event have to be disabled.

SEGGER_SYSVIEW_LOCK() can use the same locking mechanism as SEGGER_RTT_LOCK().

Default: SEGGER_RTT_LOCK()

SEGGER_RTT_LOCK() is defined for most systems (i.e. Cortex-M devices with Embedded Studio, GCC, IAR or Keil ARM, and RX devices with IAR) in SEGGER_RTT_Conf.h. If the macro is not defined, or empty, it has to be provided to match the target system.

6.1.5 SEGGER_SYSVIEW_UNLOCK()

Function macro to recursively unlock SystemView transfers from being interrupted. I.e. restore previous interrupt state.

SEGGER_SYSVIEW_UNLOCK() can use the same locking mechanism as SEGGER_RTT_UNLOCK().

Default: SEGGER_RTT_UNLOCK()

SEGGER_RTT_UNLOCK() is defined for most systems (i.e. Cortex-M devices with Embedded Studio, GCC, IAR or Keil ARM, and RX devices with IAR) in SEGGER_RTT_Conf.h. If the macro is not defined, or empty, it has to be provided to match the target system.

6.2 Generic configuration

The following compile-time flags can be used to tune or change how SystemView events are recorded.

The default compile-time configuration flags are preconfigured with valid values, to match the requirements of most systems and normally do not require modification.

6.2.1 SEGGER_SYSVIEW_RTT_BUFFER_SIZE

Number of bytes that SystemView uses for the recording buffer.

For continuous recording a buffer of 1024 bytes is sufficient in most cases. Depending on the target interface speed, the target speed and the system load the buffer size might be increased to up to 4096 bytes.

For single-shot recording the buffer size determines the number of events which can be recorded. A system might generate between 10 and 50 kByte/s, depending on its load. A buffer of at least 8 kByte, up to the whole free RAM space is recommended. The buffer can also be in external RAM.

For post-mortem analysis the buffer size determines the maximum number of events which will be available for analysis. A system might generate between 10 and 50 kByte/s, depending on its load. A buffer of at least 8 kByte, up to the whole free RAM space is recommended. The buffer can also be in external RAM.

Default: 1024 bytes

6.2.2 SEGGER_SYSVIEW_RTT_CHANNEL

The RTT Channel used for SystemView event recording and communication. 0: Auto selection

> **Note**

`SEGGER_RTT_MAX_NUM_UP_BUFFERS`, defined in `SEGGER_RTT_Conf.h` has to be greater than `SEGGER_SYSVIEW_RTT_CHANNEL`.

Default: 0

6.2.3 SEGGER_SYSVIEW_USE_STATIC_BUFFER

If set to 1 SystemView uses a static buffer to create SystemView events. This in general saves space, since only one buffer is required and task stacks can be as small as possible. When a static buffer is used, critical code executed between SystemView locking invocations takes slightly longer.

If set to 0 SystemView events are created on the stack. Make sure all task stacks, as well as the C stack for interrupts are large enough to hold the largest SystemView events (~228 bytes). SystemView locks only while transferring the stack buffer into the RTT buffer.

Default: 1

6.2.4 SEGGER_SYSVIEW_POST_MORTEM_MODE

If set to 1 post-mortem analysis mode is enabled.

In post-mortem mode, SystemView uses a cyclical buffer and preserves all events up to the final recorded even rather than dropping events when the buffer is full.

> **Note**

> Do

not use post-mortem analysis mode when an attached J-Link actively reads RTT data.

Default: 0

6.2.5 SEGGER_SYSVIEW_SYNC_PERIOD_SHIFT

Configure how often `Sync` and `System Info` events are sent in post-mortem mode. Make sure at least one sync is available in the SystemView buffer.

The recommended sync frequency is `Buffer Size / 16`

Default: 8 = Sync every 256 Packets

6.2.6 SEGGER_SYSVIEW_ID_BASE

Value to be subtracted from IDs recorded in SystemView packets.

IDs are TaskIds, TimerIds, and ResourceIds, which are usually pointers to a structure in RAM. Parameters sent in OS and middleware API events can also be encoded as IDs by the instrumentation.

> **Note**
>
> If

the instrumented OS does not use pointers for TaskIds, TimerIds, or ResourceIds, `SEGGER_SYSVIEW_ID_BASE` needs to be set to 0.

As SystemView packets use a variable-length encoding for pointers, correctly re-basing addresses can save both buffer space and bandwidth.

Define as the lowest RAM address used in the system.

Can be overridden by the application via `SEGGER_SYSVIEW_SetRAMBase` on initialization.

In case of doubt define `SEGGER_SYSVIEW_ID_BASE` as 0.

Default: `0x10000000`

6.2.7 SEGGER_SYSVIEW_ID_SHIFT

Number of bits to shift IDs recorded in SystemView packets.

IDs are TaskIds, TimerIds, and ResourceIds, which are usually pointers to a structure in RAM. Parameters sent in OS and middleware API events can also be encoded as IDs by the instrumentation.

> **Note**
>
> If

the instrumented OS does not use pointers for TaskIds, TimerIds, or ResourceIds, `SEGGER_SYSVIEW_ID_SHIFT` needs to be set to 0.

As SystemView packets use a variable-length encoding for pointers, correctly shifting addresses can save both buffer space and bandwidth.

For most applications on 32-bit processors, all IDs recorded in SystemView events are really pointers and as such multiples of 4, so that the lowest 2 bits can be safely ignored.

In case of doubt define `SEGGER_SYSVIEW_ID_SHIFT` as 0.

Default: 2

6.2.8 SEGGER_SYSVIEW_MAX_STRING_LEN

Maximum string length to be recorded by SystemView events.

Strings are used in the SystemView `printf`-style user functions, as well as in `SEGGER_SYSVIEW_SendSysDesc` and `SEGGER_SYSVIEW_RecordModuleDescription`. Make sure `SEGGER_SYSVIEW_MAX_STRING_LEN` matches the string length used in these functions.

The maximum supported string length is 255 bytes.

Default: 128

6.2.9 SEGGER_SYSVIEW_MAX_ARGUMENTS

Maximum number of arguments to be sent with `SEGGER_SYSVIEW_PrintfHost`, `SEGGER_SYSVIEW_PrintfHostEx`, `SEGGER_SYSVIEW_WarnfHost`, and `SEGGER_SYSVIEW_ErrorfHost`.

If these functions are not used in the application `SEGGER_SYSVIEW_MAX_ARGUMENTS` can be set to 0 to minimize the static buffer size.

Default: 16

6.2.10 SEGGER_SYSVIEW_BUFFER_SECTION

The SystemView RTT Buffer may be placed into a dedicated section, instead of the default data section. This allows placing the buffer into external memory or at a given address.

When `SEGGER_SYSVIEW_BUFFER_SECTION` is defined, the section has to be defined in the linker script.

Default: `SEGGER_RTT_SECTION` or not defined

Example in Embedded Studio

```
//
// SEGGER_SYSVIEW_Conf.h
//
#define SEGGER_SYSVIEW_BUFFER_SECTION "SYSTEMVIEW_RAM"

//
// flash_placement.xml
//
<MemorySegment name="ExtRAM">
  <ProgramSection load="No" name="SYSTEMVIEW_RAM" start="0x40000000" />
</MemorySegment>
```

6.2.11 RTT configuration

The following compile-time flags can be used to tune or change RTT.

The default compile-time configuration flags are preconfigured with valid values, to match the requirements of most systems and normally do not require modification.

6.2.11.1 BUFFER_SIZE_UP

Number of bytes to be used for the RTT Terminal output channel.

RTT can be used for printf terminal output without modification. `BUFFER_SIZE_UP` defines how many bytes can be buffered for this.

If RTT Terminal output is not used, define `BUFFER_SIZE_UP` to its minimum of 4.

Default: 1024 Bytes

6.2.11.2 BUFFER_SIZE_DOWN

Number of bytes to be used for the RTT Terminal input channel.

RTT can receive input from the host on the terminal input channel. `BUFFER_SIZE_DOWN` defines how many bytes can be buffered and therefore sent at once from the host.

If RTT Terminal input is not used, define `BUFFER_SIZE_DOWN` to its minimum of 4.

Default: 16 Bytes

6.2.11.3 SEGGER_RTT_MAX_NUM_UP_BUFFERS

Maximum number of RTT up (to host) buffers. Buffer 0 is always used for RTT terminal output, so to use it with SystemView `SEGGER_RTT_MAX_NUM_UP_BUFFERS` has to be at least 2.

Default: 2

6.2.11.4 SEGGER_RTT_MAX_NUM_DOWN_BUFFERS

Maximum number of RTT down (to target) buffers. Buffer 0 is always used for RTT terminal input, so to use it with SystemView `SEGGER_RTT_MAX_NUM_UP_BUFFERS` has to be at least 2.

Default: 2

6.2.11.5 SEGGER_RTT_MODE_DEFAULT

Mode for pre-initialized RTT terminal channel (buffer 0).

Default: `SEGGER_RTT_MODE_NO_BLOCK_SKIP`

6.2.11.6 SEGGER_RTT_PRINTF_BUFFER_SIZE

Size of buffer for RTT printf to bulk-send chars via RTT. Can be defined as 0 if `SEGGER_RTT_Printf` is not used.

Default: 64

6.2.11.7 SEGGER_RTT_SECTION

The RTT Control Block may be placed into a dedicated section, instead of the default data section. This allows placing it at a known address to be able to use the J-Link auto-detection or easily specify a search range.

When `SEGGER_RTT_SECTION` is defined, the application has to make sure the section is valid, either by initializing it with 0 in the startup code or explicitly calling `SEGGER_RTT_Init()` at the start of the application. `SEGGER_RTT_Init()` is implicitly called by `SEGGER_SYSVIEW_Init()`.

Default: not defined

6.2.11.8 SEGGER_RTT_BUFFER_SECTION

The RTT terminal buffer may be placed into a dedicated section, instead of the default data section. This allows placing the buffer into external memory or at a given address.

Default: `SEGGER_RTT_SECTION` or not defined

6.3 Optimizing SystemView

In order to get the most precise run-time information from a target system, the recording instrumentation code needs to be fast, least intrusive, small, and efficient. The SystemView code is written to be efficient and least intrusive. Speed and size of SystemView are a matter of target and compiler configuration. The following sections describe how to optimize SystemView.

6.3.1 Compiler optimization

The compiler optimization of the SystemView target implementation should always be turned on, even in debug builds, to generate fast recording routines, causing less overhead and be least intrusive.

The configuration to favour speed or size optimization is compiler-dependent. In some cases a balanced configuration can be faster than a speed-only configuration.

6.3.2 Recording optimization

SystemView uses a variable-length encoding to store and transfer events, which enables saving buffer space and bandwidth on the debug interface.

The size of some event parameters can be optimized via compile-time configuration.

Shrink IDs

IDs are pointers to a symbol in RAM, i.e. a Task ID is a pointer to the task control block. To minimize the length of recorded IDs they can be shrunken.

`SEGGER_SYSVIEW_ID_BASE` is subtracted from a pointer to get its ID. It can be set to subtract the base RAM address from pointers, which still results in unique, but smaller IDs. For example if the RAM range is `0x20000000` to `0x20001000` it is recommended to define `SEGGER_SYSVIEW_ID_BASE` as `0x20000000`, which results in the pointer `0x20000100` to have the ID `0x100` and requires two instead of four bits to store it.

`SEGGER_SYSVIEW_ID_SHIFT` is the number of bits a pointer is shifted right to get its ID. If all recorded pointers are 4 byte aligned, `SEGGER_SYSVIEW_ID_SHIFT` can be defined as 2. A pointer `0x20000100` would then have the ID `0x8000040` or with the previous subtraction of `SEGGER_SYSVIEW_ID_BASE` as `0x20000000` the ID would be `0x40`, requiring only one byte to be recorded.

Timestamp source

Event timestamps in SystemView are recorded as the difference of the timestamp to the previous event. This saves buffer space per se.

While it is recommended to use a timestamp source with the CPU clock frequency for highest time resolution, a lower timestamp frequency might save additional buffer space as the timestamp delta is lower.

With a CPU clock frequency of 160 MHz the timestamp might be shifted by 4, resulting in a timestamp frequency of 10 MHz (100 ns resolution), and 4 bits less to be encoded.

When the timestamp size is not 32-bit any more, i.e. it wraps around before `0xFFFFFFFF`, `SEGGER_SYSVIEW_TIMESTAMP_BITS` has to be defined as the timestamp size, e.g. as 28 when shifting a 32-bit timestamp by 4.

6.3.3 Buffer configuration

The recording and communication buffer size for SystemView and RTT can be set in the target configuration.

For continuous recording a small buffer of 1 to 4 kByte is sufficient in most cases and allows using SystemView even with a small internal RAM.

For single-shot and post-mortem mode a larger buffer can be desirable. In this case SEGGER_SYSVIEW_RTT_BUFFER_SIZE can be set to a larger value. To place the SystemView recording buffer into external RAM a SEGGER_SYSVIEW_BUFFER_SECTION can be defined and the linker script adapted accordingly.

If only SystemView is used and no terminal output with RTT, BUFFER_SIZE_UP in SEGGER_RTT_Conf.h can be set to a smaller value to save memory.

Chapter 7

Supported CPUs

This section describes how to set up and configure the SystemView modules for different target CPUs.

SEGGER SystemView virtually supports any target CPU, however, continuous recording is only possible with CPUs, which support background memory access - ARM Cortex-M and Renesas RX. On other CPUs SystemView can be used in single-shot or post-mortem analysis mode. Refer to *Single-shot recording* on page 44.

In order for SystemView to run properly, some target specific configuration needs to be done. This configuration is described for some CPUs below.

7.1 Cortex-M3 / Cortex-M4

Recording mode	Supported?
Continuous recording	Yes
Single-shot recording	Yes
Post-mortem analysis	Yes

7.1.1 Event timestamp

The timestamp source on Cortex-M3 / Cortex-M4 can be the cycle counter, which allows cycle-accurate event recording.

In order to save bandwith when recording events, the cycle counter can optionally be right-shifted, i.e. by 4 bits, which results in a timestamp frequency of core speed divided by 16.

Configuration:

```
//
// Use full cycle counter for higher precision
//
#define SEGGER_SYSVIEW_GET_TIMESTAMP()  (*(U32 *)(0xE0001004))
#define SEGGER_SYSVIEW_TIMESTAMP_BITS   (32)
//
// Use cycle counter divided by 16 for smaller size / bandwidth
//
#define SEGGER_SYSVIEW_GET_TIMESTAMP()  ((*(U32 *)(0xE0001004)) >> 4)
#define SEGGER_SYSVIEW_TIMESTAMP_BITS   (28)
```

7.1.2 Interrupt ID

The currently active interrupt can be directly identified by reading the Cortex-M ICSR[8:0], which is the active vector field in the interrupt controller status register (ICSR).

Configuration:

```
//
// Get the interrupt Id by reading the Cortex-M ICSR[8:0]
//
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()   ((*(U32 *)(0xE000ED04)) & 0x1FF)
```

7.1.3 SystemView lock and unlock

Locking and unlocking SystemView to prevent transferring records from being interrupted can be done by disabling interrupts. On Cortex-M3 / Cortex-M4 not all interrupts need to be disabled, only those which might itself generate SystemView events or cause a task switch in the OS.

By default the priority mask is set to 32, disabling all interrupts with a priority of 32 or lower (higher numerical value).

Make sure to mask all interrupts which can send RTT data, i.e. generate SystemView events, or cause task switches. When high-priority interrupts must not be masked while sending RTT data, `SEGGER_RTT_MAX_INTERRUPT_PRIORITY` needs to be adjusted accordingly. (Higher priority = lower priority number)

Default value for embOS: 128u

Default configuration in FreeRTOS: `configMAX_SYSCALL_INTERRUPT_PRIORITY: ( configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY << (8 - configPRIO_BITS) )`

In case of doubt disable all interrupts.

Lock and unlock for SystemView and RTT can be the same.

Configuration:

```
  //
  // RTT locking for GCC toolchains in SEGGER_RTT_Conf.h
  //
  #define SEGGER_RTT_LOCK()     {                                           \
                                    unsigned int LockState;                 \
                                  __asm volatile ("mrs   %0, basepri  \n\t"     \
                                                  "mov   r1, $32     \n\t"     \
                                                  "msr   basepri, r1  \n\t"     \
                                                  : "=r" (LockState)            \
                                                  :                             \
                                                  : "r1"                        \
                                                  );

  #define SEGGER_RTT_UNLOCK()     __asm volatile ("msr   basepri, %0  \n\t"     \
                                                  :                             \
                                                  : "r" (LockState)             \
                                                  :                             \
                                                  );                            \
                                }
  //
  // Define SystemView locking in SEGGER_SYSVIEW_Conf.h
  //
  #define SEGGER_SYSVIEW_LOCK()   SEGGER_RTT_LOCK()
  #define SEGGER_SYSVIEW_UNLOCK() SEGGER_RTT_UNLOCK()
```

7.1.4 Sample configuration

SEGGER_SYSVIEW_Conf.h

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
**********************************************************************
-------------------------- END-OF-HEADER -----------------------------

File        : SEGGER_SYSVIEW_Conf.h
Purpose     : SEGGER SysView configuration for Cortex-M3 / Cortex-M4.
*/

#ifndef SEGGER_SYSVIEW_CONF_H
#define SEGGER_SYSVIEW_CONF_H

/*********************************************************************
*
*       SysView timestamp configuration
*/
// Cortex-M cycle counter.
#define SEGGER_SYSVIEW_GET_TIMESTAMP()      ((*(U32 *)(0xE0001004)))
// Number of valid bits low-order delivered as timestamp.
#define SEGGER_SYSVIEW_TIMESTAMP_BITS       32

/*********************************************************************
*
*       SysView Id configuration
*/
// Default value for the lowest Id reported by the application.
// Can be overridden by the application via SEGGER_SYSVIEW_SetRAMBase().
#define SEGGER_SYSVIEW_ID_BASE         0x20000000
// Number of bits to shift the Id to save bandwidth.
// (i.e. 2 when all reported Ids (pointers) are 4 byte aligned)
#define SEGGER_SYSVIEW_ID_SHIFT        0
```



```
/*********************************************************************
*
*       SysView interrupt configuration
*/
// Get the currently active interrupt Id. (read Cortex-M ICSR[8:0]
 = active vector)
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()   ((*(U32 *)(0xE000ED04)) & 0x1FF)

/*********************************************************************
*
*       SysView locking
*/
// Lock SysView (nestable)
#define SEGGER_SYSVIEW_LOCK()             SEGGER_RTT_LOCK()
// Unlock SysView (nestable)
#define SEGGER_SYSVIEW_UNLOCK()           SEGGER_RTT_UNLOCK()

#endif

/************************** End of file ****************************/
```

SEGGER_SYSVIEW_Config_NoOS_CM3.c

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
*                       The Embedded Experts                         *
*                          www.segger.com                            *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File    : SEGGER_SYSVIEW_Config_NoOS.c
Purpose : Sample setup configuration of SystemView without an OS.
Revision: $Rev: 3734 $
*/
#include "SEGGER_SYSVIEW.h"

// SystemcoreClock can be used in most CMSIS compatible projects.
// In non-CMSIS projects define SYSVIEW_CPU_FREQ.
extern unsigned int SystemCoreClock;

/*********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/
// The application name to be displayed in SystemViewer
#define SYSVIEW_APP_NAME        "Demo Application"

// The target device name
#define SYSVIEW_DEVICE_NAME     "Cortex-M4"

// Frequency of the timestamp. Must match SEGGER_SYSVIEW_Conf.h
#define SYSVIEW_TIMESTAMP_FREQ  (SystemCoreClock)

// System Frequency. SystemcoreClock is used in most CMSIS compatible projects.
#define SYSVIEW_CPU_FREQ        (SystemCoreClock)

// The lowest RAM address used for IDs (pointers)
#define SYSVIEW_RAM_BASE        (0x10000000)

/*********************************************************************
*
*       _cbSendSystemDesc()
*
*  Function description
```



```
*    Sends SystemView description strings.
*/
static void _cbSendSystemDesc(void) {
  SEGGER_SYSVIEW_SendSysDesc("N="SYSVIEW_APP_NAME",D="SYSVIEW_DEVICE_NAME);
  SEGGER_SYSVIEW_SendSysDesc("I#15=SysTick");
}

/*********************************************************************
*
*       Global functions
*
**********************************************************************
*/
void SEGGER_SYSVIEW_Conf(void) {
  SEGGER_SYSVIEW_Init(SYSVIEW_TIMESTAMP_FREQ, SYSVIEW_CPU_FREQ,
                      0, _cbSendSystemDesc);
  SEGGER_SYSVIEW_SetRAMBase(SYSVIEW_RAM_BASE);
}

/************************** End of file ****************************/
```

7.2 Cortex-M7

Same features / settings etc. as for Cortex-M4 apply. For more information, please refer to *Cortex-M3 / Cortex-M4* on page 66.

Cache

When placing the RTT buffer for SystemView into memory that is cacheable, the performance is slightly lower (< 1% decrease in performance) for continuous recording mode via J-Link and RTT. This is because J-Link needs to perform cache maintenance operations when accessing the RTT buffer.

7.3 Cortex-M0 / Cortex-M0+ / Cortex-M1

Recording mode	Supported?
Continuous recording	Yes
Single-shot recording	Yes
Post-mortem analysis	Yes

7.3.1 Cortex-M0 Event timestamp

Cortex-M0, Cortex-M0+ and Cortex-M1 do not have a cycle count register. the event timestamp has to be provided by an application clock source, i.e. the system timer, SysTick. `SEGGER_SYSVIEW_X_GetTimestamp()` can be used to implement the functionality.

When the SysTick interrupt is used in the application, i.e. by the RTOS, the SysTick handler should increment `SEGGER_SYSVIEW_TickCnt`, otherwise a SysTick handler has to be added to the application and configured accordingly.

Configuration:

```
//
// SEGGER_SYSVIEW_TickCnt has to be defined in the module which
// handles the SysTick and must be incremented in the SysTick
// handler before any SYSVIEW event is generated.
//
// Example in embOS RTOSInit.c:
//
// unsigned int SEGGER_SYSVIEW_TickCnt; // <<-- Define SEGGER_SYSVIEW_TickCnt.
// void SysTick_Handler(void) {
// #if OS_PROFILE
//   SEGGER_SYSVIEW_TickCnt++;  // <<-- Increment SEGGER_SYSVIEW_TickCnt asap.
// #endif
//   OS_EnterNestableInterrupt();
//   OS_TICK_Handle();
//   OS_LeaveNestableInterrupt();
// }
//
extern unsigned int SEGGER_SYSVIEW_TickCnt;

/*********************************************************************
*
*       Defines, fixed
*
**********************************************************************
*/
#define SCB_ICSR
  (*(volatile U32*) (0xE000ED04uL)) // Interrupt Control State Register
#define SCB_ICSR_PENDSTSET_MASK     (1UL << 26)     // SysTick pending bit
#define SYST_RVR
  (*(volatile U32*) (0xE000E014uL)) // SysTick Reload Value Register
#define SYST_CVR
  (*(volatile U32*) (0xE000E018uL)) // SysTick Current Value Register
/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetTimestamp()
*
* Function description
*   Returns the current timestamp in ticks using the system tick
*   count and the SysTick counter.
*   All parameters of the SysTick have to be known and are set via
*   configuration defines on top of the file.
*
* Return value
*   The current timestamp.
*
```



```
* Additional information
*   SEGGER_SYSVIEW_X_GetTimestamp is always called when interrupts are
*   disabled. Therefore locking here is not required.
*/
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
  U32 TickCount;
  U32 Cycles;
  U32 CyclesPerTick;
  //
  // Get the cycles of the current system tick.
  // SysTick is down-counting, subtract the current value from the number of cycles per tick.
  //
  CyclesPerTick = SYST_RVR + 1;
  Cycles = (CyclesPerTick - SYST_CVR);
  //
  // Get the system tick count.
  //
  TickCount = SEGGER_SYSVIEW_TickCnt;
  //
  // If a SysTick interrupt is pending, re-read timer and adjust result
  //
  if ((SCB_ICSR & SCB_ICSR_PENDSTSET_MASK) != 0) {
    Cycles = (CyclesPerTick - SYST_CVR);
    TickCount++;
  }
  Cycles += TickCount * CyclesPerTick;

  return Cycles;
}
```

7.3.2 Cortex-M0 Interrupt ID

The currently active interrupt can be directly identified by reading the Cortex-M ICSR[5:0], which is the active vector field in the interrupt controller status register (ICSR).

Configuration:

```
//
// Get the interrupt Id by reading the Cortex-M ICSR[5:0]
//
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()   ((*(U32 *)(0xE000ED04)) & 0x3F)
```

7.3.3 Cortex-M0 SystemView lock and unlock

Locking and unlocking SystemView to prevent transferring records from being interrupted can be done by disabling interrupts.

Lock and unlock for SystemView and RTT can be the same.

Configuration:

```
//
// RTT locking for GCC toolchains in SEGGER_RTT_Conf.h
//
#define SEGGER_RTT_LOCK()       {                                           \
                                  unsigned int LockState;                   \
                                  __asm volatile ("mrs   %0, primask  \n\t" \
                                                  "mov   r1, $1     \n\t"   \
                                                  "msr   primask, r1  \n\t" \
                                                  : "=r" (LockState)        \
                                                  :                         \
                                                  : "r1"                    \
                                                  );

#define SEGGER_RTT_UNLOCK()       __asm volatile ("msr   primask, %0  \n\t" \
```



```
                                                  :                                \
                                                  : "r" (LockState)                \
                                                  :                                \
                                                  );                               \
                                  }

//
// Define SystemView locking in SEGGER_SYSVIEW_Conf.h
//
#define SEGGER_SYSVIEW_LOCK()   SEGGER_RTT_LOCK()
#define SEGGER_SYSVIEW_UNLOCK() SEGGER_RTT_UNLOCK()
```

7.3.4 Cortex-M0 Sample configuration

SEGGER_SYSVIEW_Conf.h

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
**********************************************************************
-------------------------- END-OF-HEADER -----------------------------

File        : SEGGER_SYSVIEW_Conf.h
Purpose     : SEGGER SysView configuration for Cortex-M0, Cortex-M0+,
              and Cortex-M1
*/

#ifndef SEGGER_SYSVIEW_CONF_H
#define SEGGER_SYSVIEW_CONF_H

/*********************************************************************
*
*       SysView timestamp configuration
*/
// Retrieve a system timestamp via user-defined function
#define SEGGER_SYSVIEW_GET_TIMESTAMP()      SEGGER_SYSVIEW_X_GetTimestamp()
// number of valid bits low-order delivered by SEGGER_SYSVIEW_X_GetTimestamp()
#define SEGGER_SYSVIEW_TIMESTAMP_BITS       32

/*********************************************************************
*
*       SysView Id configuration
*/
// Default value for the lowest Id reported by the application.
// Can be overridden by the application via SEGGER_SYSVIEW_SetRAMBase().
#define SEGGER_SYSVIEW_ID_BASE         0x20000000
// Number of bits to shift the Id to save bandwidth.
// (i.e. 2 when all reported Ids (pointers) are 4 byte aligned)
#define SEGGER_SYSVIEW_ID_SHIFT        0

/*********************************************************************
*
*       SysView interrupt configuration
*/
// Get the currently active interrupt Id. (read Cortex-M ICSR[8:0]
 = active vector)
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()   ((*(U32 *)(0xE000ED04)) & 0x3F)

/*********************************************************************
*
*       SysView locking
*/
// Lock SysView (nestable)
#define SEGGER_SYSVIEW_LOCK()             SEGGER_RTT_LOCK()
// Unlock SysView (nestable)
#define SEGGER_SYSVIEW_UNLOCK()           SEGGER_RTT_UNLOCK()
```



```
#endif

/*************************** End of file ****************************/
```

SEGGER_SYSVIEW_Config_embOS_CM0.c

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
*                       The Embedded Experts                         *
*                         www.segger.com                             *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File    : SEGGER_SYSVIEW_Config_embOS_CM0.c
Purpose : Sample setup configuration of SystemView with embOS
          on Cortex-M0/Cortex-M0+/Cortex-M1 systems which do not
          have a cycle counter.
Revision: $Rev: 3734 $

Additional information:
  SEGGER_SYSVIEW_TickCnt has to be defined in the module which handles
  the SysTick and must be incremented in the SysTick_Handler.

  This configuration can be adopted for any other OS and device.
*/
#include "RTOS.h"
#include "SEGGER_SYSVIEW.h"
#include "SEGGER_SYSVIEW_embOS.h"

//
// SystemcoreClock can be used in most CMSIS compatible projects.
// In non-CMSIS projects define SYSVIEW_CPU_FREQ directly.
//
extern unsigned int SystemCoreClock;

//
// SEGGER_SYSVIEW_TickCnt has to be defined in the module which
// handles the SysTick and must be incremented in the SysTick
// handler before any SYSVIEW event is generated.
//
// Example in embOS RTOSInit.c:
//
// unsigned int SEGGER_SYSVIEW_TickCnt; // <<-- Define SEGGER_SYSVIEW_TickCnt.
// void SysTick_Handler(void) {
// #if OS_PROFILE
//   SEGGER_SYSVIEW_TickCnt++;                  //
 <<-- Increment SEGGER_SYSVIEW_TickCnt before calling OS_EnterNestableInterrupt.
// #endif
//   OS_EnterNestableInterrupt();
//   OS_TICK_Handle();
//   OS_LeaveNestableInterrupt();
// }
//
extern unsigned int SEGGER_SYSVIEW_TickCnt;

/*********************************************************************
*
*       Defines, fixed
*
**********************************************************************
*/
#define SCB_ICSR
  (*(volatile U32*) (0xE000ED04uL)) // Interrupt Control State Register
#define SCB_ICSR_PENDSTSET_MASK     (1UL << 26)     // SysTick pending bit
#define SYST_RVR
  (*(volatile U32*) (0xE000E014uL)) // SysTick Reload Value Register
```



```
#define SYST_CVR
  (*(volatile U32*) (0xE000E018uL)) // SysTick Current Value Register

/*********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/
// The application name to be displayed in SystemViewer
#ifndef   SYSVIEW_APP_NAME
  #define SYSVIEW_APP_NAME          "Demo Application"
#endif

// The target device name
#ifndef   SYSVIEW_DEVICE_NAME
  #define SYSVIEW_DEVICE_NAME       "Cortex-M0"
#endif

// Frequency of the timestamp. Must match SEGGER_SYSVIEW_Conf.h
#ifndef   SYSVIEW_TIMESTAMP_FREQ
  #define SYSVIEW_TIMESTAMP_FREQ  (SystemCoreClock)
#endif

// System Frequency. SystemcoreClock is used in most CMSIS compatible projects.
#ifndef   SYSVIEW_CPU_FREQ
  #define SYSVIEW_CPU_FREQ        (SystemCoreClock)
#endif

// The lowest RAM address used for IDs (pointers)
#ifndef   SYSVIEW_RAM_BASE
  #define SYSVIEW_RAM_BASE        (0x20000000)
#endif

#ifndef   SYSVIEW_SYSDESC0
  #define SYSVIEW_SYSDESC0        "I#15=SysTick"
#endif

//#ifndef   SYSVIEW_SYSDESC1
//  #define SYSVIEW_SYSDESC1      ""
//#endif

//#ifndef   SYSVIEW_SYSDESC2
//  #define SYSVIEW_SYSDESC2      ""
//#endif

/*********************************************************************
*
*       _cbSendSystemDesc()
*
*  Function description
*    Sends SystemView description strings.
*/
static void _cbSendSystemDesc(void) {
  SEGGER_SYSVIEW_SendSysDesc("N="SYSVIEW_APP_NAME",O=embOS,D="SYSVIEW_DEVICE_NAME);
#ifdef SYSVIEW_SYSDESC0
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC0);
#endif
#ifdef SYSVIEW_SYSDESC1
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC1);
#endif
#ifdef SYSVIEW_SYSDESC2
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC2);
#endif
}

/*********************************************************************
*
```



```
*       Global functions
*
**********************************************************************
*/
void SEGGER_SYSVIEW_Conf(void) {
  SEGGER_SYSVIEW_Init(SYSVIEW_TIMESTAMP_FREQ, SYSVIEW_CPU_FREQ,
                      &SYSVIEW_X_OS_TraceAPI, _cbSendSystemDesc);
  SEGGER_SYSVIEW_SetRAMBase(SYSVIEW_RAM_BASE);
  OS_SetTraceAPI(&embOS_TraceAPI_SYSVIEW);    // Configure embOS to use SYSVIEW.
}

/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetTimestamp()
*
* Function description
*   Returns the current timestamp in ticks using the system tick
*   count and the SysTick counter.
*   All parameters of the SysTick have to be known and are set via
*   configuration defines on top of the file.
*
* Return value
*   The current timestamp.
*
* Additional information
*   SEGGER_SYSVIEW_X_GetTimestamp is always called when interrupts are
*   disabled. Therefore locking here is not required.
*/
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
  U32 TickCount;
  U32 Cycles;
  U32 CyclesPerTick;
  //
  // Get the cycles of the current system tick.
  // SysTick is down-counting, subtract the current value from the number of cycles per tick.
  //
  CyclesPerTick = SYST_RVR + 1;
  Cycles = (CyclesPerTick - SYST_CVR);
  //
  // Get the system tick count.
  //
  TickCount = SEGGER_SYSVIEW_TickCnt;
  //
  // If a SysTick interrupt is pending, re-read timer and adjust result
  //
  if ((SCB_ICSR & SCB_ICSR_PENDSTSET_MASK) != 0) {
    Cycles = (CyclesPerTick - SYST_CVR);
    TickCount++;
  }
  Cycles += TickCount * CyclesPerTick;

  return Cycles;
}

/************************** End of file ****************************/
```


7.4 Cortex-A / Cortex-R

Recording mode	Supported?
Continuous recording	Yes/NO
Single-shot recording	Yes
Post-mortem analysis	Yes

Continuous recording is only supported on Cortex-A / Cortex-R devices, which support RTT via background memory access via the AHB-AP. For more information please refer to the J-Link User Manual and website.

7.4.1 Cortex-A/R Event timestamp

The Cortex-A and Cortex-R cycle counter is implemented only as part of the Performance Monitor Extension and might not always be accessible. Cortex-A and Cortex-R do not have a generic system timer source, like the Cortex-M SysTick, either.

For an example on how to initialize the Performance counter, refer to *TI AM3358 Cortex-A8 sample configuration* on page 83.

Otherwise the event timestamp has to be provided by an application clock source. Refer to *Renesas RZA1 Cortex-A9 sample configuration* on page 80.

For the clock source any suitable timer can be used. It is recommended to use the OS system timer if possible, since it normally saves additional configuration and resource usage. If no timer is used in the application, a suitable timer has to be configured to be used with SystemView.

Some OSes implement API functions to get the OS time in cycles. If such a function is available it can be used directly or wrapped by `SEGGER_SYSVIEW_X_GetTimestamp()` If the OS does not provide functionality to retrieve the OS time in cycles, `SEGGER_SYSVIEW_X_GetTimestamp()` has to be implemented to get the timestamp from the timer.

- The timer should run at 1 MHz (1 tick/us) or faster.
- The timer should generate an interrupt on overflow or zero
- The timer should be in auto reload mode

Dummy configuration:

```
//
// SEGGER_SYSVIEW_TickCnt has to be defined in the module which
// handles interrupts and must be incremented in the interrupt
// handler as soon as the timer interrupt is acknowledged and
// before any SYSVIEW event is generated.
//
// Example:
//
// unsigned int SEGGER_SYSVIEW_TickCnt; // <<-- Define SEGGER_SYSVIEW_TickCnt.
// void OS_irq_handler(void) {
//   U32  InterruptId;
//   InterruptId  = INTC_ICCIAR & 0x3FF;  // read and extract the interrupt ID
//   if (InterruptId == TIMER_TICK_ID) {
//     SEGGER_SYSVIEW_TickCnt++;  // <<-- Increment SEGGER_SYSVIEW_TickCnt asap.
//   }
//   SEGGER_SYSVIEW_InterruptId
 = InterruptId; // Save active interrupt for SystemView event
//   SEGGER_SYSVIEW_RecordEnterISR();
//   //
//   // Handle interrupt, call ISR
//   //
//   SEGGER_SYSVIEW_RecordExitISR();
// }
//
```



```
extern unsigned int SEGGER_SYSVIEW_TickCnt;

/*********************************************************************
*
*       Defines, fixed
*
**********************************************************************
*/
//
// Define the required timer registers here.
//
#define TIMER_RELOAD_VALUE        /* as value which is used to initialize and
 reload the timer */
#define TIMER_COUNT               /* as timer register which holds the current
 counter value */
#define TIMER_INTERRUPT_PENDING() /* as check if a timer interrupt is pending */

/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetTimestamp()
*
* Function description
*   Returns the current timestamp in ticks using the system tick
*   count and the SysTick counter.
*   All parameters of the SysTick have to be known and are set via
*   configuration defines on top of the file.
*
* Return value
*   The current timestamp.
*
* Additional information
*   SEGGER_SYSVIEW_X_GetTimestamp is always called when interrupts are
*   disabled. Therefore locking here is not required.
*/
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
  U32 TickCount;
  U32 Cycles;
  U32 CyclesPerTick;
  //
  // Get the cycles of the current system tick.
  // Sample timer is down-counting,
  // subtract the current value from the number of cycles per tick.
  //
  CyclesPerTick = TIMER_RELOAD_VALUE + 1;
  Cycles = (CyclesPerTick - TIMER_COUNT);
  //
  // Get the system tick count.
  //
  TickCount = SEGGER_SYSVIEW_TickCnt;
  //
  // Check if a timer interrupt is pending
  //
  if (TIMER_INTERRUPT_PENDING()) {
    TickCount++;
    Cycles = (CyclesPerTick - TIMER_COUNT);
  }
  Cycles += TickCount * CyclesPerTick;

  return Cycles;
}
```

7.4.2 Cortex-A/R Interrupt ID

As the Cortex-A and Cortex-R core does not have an internal interrupt controller, retrieving the currently active interrrupt Id depends on the interrupt controller, which is used on the

target device. SEGGER_SYSVIEW_GET_INTERRUPT_ID() needs to be implemented to match this interrupt controller.

The configuration below shows how to get the interrupt Id on devices, which include the ARM Generic Interrupt Controller (GIC).

For other interrupt controllers the operation may vary. Refer to *TI AM3358 Cortex-A8 sample configuration* on page 83.

Since the active interrupt Id can only be retrieved from the GIC in connection with an acknowledge of the interrupt it can only be read once. Therefore the Id has to be stored in a variable when acknowledging it in the generic interrupt handler.

Dummy configuration:

```
//
// SEGGER_SYSVIEW_InterruptId has to be defined in the module which
// handles the interrupts and must be set to the acknowledged interrupt Id.
//
// Example:
//
// #define GIC_BASE_ADDR   /* as base address of the GIC on the device */
// #define GICC_BASE_ADDR  (GIC_BASE_ADDR + 0x2000u)
// #define GICC_IAR        (*(volatile unsigned*)(GICC_BASE_ADDR + 0x000C))
//
// unsigned int SEGGER_SYSVIEW_InterruptId; //
 <<-- Define SEGGER_SYSVIEW_InterruptId.
// void OS_irq_handler(void) {
//
//   int_id = GICC_IAR & 0x03FF;  // Read interrupt ID, acknowledge interrupt
//   SEGGER_SYSVIEW_InterruptId = iar_val;
//   OS_EnterInterrupt();         // Inform OS that interrupt handler is running
//   pISR();                      // Call interrupt service routine
//   OS_LeaveInterrupt();
  // Leave interrupt, perform task switch if required
// }
//
extern unsigned int SEGGER_SYSVIEW_InterruptId;

#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()  (SEGGER_SYSVIEW_InterruptId)
```

7.4.3 Cortex-A/R SystemView lock and unlock

As the Cortex-A and Cortex-R core does not have an internal interrupt controller, locking and unlocking SystemView to prevent transferring records from being interrupted can be done generic by disabling FIQ and IRQ completely, or by using interrupt controller specific methods. The configuration below shows how to disable all interrupts for RTT and SystemView.

Lock and unlock for SystemView and RTT can be the same.

Configuration:

```
//
// RTT locking for GCC toolchains in SEGGER_RTT_Conf.h
// Set and restore IRQ and FIQ mask bits.
//
#define SEGGER_RTT_LOCK()       {                                          \
                                  unsigned int LockState;                  \
                                  __asm volatile ("mrs  r1, CPSR      \n\t"     \
                                                  "mov  %0, r1        \n\t"     \
                                                  "orr  r1, r1, #0xC0 \n\t"     \
                                                  "msr  CPSR_c, r1    \n\t"     \
                                                  : "=r" (LockState)           \
                                                  :                            \
                                                  : "r1"                       \
```

```
                                                  );

#define SEGGER_RTT_UNLOCK()       __asm volatile ("mov  r0, %0        \n\t"     \
                                                  "mrs  r1, CPSR      \n\t"     \
                                                  "bic  r1, r1, #0xC0 \n\t"     \
                                                  "and  r0, r0, #0xC0 \n\t"     \
                                                  "orr  r1, r1, r0    \n\t"     \
                                                  "msr  CPSR_c, r1    \n\t"     \
                                                  :                             \
                                                  : "r" (LockState)             \
                                                  : "r0", "r1"                  \
                                                  );                            \
                                }

//
// Define SystemView locking in SEGGER_SYSVIEW_Conf.h
//
#define SEGGER_SYSVIEW_LOCK()   SEGGER_RTT_LOCK()
#define SEGGER_SYSVIEW_UNLOCK() SEGGER_RTT_UNLOCK()
```

7.4.4 Renesas RZA1 Cortex-A9 sample configuration

This sample configuration for the Renesas RZA1 (R7S72100) retrieves the currently active interrupt and the system tick counter from embOS.

It uses the OS Timer for timestamp generation. The RZA1 includes a GIC.

SEGGER_SYSVIEW_Conf.h

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
**********************************************************************
-------------------------- END-OF-HEADER -----------------------------

File        : SEGGER_SYSVIEW_Conf.h
Purpose     : SEGGER SysView configuration for Renesas RZA1 Cortex-A9
              with SEGGER embOS.
*/

#ifndef SEGGER_SYSVIEW_CONF_H
#define SEGGER_SYSVIEW_CONF_H

/*********************************************************************
*
*       SysView buffer configuration
*/
// Number of bytes that SysView uses for the buffer.
// Should be large enough for single-shot recording.
#define SEGGER_SYSVIEW_RTT_BUFFER_SIZE      1024 * 1024
// The RTT channel that SysView will use.
#define SEGGER_SYSVIEW_RTT_CHANNEL          1

/*********************************************************************
*
*       SysView timestamp configuration
*/
// Retrieve a system timestamp via OS-specific function
#define SEGGER_SYSVIEW_GET_TIMESTAMP()      SEGGER_SYSVIEW_X_GetTimestamp()
// number of valid bits low-order delivered by SEGGER_SYSVIEW_X_GetTimestamp()
#define SEGGER_SYSVIEW_TIMESTAMP_BITS       32

/*********************************************************************
*
*       SysView interrupt configuration
*/
//
```



```
// SEGGER_SYSVIEW_InterruptId has to be defined in the module which
// handles the interrupts and must be set to the acknowledged interrupt Id.
//
// Example:
//
// #define GIC_BASE_ADDR   /* as base address of the GIC on the device */
// #define GICC_BASE_ADDR  (GIC_BASE_ADDR + 0x2000u)
// #define GICC_IAR        (*(volatile unsigned*)(GICC_BASE_ADDR + 0x000C))
//
// unsigned int SEGGER_SYSVIEW_InterruptId; //
 <<-- Define SEGGER_SYSVIEW_InterruptId.
// void OS_irq_handler(void) {
//
//   int_id = GICC_IAR & 0x03FF;  // Read interrupt ID, acknowledge interrupt
//   SEGGER_SYSVIEW_InterruptId = iar_val;
//   OS_EnterInterrupt();         // Inform OS that interrupt handler is running
//   pISR();                      // Call interrupt service routine
//   OS_LeaveInterrupt();
  // Leave interrupt, perform task switch if required
// }
//
extern unsigned int SEGGER_SYSVIEW_InterruptId;

#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()  (SEGGER_SYSVIEW_InterruptId)

/*********************************************************************
*
*       SysView locking
*/
// Lock SysView (nestable)
#define SEGGER_SYSVIEW_LOCK()             SEGGER_RTT_LOCK()
// Unlock SysView (nestable)
#define SEGGER_SYSVIEW_UNLOCK()           SEGGER_RTT_UNLOCK()

#endif

/*************************** End of file ****************************/
```

SEGGER_SYSVIEW_Config_embOS_RZA1.c

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
**********************************************************************
-------------------------- END-OF-HEADER -----------------------------

File        : SEGGER_SYSVIEW_Config_embOS_RZA1.c
Purpose     : Sample setup configuration of SystemView with embOS
              for Renesas RZA1 Cortex-A9.
*/
#include "RTOS.h"
#include "SEGGER_SYSVIEW.h"
#include "SEGGER_SYSVIEW_embOS.h"

// SystemcoreClock can be used in most CMSIS compatible projects.
// In non-CMSIS projects define SYSVIEW_CPU_FREQ below.
extern unsigned int SystemCoreClock;

/*********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/
// The application name to be displayed in SystemView
#define SYSVIEW_APP_NAME        "embOS Demo Application"

// The target device name
```



```
#define SYSVIEW_DEVICE_NAME     "R7S72100"

// Frequency of the timestamp. Must match SEGGER_SYSVIEW_Conf.h
// and SEGGER_SYSVIEW_X_GetTimestamp().
#define SYSVIEW_TIMESTAMP_FREQ  (399900000u / 12)

// System Frequency. SystemcoreClock is used in most CMSIS compatible projects.
#define SYSVIEW_CPU_FREQ        (399900000u)

// The lowest RAM address used for IDs (pointers)
// Should be adjusted if the RAM does not start at 0x20000000.
#define SYSVIEW_RAM_BASE        (0x60020000)


#define TIMER_INTERRUPT_PENDING() /* as check if a timer interrupt is pending */


/*********************************************************************
*
*       _cbSendSystemDesc()
*
*  Function description
*    Sends SystemView description strings.
*/
static void _cbSendSystemDesc(void) {
  SEGGER_SYSVIEW_SendSysDesc("N="SYSVIEW_APP_NAME",D="SYSVIEW_DEVICE_NAME);
}

/*********************************************************************
*
*       Global functions
*
**********************************************************************
*/
void SEGGER_SYSVIEW_Conf(void) {
  SEGGER_SYSVIEW_Init(SYSVIEW_TIMESTAMP_FREQ, SYSVIEW_CPU_FREQ,
                      &SYSVIEW_X_OS_TraceAPI, _cbSendSystemDesc);
  SEGGER_SYSVIEW_SetRAMBase(SYSVIEW_RAM_BASE);
  OS_SetTraceAPI(&embOS_TraceAPI_SYSVIEW);    // Configure embOS to use SYSVIEW.
}

/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetTimestamp()
*
* Function description
*   Returns the current timestamp in ticks using the system tick
*   count and the SysTick counter.
*   All parameters of the SysTick have to be known and are set via
*   configuration defines on top of the file.
*
* Return value
*   The current timestamp.
*
* Additional information
*   SEGGER_SYSVIEW_X_GetTimestamp is always called when interrupts are
*   disabled. Therefore locking here is not required.
*/
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
  U32 TickCount;
  U32 Cycles;
  U32 CyclesPerTick;
  //
  // Get the cycles of the current system tick.
  // Sample timer is down-counting,
  // subtract the current value from the number of cycles per tick.
  //
  CyclesPerTick = 33249 + 1;
```



```
  Cycles = (CyclesPerTick - OSTM_CNT);
  //
  // Get the system tick count.
  //
  TickCount = SEGGER_SYSVIEW_TickCnt;
  //
  // Check if a timer interrupt is pending
  //
  if (TIMER_INTERRUPT_PENDING()) {
    TickCount++;
    Cycles = (CyclesPerTick - OSTM_CNT);
  }
  Cycles += TickCount * CyclesPerTick;

  return Cycles;
}

/*************************** End of file ****************************/
```

7.4.5 TI AM3358 Cortex-A8 sample configuration

This sample configuration for the TI AM3358 retrieves the currently active interrupt directly. It initializes and uses the Cortex-A performance counter for timestamp generation.

The SystemView timestmap generation can be used for other Cortex-A devices, which include the performance counter unit.

SEGGER_SYSVIEW_Conf.h

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
**********************************************************************
-------------------------- END-OF-HEADER -----------------------------

File        : SEGGER_SYSVIEW_Conf.h
Purpose     : Generic SEGGER SysView configuration for non-Cortex-M
              devices.
*/

#ifndef SEGGER_SYSVIEW_CONF_H
#define SEGGER_SYSVIEW_CONF_H

/*********************************************************************
*
*       SysView timestamp configuration
*/
// Retrieve a system timestamp via user-defined function
#define SEGGER_SYSVIEW_GET_TIMESTAMP()      SEGGER_SYSVIEW_X_GetTimestamp()
// number of valid bits low-order delivered by SEGGER_SYSVIEW_X_GetTimestamp()
#define SEGGER_SYSVIEW_TIMESTAMP_BITS       32

/*********************************************************************
*
*       SysView Id configuration
*/
// Default value for the lowest Id reported by the application.
// Can be overridden by the application via SEGGER_SYSVIEW_SetRAMBase().
#define SEGGER_SYSVIEW_ID_BASE         0
// Number of bits to shift the Id to save bandwidth.
// (i.e. 2 when all reported Ids (pointers) are 4 byte aligned)
#define SEGGER_SYSVIEW_ID_SHIFT        0

/*********************************************************************
*
*       SysView interrupt configuration
*/
```



```
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()   SEGGER_SYSVIEW_X_GetInterruptId()

/*********************************************************************
*
*       SysView locking
*/
// Lock SysView (nestable)
#define SEGGER_SYSVIEW_LOCK()              SEGGER_RTT_LOCK()
// Unlock SysView (nestable)
#define SEGGER_SYSVIEW_UNLOCK()            SEGGER_RTT_UNLOCK()

#endif

/************************** End of file ****************************/
```

SEGGER_SYSVIEW_Config_embOS_AM3358.c

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
**********************************************************************
-------------------------- END-OF-HEADER -----------------------------

File        : SEGGER_SYSVIEW_Config_embOS_RZA1.c
Purpose     : Sample setup configuration of SystemView with embOS
              for TI AM3358 Cortex-A8.
*/
#include "RTOS.h"
#include "SEGGER_SYSVIEW.h"
#include "SEGGER_SYSVIEW_embOS.h"

//
// SystemcoreClock can be used in most CMSIS compatible projects.
// In non-CMSIS projects define SYSVIEW_CPU_FREQ directly.
//
extern unsigned int SystemCoreClock;

/*********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/
// The application name to be displayed in SystemView
#ifndef   SYSVIEW_APP_NAME
  #define SYSVIEW_APP_NAME        "embOS start project"
#endif

// The target device name
#ifndef   SYSVIEW_DEVICE_NAME
  #define SYSVIEW_DEVICE_NAME     "AM3358"
#endif

// Frequency of the timestamp. Must match SEGGER_SYSVIEW_Conf.h
// The performance counter frequency equals the core clock frequency.
#define SYSVIEW_TIMESTAMP_FREQ  (SystemCoreClock)

// System Frequency. SystemcoreClock is used in most CMSIS compatible projects.
#ifndef   SYSVIEW_CPU_FREQ
  #define SYSVIEW_CPU_FREQ        (SystemCoreClock)
#endif

// The lowest RAM address used for IDs (pointers)
#ifndef   SYSVIEW_RAM_BASE
  #define SYSVIEW_RAM_BASE        (0x80000000)
#endif

#ifndef   SYSVIEW_SYSDESC0
```



```
  #define SYSVIEW_SYSDESC0        "I#67=SysTick,I#18=USB,I#17=USBSS,I#36=LCDC"
#endif

#define INTC_BASE                 (0x48200000uL)
#define INTC_SIR_IRQ              (*(volatile U32*) (INTC_BASE + 0x40uL))

/*********************************************************************
*
*       Local functions
*
**********************************************************************
*/
/*********************************************************************
*
*       _cbSendSystemDesc()
*
*  Function description
*    Sends SystemView description strings.
*/
static void _cbSendSystemDesc(void) {
  SEGGER_SYSVIEW_SendSysDesc("N="SYSVIEW_APP_NAME",O=embOS,D="SYSVIEW_DEVICE_NAME);
#ifdef SYSVIEW_SYSDESC0
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC0);
#endif
#ifdef SYSVIEW_SYSDESC1
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC1);
#endif
#ifdef SYSVIEW_SYSDESC2
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC2);
#endif
}

/*********************************************************************
*
*       _InitPerformanceCounter
*
*  Function description
*    Initialize the internal Cortex-A Performance counter.
*    The function will work for Cortex-A8, Cortex-A9.
*    Please check whether this also suites for your core.
*/
static void _InitPerformanceCounter(U32 PerformReset, I32 UseDivider) {
 //
 // in general enable all counters (including cycle counter)
 //
 I32 Value = 1;

 //
 // Peform reset:
 //
 if (PerformReset) {
   Value |= 2;     // reset all counters to zero.
   Value |= 4;     // reset cycle counter to zero.
 }

 if (UseDivider) {
   Value |= 8;     // enable "by 64" divider for CCNT.
 }
 Value |= 16;

 // program the performance-counter control-register:
 __asm volatile ("MCR p15, 0, %0, c9, c12, 0\t\n"
                 :              // Output result
                 : "r"(Value)   // Input
                 :              // Clobbered list
                );
 //
 // Enable all counters
```



```
  //
  __asm volatile ("MCR p15, 0, %0, c9, c12, 1\t\n"
                  :                   // Output result
                  : "r"(0x8000000f)   // Input
                  :                   // Clobbered list
                 );
  //
  // Clear overflows
  //
  __asm volatile ("MCR p15, 0, %0, c9, c12, 3\t\n"
                  :                   // Output result
                  : "r"(0x8000000f)   // Input
                  :                   // Clobbered list
                 );
}

/*********************************************************************
*
*       Global functions
*
**********************************************************************
*/

/*********************************************************************
*
*       SEGGER_SYSVIEW_Conf
*
*  Function description
*    Configures SYSVIEW.
*
*    Please check whether this also suites for your core.
*/
void SEGGER_SYSVIEW_Conf(void) {
  //
  // Write USEREN Register
  //
  __asm volatile ("MCR p15, 0, %0, C9, C14, 0\n\t"
                  :               // Output result
                  : "r"(1)        // Input
                  :               // Clobbered list
                 );

  //
  // Disable counter overflow interrupts
  //
  __asm volatile ("MCR p15, 0, %0, C9, C14, 2\n\t"
                  :                   // Output result
                  : "r"(0x8000000f)   // Input
                  :                   // Clobbered list
                 );
  _InitPerformanceCounter(1, 0);
  SEGGER_SYSVIEW_Init(SYSVIEW_TIMESTAMP_FREQ, SYSVIEW_CPU_FREQ,
                      &SYSVIEW_X_OS_TraceAPI, _cbSendSystemDesc);
  SEGGER_SYSVIEW_SetRAMBase(SYSVIEW_RAM_BASE);
  OS_SetTraceAPI(&embOS_TraceAPI_SYSVIEW);    // Configure embOS to use SYSVIEW.
}

/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetTimestamp()
*
* Function description
*   Returns the current timestamp in ticks using the performance counter.
*
* Return value
*   The current timestamp.
*
* Additional information
```



```
*   SEGGER_SYSVIEW_X_GetTimestamp is always called when interrupts are
*   disabled. Therefore locking here is not required.
*/
U32  SEGGER_SYSVIEW_X_GetTimestamp(void) {
  register U32 r = 0;
  //
  // Read CCNT Register
  //
  __asm volatile ("MRC p15, 0, %0, c9, c13, 0"
                  : "+r"(r)   // Output result
                  :           // Inputs
                  : );        // Clobbered list
  return r;
}

/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetInterruptId()
*
*  Function description
*    Return the currently active IRQ interrupt number
*    from the INTC_SIR_IRQ.
*/
U32 SEGGER_SYSVIEW_X_GetInterruptId(void) {
  return (INTC_SIR_IRQ & (0x7Fu)); // INTC_SIR_IRQ[6:0]: ActiveIRQ
}
```

7.5 Renesas RX

Recording mode	Supported?
Continuous recording	Yes
Single-shot recording	Yes
Post-mortem analysis	Yes

7.5.1 Renesas RX Event timestamp

The event timestamp has to be provided by an application clock source timer. `SEGGER_SYSVIEW_X_GetTimestamp()` can be used to implement the functionality.

Before creating any other event in the timer interrupt, the interrupt handler should increment `SEGGER_SYSVIEW_TickCnt`.

Configuration:

```
//
// SEGGER_SYSVIEW_TickCnt has to be defined in the module which
// handles the system tick timer and must be incremented in the timer interrupt
// handler before any SYSVIEW event is generated.
//
// Example in embOS RTOSInit.c:
//
// unsigned int SEGGER_SYSVIEW_TickCnt; // <<-- Define SEGGER_SYSVIEW_TickCnt.
// void SysTick_Handler(void) {
// #if OS_PROFILE
//   SEGGER_SYSVIEW_TickCnt++;  // <<-- Increment SEGGER_SYSVIEW_TickCnt asap.
// #endif
//   OS_EnterNestableInterrupt();
//   OS_TICK_Handle();
//   OS_LeaveNestableInterrupt();
// }
//
extern unsigned int SEGGER_SYSVIEW_TickCnt;

/*********************************************************************
*
*       Defines, fixed
*
**********************************************************************
*/
// System Timer configuration
#define IRR_BASE_ADDR        (0x00087000u)
#define CMT0_VECT            28u
#define OS_TIMER_VECT        CMT0_VECT
#define TIMER_PRESCALE       (8u)
#define CMT0_BASE_ADDR       (0x00088000u)
#define CMT0_CMCNT           (*(volatile U16*) (CMT0_BASE_ADDR + 0x04u))

/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetTimestamp()
*
* Function description
*   Returns the current timestamp in ticks using the system tick
*   count and the system timer counter.
*   All parameters of the system timer have to be known and are set via
*   configuration defines on top of the file.
*
* Return value
*   The current timestamp.
*
* Additional information
```



```
*   SEGGER_SYSVIEW_X_GetTimestamp is always called when interrupts are
*   disabled. Therefore locking here is not required.
*/
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
  U32 Time;
  U32 Cnt;

  Time = SEGGER_SYSVIEW_TickCnt;
  Cnt  = CMT0_CMCNT;
  //
  // Check if timer interrupt pending ...
  //
  if ((*(volatile U8*)(IRR_BASE_ADDR + OS_TIMER_VECT) & (1u << 0u)) != 0u) {
    Cnt = CMT0_CMCNT;      // Interrupt pending, re-read timer and adjust result
    Time++;
  }
  return ((SYSVIEW_TIMESTAMP_FREQ/1000) * Time) + Cnt;
}
```

7.5.2 Renesas RX Interrupt ID

The currently active interrupt level can be used as the interrupt ID on RX devices. In the sample configuration it is provided by `SEGGER_SYSVIEW_X_GetInterruptId()` in `SEGGER_SYSVIEW_Config_[System]_RX.c`.

Configuration:

```
//
// Get the interrupt Id via user-provided function
//
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()   SEGGER_SYSVIEW_X_GetInterruptId()
```

7.5.3 Renesas RX SystemView lock and unlock

Locking and unlocking SystemView to prevent transferring records from being interrupted can be done by disabling interrupts.

Lock and unlock for SystemView and RTT can be the same.

Configuration:

```
//
// RTT locking for IAR toolchains in SEGGER_RTT_Conf.h
//
  #define SEGGER_RTT_LOCK()   {                                    \
                                unsigned long LockState;           \
                                LockState = __get_interrupt_state(); \
                                __disable_interrupt();

  #define SEGGER_RTT_UNLOCK()   __set_interrupt_state(LockState);     \
                              }

//
// Define SystemView locking in SEGGER_SYSVIEW_Conf.h
//
#define SEGGER_SYSVIEW_LOCK()   SEGGER_RTT_LOCK()
#define SEGGER_SYSVIEW_UNLOCK() SEGGER_RTT_UNLOCK()
```

7.5.4 Renesas RX Sample configuration

SEGGER_SYSVIEW_Conf.h

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
**********************************************************************
-------------------------- END-OF-HEADER -----------------------------

File        : SEGGER_SYSVIEW_Conf.h
Purpose     : SEGGER SysView configuration for Renesas RX
*/

#ifndef SEGGER_SYSVIEW_CONF_H
#define SEGGER_SYSVIEW_CONF_H

/*********************************************************************
*
*       SysView timestamp configuration
*/
// Retrieve a system timestamp via user-defined function
#define SEGGER_SYSVIEW_GET_TIMESTAMP()      SEGGER_SYSVIEW_X_GetTimestamp()
// number of valid bits low-order delivered by SEGGER_SYSVIEW_X_GetTimestamp()
#define SEGGER_SYSVIEW_TIMESTAMP_BITS       32

/*********************************************************************
*
*       SysView Id configuration
*/
// Default value for the lowest Id reported by the application.
// Can be overridden by the application via SEGGER_SYSVIEW_SetRAMBase().
#define SEGGER_SYSVIEW_ID_BASE         0
// Number of bits to shift the Id to save bandwidth.
// (i.e. 2 when all reported Ids (pointers) are 4 byte aligned)
#define SEGGER_SYSVIEW_ID_SHIFT        0

/*********************************************************************
*
*       SysView interrupt configuration
*/
// Get the currently active interrupt Id. (read Cortex-M ICSR[8:0]
 = active vector)
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()   SEGGER_SYSVIEW_X_GetInterruptId()

/*********************************************************************
*
*       SysView locking
*/
// Lock SysView (nestable)
#define SEGGER_SYSVIEW_LOCK()             SEGGER_RTT_LOCK()
// Unlock SysView (nestable)
#define SEGGER_SYSVIEW_UNLOCK()           SEGGER_RTT_UNLOCK()

#endif

/************************** End of file ******************************/
```

SEGGER_SYSVIEW_Config_embOS_CM0.c

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
*                       The Embedded Experts                         *
*                          www.segger.com                            *
**********************************************************************
```



```
------------------------- END-OF-HEADER -----------------------------

File    : SEGGER_SYSVIEW_Config_NoOS_RX.c
Purpose : Sample setup configuration of SystemView on Renesas RX
          systems without an operating system.
Revision: $Rev: 3734 $
*/
#include "RTOS.h"
#include "SEGGER_SYSVIEW.h"
#include "SEGGER_SYSVIEW_embOS.h"

//
// SystemcoreClock can be used in most CMSIS compatible projects.
// In non-CMSIS projects define SYSVIEW_CPU_FREQ directly.
//
extern unsigned int SystemCoreClock;

/*********************************************************************
*
*       Defines, fixed
*
**********************************************************************
*/

/*********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/
// The application name to be displayed in SystemViewer
#ifndef   SYSVIEW_APP_NAME
  #define SYSVIEW_APP_NAME          "Demo Application"
#endif

// The target device name
#ifndef   SYSVIEW_DEVICE_NAME
  #define SYSVIEW_DEVICE_NAME       "RX64M"
#endif

// System Frequency. SystemcoreClock is used in most CMSIS compatible projects.
#ifndef   SYSVIEW_CPU_FREQ
  #define SYSVIEW_CPU_FREQ        (SystemCoreClock)
#endif

// Frequency of the timestamp. Must match SEGGER_SYSVIEW_Conf.h and RTOSInit.c
#ifndef   SYSVIEW_TIMESTAMP_FREQ
  #define SYSVIEW_TIMESTAMP_FREQ
  (SYSVIEW_CPU_FREQ/2u/8u) // Assume system timer runs at
 1/16th of the CPU frequency
#endif

// The lowest RAM address used for IDs (pointers)
#ifndef   SYSVIEW_RAM_BASE
  #define SYSVIEW_RAM_BASE        (0)
#endif

#ifndef   SYSVIEW_SYSDESC0
  #define SYSVIEW_SYSDESC0
  "I#0=IntPrio0,I#1=IntPrio1,I#2=IntPrio2,I#3=IntPrio3,I#4=IntPrio4"
#endif

//#ifndef   SYSVIEW_SYSDESC1
//  #define SYSVIEW_SYSDESC1
 "I#5=IntPrio5,I#6=IntPrio6,I#7=IntPrio7,I#8=IntPrio8,I#9=IntPrio9,I#10=IntPrio10"
//#endif

//#ifndef   SYSVIEW_SYSDESC2
```

```
//  #define SYSVIEW_SYSDESC2
 "I#11=IntPrio11,I#12=IntPrio12,I#13=IntPrio13,I#14=IntPrio14,I#15=IntPrio15"
//#endif

// System Timer configuration
#define IRR_BASE_ADDR        (0x00087000u)
#define CMT0_VECT            28u
#define OS_TIMER_VECT        CMT0_VECT
#define TIMER_PRESCALE       (8u)
#define CMT0_BASE_ADDR       (0x00088000u)
#define CMT0_CMCNT           (*(volatile U16*) (CMT0_BASE_ADDR + 0x04u))

extern unsigned SEGGER_SYSVIEW_TickCnt;
  // Tick Counter value incremented in the tick handler.

/*********************************************************************
*
*       _cbSendSystemDesc()
*
*  Function description
*    Sends SystemView description strings.
*/
static void _cbSendSystemDesc(void) {
  SEGGER_SYSVIEW_SendSysDesc("N="SYSVIEW_APP_NAME",D="SYSVIEW_DEVICE_NAME);
#ifdef SYSVIEW_SYSDESC0
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC0);
#endif
#ifdef SYSVIEW_SYSDESC1
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC1);
#endif
#ifdef SYSVIEW_SYSDESC2
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC2);
#endif
}

/*********************************************************************
*
*       Global functions
*
**********************************************************************
*/
void SEGGER_SYSVIEW_Conf(void) {
  SEGGER_SYSVIEW_Init(SYSVIEW_TIMESTAMP_FREQ, SYSVIEW_CPU_FREQ,
                      0, _cbSendSystemDesc);
  SEGGER_SYSVIEW_SetRAMBase(SYSVIEW_RAM_BASE);
}

/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetTimestamp()
*
* Function description
*   Returns the current timestamp in ticks using the system tick
*   count and the SysTick counter.
*   All parameters of the SysTick have to be known and are set via
*   configuration defines on top of the file.
*
* Return value
*   The current timestamp.
*
* Additional information
*   SEGGER_SYSVIEW_X_GetTimestamp is always called when interrupts are
*   disabled.
*   Therefore locking here is not required and OS_GetTime_Cycles() may
*   be called.
*/
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
  U32 Time;
```



```
  U32 Cnt;

  Time = SEGGER_SYSVIEW_TickCnt;
  Cnt  = CMT0_CMCNT;
  //
  // Check if timer interrupt pending ...
  //
  if ((*(volatile U8*)(IRR_BASE_ADDR + OS_TIMER_VECT) & (1u << 0u)) != 0u) {
    Cnt = CMT0_CMCNT;      // Interrupt pending, re-read timer and adjust result
    Time++;
  }
  return ((SYSVIEW_TIMESTAMP_FREQ/1000) * Time) + Cnt;
}

/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetInterruptId()
*
*  Function description
*    Return the priority of the currently active interrupt.
*/
U32 SEGGER_SYSVIEW_X_GetInterruptId(void) {
  U32 IntId;
 __asm volatile ("mvfc    PSW, %0           \t\n" // Load current PSW
                 "and     #0x0F000000, %0   \t\n" // Clear all except IPL
 ([27:24])
                 "shlr    #24, %0           \t\n" // Shift IPL to [3:0]
                 : "=r" (IntId)                   // Output result
                 :                                // Input
                 :                                // Clobbered list
                );
  return IntId;
}

/************************** End of file ****************************/
```

7.6 Other CPUs

Recording mode	Supported?
Continuous recording	No
Single-shot recording	Yes
Post-mortem analysis	Yes

On CPUs, which are not covered by the sections above SystemView can be used in single-shot mode, too.

To properly run SystemView the same items have to be configured:

- Get an event timestamp.
- Get an interrupt Id of the active interrupt.
- Lock and unlock SystemView to prevent recording being interrupted.

Chapter 8

Supported OSes

The following chapter describes which (RT)OSes are already instrumented to use SystemView and how to configure them.

8.1 embOS

SEGGER embOS (V4.12a and later) can generate trace events for SystemView and other recording implementations when profiling is enabled.

8.1.1 Configuring embOS for SystemView

Profiling is enabled in the `OS_LIBMODE_SP`, `OS_LIBMODE_DP` and `OS_LIBMODE_DT` embOS library configurations (For detailed information refer to the embOS User Manual UM01001).

In addition to the SYSTEMVIEW and RTT core module, the following file needs to be included in the application:

For Cortex-M3 and Cortex-M4 targets include `SEGGER_SYSVIEW_Config_embOS.c`. For Cortex-M0 and Cortex-M1 targets include `SEGGER_SYSVIEW_Config_embOS.c`.

This file provides additionally required functions for SystemView and allows configuration to fit the target system, like defines for the application name, the target device and the target core frequency. It initializes the SYSTEMVIEW module and configures embOS to send trace events to SYSTEMVIEW. For an example configuration, refer to *The SystemView system information config* on page 52.

At the start of the application, at main, after the target is initialized, `SEGGER_SYSVIEW_Conf()` has to be called to enable SystemView.

Now, when the application is running, SystemView can connect to the target and start recording events. All task, interrupt, and OS Scheduler activity, as well as embOS API calls are recorded when SystemView is connected or `SEGGER_SYSVIEW_Start()` has been called.

8.2 uC/OS-III

SystemView can be used with Micrium's uC/OS-III to record task, interrupt, and scheduler activity.

8.2.1 Configuring uC/OS-III for SystemView

In addition to the SYSTEMVIEW and RTT core module the following files have to be included in the application project:

`SEGGER_SYSVIEW_Config_uCOSIII.c` provides additionally required functions for SystemView and allows configuration to fit the target system, like defines for the application name, the target device and the target core frequency. The example configuration file, shipped with the SystemView package is configured to be used with most Cortex-M3, Cortex-M4, and Cortex-M7 targets. For an example configuration, refer to *The SystemView system information config* on page 52.

`SEGGER_SYSVIEW_uCOSIII.c` and `os_trace.h` provide the interface between uC/OS-III and SystemView. They usually do not need to be modified.

`os_cfg_trace.h` is the minimal uc/OS-III Trace configuration file required for SystemView. If the project already includes this file, make sure the content fits the application. This file includes two defines to configure the maximum number of tasks and the maximum number of resources to be managed and named in the SystemView recording.

```
#define  TRACE_CFG_MAX_TASK                  16u
#define  TRACE_CFG_MAX_RESOURCES             16u
```

Enable recording

Recording of uC/OS-III events can be configured in `os_cfg.h`.

Define `OS_CFG_TRACE_EN` as `1u` to enable basic recording.

When `OS_CFG_TRACE_API_ENTER_EN` is defined as `1u`, API function calls will be recorded, too.

To also record when an API function exits, define `OS_CFG_TRACE_API_EXIT_EN` as `1u` as well.

Call `TRACE_INIT()` at the beginning of the application, after the system has been initialized:

```
  [...]
    BSP_Init(); /* Initialize BSP functions       */
    CPU_Init(); /* Initialize the uC/CPU services */

#if (defined(OS_CFG_TRACE_EN) && (OS_CFG_TRACE_EN > 0u))
    /* Initialize uC/OS-III Trace. Should be called after initializing the
 system. */
    TRACE_INIT();
#endif
  [...]
```

8.3 FreeRTOS

FreeRTOS can also generate trace events for SystemView and allows basic but useful analysis without modification.

For more detailed analysis, like Scheduler activity and interrupts, the FreeRTOS source and the used port have to be slightly modified.

8.3.1 Configuring FreeRTOS for SystemView

In addition to the SYSTEMVIEW and RTT core module, `SEGGER_SYSVIEW_Config_FreeRTOS.c` needs to be included in the application. This file provides additionally required functions for SystemView and allows configuration to fit the target system, like defines for the application name, the target device and the target core frequency. For an example configuration, refer to *The SystemView system information config* on page 52.

The `SEGGER_SYSVIEW_FreeRTOS.h` header has to be included at the end of `FreeRTOSConfig.h` or above every include of `FreeRTOS.h`. It defines the trace macros to create SYSTEMVIEW events..

To get the best results `INCLUDE_xTaskGetIdleTaskHandle` and `INCLUDE_pxTaskGetStackStart` should be defined as 1 in `FreeRTOSConfig.h`.

The patch file `Sample.2.3_Core.patch` shows the required modifications of the FreeRTOS 8.2.3 source and the GCC/`ARM_CM4F` port. It can be used as a reference when using another version or port of FreeRTOS. I.e. if another port than GCC/`ARM_CM4F` is used, the `traceISR_ENTER()`, `traceISR_EXIT()`, and `traceISR_EXIT_TO_SCHEDULER()` calls have to be added accordingly.

8.4 Other OSes

Other OSes are not officially instrumented, yet.

If you want to use SystemView with an other OS, get it touch with SEGGER or the OS Vendor. The OS instrumentation can also be done with the guide in the following chapter.

8.5 No OS

SystemView can be used without any instrumented OS at all, to record interrupt activity and user events.

8.5.1 Configuring a system for SystemView

In addition to the SYSTEMVIEW and RTT core module, `SEGGER_SYSVIEW_Config_NoOS.c` needs to be included in the application. This file provides the basic configuration of the required functions for SystemView and can be modified to fit the system. For an example configuration, refer to *The SystemView system information config* on page 52. An additional `SEGGER_SYSVIEW_OS_API` pointer can be passed in `SEGGER_SYSVIEW_Init` to provide information about the system time or "tasks" of the system.

For a description on how to record interrupts in the system, refer to *Recording interrupts* on page 109.

Chapter 9

Performance and resource usage

This chapter covers the performance and resource usage of SystemView. It contains information about the memory requirements in typical systems which can be used to obtain sufficient estimates for most target systems.

9.1 Memory requirements

The memory requirements may differ, depending on the used OS integration, the target configuration and the compiler optimizations.

To achieve a balanced result of performance and memory usage, it is recommended to set the compiler optimization level for the SystemView and RTT module accordingly. Compiler optimizations should always be switched on for the SystemView and RTT module - even in Debug configuration builds.

9.1.1 ROM usage

The following table lists the ROM usage of SystemView by component. With a smart linker only the used functions will be included in the application.

Description	ROM
Minimum core code required when using SystemView	~920 Byte
Basic SystemView recording functions for application, OS and module events	~380 Byte
OS-related SystemView recording functions	~360 Byte
Middleware module-related recording functions	~120 Byte
Complete SystemView Module	*~1.8 KByte*

The following table list the ROM usage of SystemView with different configurations.

Description	Configuration	ROM
SystemView Module	Balanced optimization, no static buffer	~1.8 KByte
SystemView Module	Balanced optimization, static buffer	~2.1 KByte
SystemView Module	Balanced optimization, no static buffer, post-mortem mode	~1.4 KByte
SystemView Module	Balanced optimization, static buffer, post-mortem mode	~1.7 KByte
RTT Module	Balanced optimization	~0.5 KByte

9.1.2 Static RAM usage

The following table list the static RAM usage of SystemView with different configurations.

Description	Configuration	RAM
SystemView Module	No static buffer	~70 Byte + Channel Buffer
SystemView Module	Static buffer	~280 Byte + Channel Buffer
SystemView Module	No static buffer, post-mortem mode	~60 Byte + Channel Buffer
SystemView Module	Static buffer, post-mortem mode	~180 Byte + Channel Buffer
RTT Module		~30 Byte + Channel Buffer

9.1.3 Stack RAM usage

SystemView requires stack to record events in every context, which might record events in the application. This typically includes the system stack used by the scheduler, the interrupt stack and the task stacks.

Since SystemView handles incoming requests for the system description and task information, there must be enough free space on the stack to record an event and to send the system description, which is recording another event.

SystemView can be configured to select between lower stack usage or less static RAM use.

Description	Maximum Stack
Static buffer for event generation and encoding	~230 Bytes
Stack buffer for event generation and encoding	~510 Bytes
Static buffer for event generation and encoding, post-mortem mode	~150 Bytes
Stack buffer for event generation and encoding, post-mortem mode	~280 Bytes

Chapter 10

Integration guide

This section describes how to integrate SEGGER SystemView into an OS or middleware module to be able to record its execution.

10.1 Integrating SEGGER SystemView into an OS

SEGGER SystemView can be integrated in any (RT)OS to get information about task execution, OS internal activity, like a scheduler, and OS API calls. For the following RTOSes this integration has already been done and can be used out-of-the box.

- SEGGER embOS (V4.12a or later)
- Micrium uC/OS-III (Upcoming V3.06)
- FreeRTOS (Tested with V8.2.3)

For integration into other OSes, contact the OS distributor or do the integration following the instructions in this sections.

The examples in this section are pseudo-code to illustrate when to call specific SystemView functions. To allow general integration of trace instrumentation tools calls to these functions can also be integrated as function macros or via a configurable trace API.

Instrumenting the OS core

In order to be able to record task execution and context switches, the OS core has to be instrumented to generate SystemView events at the appropriate core functions.

Interrupt execution is in most cases handled by the OS, too. This allows instrumenting the according OS functions called on enter and exit interrupt, which would otherwise have to be done for each ISR in the application.

The third aspect of instrumenting the OS core is to provide run-time information for a more detailed analysis. This information includes the system time to allow SystemView to display timestamps relative to the start of the application, instead of to the start of recording, and the task list, which is used by SystemView to display task names, stack information and to order tasks by priority.

10.1.1 Recording task activity

SystemView can record a set of pre-defined system events for the main information of system and OS activity, like task execution. These events should be generated by the OS in the corresponding functions.

The pre-defined events are:

Event	Description	SystemView API
Task Create	A new task is created.	*SEGGER_SYSVIEW_OnTaskCreate* on page 139
Task Start Ready	A task is marked as ready to start or resume execution.	*SEGGER_SYSVIEW_OnTaskStartReady* on page 141
Task Start Exec	A task is activated, it starts or resumes execution.	*SEGGER_SYSVIEW_OnTaskStartExec* on page 140
Task Stop Ready	A task is blocked or suspended.	*SEGGER_SYSVIEW_OnTaskStopReady* on page 143
Task Stop Exec	A task terminates.	*SEGGER_SYSVIEW_OnTaskStopExec* on page 142
System Idle	No task is executing, the system goes into Idle state.	*SEGGER_SYSVIEW_OnIdle* on page 138

10.1.1.1 Task Create

A new task is created.

Task Create events happen when a task is created by the system.

On Task Create events call SEGGER_SYSVIEW_OnTaskCreate() with the Id of the new task. Additionally it is recommended to record the task information of the new task with SEGGER_SYSVIEW_SendTaskInfo().

Example

```
void OS_CreateTask(TaskFunc* pF, unsigned Prio, const char* sName, void* pStack) {
  SEGGER_SYSVIEW_TASKINFO Info;
  OS_TASK*                pTask;  // Pseudo struct to be replaced

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskCreate((unsigned)pTask);
  memset(&Info, 0, sizeof(Info));
  //
  // Fill elements with current task information
  //
  Info.TaskID     = (U32)pTask;
  Info.sName      = pTask->Name;
  Info.Prio       = pTask->Priority;
  Info.StackBase  = (U32)pTask->pStack;
  Info.StackSize  = pTask->StackSize;
  SEGGER_SYSVIEW_SendTaskInfo(&Info);
}
```

10.1.1.2 Task Start Ready

A task is marked as ready to start or resume execution.

Task Start Ready events can for example happen, when the delay time of the task expired, or when a resource the task was waiting for is available, or when an event was triggered.

On Task Start Ready events call SEGGER_SYSVIEW_OnTaskStartReady() with the Id of the task which has become ready.

Example

```
int OS_HandleTick(void) {
  int TaskReady = 0;   // Pseudo variable indicating a task is ready

  [OS specific code ...]

  if (TaskReady) {
    SEGGER_SYSVIEW_OnTaskStartReady((unsigned)pTask);
  }
}
```

10.1.1.3 Task Start Exec

A task is activated, it starts or resumes execution.

Task Start Exec events happen when the context is about to be switched to the activated task. This is normally done by the Scheduler when there is a ready task.

On Task Start Exec events call SEGGER_SYSVIEW_OnTaskStartExec() with the Id of the task which will execute.

Example

```
void OS_Switch(void) {

  [OS specific code ...]

  //
  // If a task is activated
```



```
  //
  SEGGER_SYSVIEW_OnTaskStartExec((unsigned)pTask);
  //
  // Else no task activated, go into idle state
  //
  SEGGER_SYSVIEW_OnIdle()
}
```

10.1.1.4 Task Stop Ready

A task is blocked or suspended.

Task Stop Ready events happen when a task is suspended or blocked, for example because it delays for a specific time, or when it tries to claim a resource which is in use by another task, or when it waits for an event to happen. When a task is suspended or blocked the Scheduler will activate another task or go into idle state.

On Task Stop Ready events call `SEGGER_SYSVIEW_OnTaskStopReady()` with the Id of the task which is blocked and a ReasonId which can indicate why the task is blocked.

Example

```
void OS_Delay(unsigned NumTicks) {

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskStopReady(OS_Global.pCurrentTask, OS_CAUSE_WAITING);
}
```

10.1.1.5 Task Stop Exec

A task terminates.

Task Stop Exec events happen when a task finally stops execution, for example when it has done its job and terminates.

On Task Stop Ready events call `SEGGER_SYSVIEW_OnTaskStopExec()` to record the current task as stopped.

Example

```
void OS_TerminateTask(void) {

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskStopExec();
}
```

10.1.1.6 System Idle

No task is executing, the system goes into Idle state.

System Idle events happen, when a task is suspended or stopped and no other task is ready. The system can switch into an idle state to save power, wait for an interrupt or a task to become ready.

In some OSes Idle is handled by an additional task. In this case it is recommended to record System Idle events, when the Idle task is activated, too.

Time spent in Idle state is displayed as not CPU Load in SystemView.

On System Idle events call `SEGGER_SYSVIEW_OnIdle()`.

Example

```
void OS_Switch(void) {

  [OS specific code ...]

  //
  // If a task is activated
  //
  SEGGER_SYSVIEW_OnTaskStartExec((unsigned)pTask);
  //
  // Else no task activated, go into idle state
  //
  SEGGER_SYSVIEW_OnIdle()
}
```

10.1.2 Recording interrupts

SystemView can record entering and leaving interrupt service routines (ISRs). The SystemView API provides functions for these events which should be added to the OS when it provides functions to mark interrupt execution.

When the OS scheduler is controlled by interrupts, i.e. the SysTick interrupt, the exit interrupt event should distinguish between resuming normal execution or switching into the scheduler, and call the appropriate SystemView function.

10.1.2.1 Enter Interrupt

When the OS provides a function to inform the OS that interrupt code is executing, to be called at the start of an Interrupt Service Routine (ISR), the OS function should call `SEGGER_SYSVIEW_RecordEnterISR()` to record the Enter Interrupt event.

When the OS does not provide an enter interrupt function, or the ISR does not call it, it is the user's responsibility to call `SEGGER_SYSVIEW_RecordEnterISR()` to be able to record interrupt execution.

`SEGGER_SYSVIEW_RecordEnterISR()` automatically retrieves the interrupt ID via the `SEGGER_SYSVIEW_GET_INTERRUPT_ID()` function macro as defined in `SEGGER_SYSVIEW_Conf.h`.

Example

```
void OS_EnterInterrupt(void) {

  [OS specific code ...]

  SEGGER_SYSVIEW_RecordEnterISR();
}
```

10.1.2.2 Exit Interrupt

When the OS provides a function to inform the OS that interrupt code has executed, to be called at the and of an Interrupt Service Routine (ISR), the OS function should call:

- `SEGGER_SYSVIEW_RecordExitISR()` when the system will resume normal execution.
- `SEGGER_SYSVIEW_RecordExitISRToScheduler()` when the interrupt caused a context switch.

Example

```
void OS_ExitInterrupt(void) {

  [OS specific code ...]
  //
```



```
  // If the interrupt will switch to the Scheduler
  //
  SEGGER_SYSVIEW_RecordExitISRToScheduler();
  //
  // Otherwise
  //
  SEGGER_SYSVIEW_RecordExitISR();
}
```

10.1.2.3 Example ISRs

The following two examples show how to record interrupt execution with SystemView with OS interrupt handling and without.

Example with OS handling

```
void Timer_Handler(void) {
  //
  // Inform OS about start of interrupt execution
  // (records SystemView Enter Interrupt event).
  //
  OS_EnterInterrupt();
  //
  // Interrupt functionality could be here
  //
  APP_TimerCnt++;
  //
  // Inform OS about end of interrupt execution
  // (records SystemView Exit Interrupt event).
  //
  OS_ExitInterrupt();
}
```

Example without OS handling

```
void ADC_Handler(void) {
  //
  // Explicitly record SystemView Enter Interrupt event.
  // Should not be called in high-frequency interrupts.
  //
  SEGGER_SYSVIEW_RecordEnterISR();
  //
  // Interrupt functionality could be here
  //
  APP_ADCValue = ADC.Value;
  //
  // Explicitly record SystemView Exit Interrupt event.
  // Should not be called in high-frequency interrupts.
  //
  SEGGER_SYSVIEW_RecordExitISR();
}
```

10.1.3 Recording run-time information

SystemView can record more detailed run-time information like the system time and information about tasks. These information are recorded when the recording is started and periodically requested when SystemView is running.

To request the information a `SEGGER_SYSVIEW_OS_API` struct with the OS-specific functions as callbacks can be passed to SystemView upon initialization.

Setting the `SEGGER_SYSVIEW_OS_API` is optional, but is recommended to allow SystemView to display more detailed information.

SEGGER_SYSVIEW_OS_API

```
typedef struct {
  U64  (*pfGetTime)      (void);
  void (*pfSendTaskList) (void);
} SEGGER_SYSVIEW_OS_API;
```

Parameters

Parameter	Description
pfGetTime	Pointer to a function returning the system time.
pfSendTaskList	Pointer to a function recording the entire task list.

10.1.3.1 pfGetTime

Description

Get the system time, i.e. the time since starting the system in microseconds.

If pfGetTime is NULL SystemView can show timestamps relative to the start of recording only.

Prototype

```
U64 (*pfGetTime) (void);
```

10.1.3.2 pfSendTaskList

Description

Record the entire task list via `SEGGER_SYSVIEW_SendTaskInfo()`.

If pfSendTaskList is NULL SystemView might only get task information of tasks which are newly created while recording. pfSendTaskList is called periodically when SystemView is connected to keep track on the current task list.

Prototype

```
void (*pfSendTaskList) (void);
```

Example

```
void cbSendTaskList(void) {
  SEGGER_SYSVIEW_TASKINFO Info;
  OS_TASK*                pTask;

  OS_EnterRegion(); // Disable scheduling to make sure the task list does not change.
  for (pTask = OS_Global.pTask; pTask; pTask = pTask->pNext) {
    //
    // Fill all elements with 0 to allow extending the structure
    // in future version without breaking the code.
    //
    memset(&Info, 0, sizeof(Info));
    //
    // Fill elements with current task information
    //
    Info.TaskID     = (U32)pTask;
    Info.sName      = pTask->Name;
    Info.Prio       = pTask->Priority;
    Info.StackBase  = (U32)pTask->pStackBot;
    Info.StackSize  = pTask->StackSize;
    //
```

```
    // Record current task information
    //
    SEGGER_SYSVIEW_SendTaskInfo(&Info);
  }
  OS_LeaveRegion(); // Enable scheduling again.
}
```

10.1.4 Recording OS API calls

In addition to the OS core instrumentation, SystemView can record OS API calls which are done from the application. API functions can be instrumented like the OS core.

Recording API events with SystemView can be done with the ready-to-use `SEGGER_SYSVIEW_RecordXXX()` functions when passing simple parameters, or by using the appropriate `SEGGER_SYSVIEW_EncodeXXX()` functions to create a SystemView event and calling `SEGGER_SYSVIEW_SendPacket()` to record it.

Example

```
/*********************************************************************
*
*       OS_malloc()
*
*  Function description
*    API function to allocate memory on the heap.
*/
void OS_malloc(unsigned Size) {
  SEGGER_SYSVIEW_RecordU32(ID_OS_MALLOC,  // Id of OS_malloc (>= 32)
                           Size           // First parameter
                          );

  [OS specific code...]
}
```

To record how long the execution of an API function takes and to record its return value, the return of an API function can be instrumented, too by calling `SEGGER_SYSVIEW_RecordEndCall` to only record the return or `SEGGER_SYSVIEW_RecordEndCallReturnValue` to record the return and its return value.

10.1.5 OS description file

In order for SystemView to properly decode API calls it requires a description file to be present in the `/description/` directory of SystemView. The name of the file has to be `SYSVIEW_<OSName>.txt` where `<OSName` is the name as sent in the system description.

10.1.5.1 API Function description

A description file includes all API functions which can be recorded by the OS. Each line in the file is one function in the following format:

```
<EventID> <FunctionName> <ParameterDescription> | <ReturnValueDescription>
```

`<EventId>` is the Id which is recorded for the API function. It can be in the range of 32 to 511.

`<FunctionName>` is the name of the API function, displayed in the Event column of SystemView. It may not contain spaces.

`<ParameterDescription>` is the description string of the parameters which are recorded with the API function.

`<ReturnValueDescription>` is the description string of the return value which can be recorded with SystemView. The ReturnValueDescription is optional.

The parameter display can be configured by a set of modifiers:

- %b - Display parameter as binary.
- %B - Display parameter as hexadecimal string (e.g. 00 AA FF …).
- %d - Display parameter as signed decimal integer.
- %D - Display parameter as time value.
- %I - Display parameter as a resource name if the resource id is known to SystemView.
- %p - Display parameter as 4 byte hexadecimal integer (e.g. 0xAABBCCDD).
- %s - Display parameter as string.
- %t - Display parameter as a task name if the task id is known to SystemView.
- %u - Display parameter as unsigned decimal integer.
- %x - Display parameter as hexadecimal integer.

Example

The following example shows a part of SYSVIEW_embOS.txt

```
35        OS_CheckTimer             pGlobal=%p
42        OS_Delay                  Delay=%u
43        OS_DelayUntil             Time=%u
44        OS_setPriority            Task=%t Pri=%u
45        OS_WakeTask               Task=%t
46        OS_CreateTask             Task=%t Pri=%u Stack=%p Size=%u
```

In addition to the default modifiers the description file can define NamedTypes to map numerical values to strings, which can for example be useful to display the textual value of enums or error codes.

NamedTypes have following format:

```
NamedType <TypeName> <Key>=<Value> [<Key1>=<Value1> ...]
```

NamedTypes can be used in the ParameterDescription and the ReturnValueDescription.

Example

```
#
# Types for parameter formatters
#
NamedType OSErr 0=OS_ERR_NONE
NamedType OSErr 10000=OS_ERR_A 10001=OS_ERR_ACCEPT_ISR
NamedType OSErr 12000=OS_ERR_C 12001=OS_ERR_CREATE_ISR
NamedType OSErr 13000=OS_ERR_D 13001=OS_ERR_DEL_ISR

NamedType OSFlag 0=FLAG_NONE 1=FLAG_READ 2=FLAG_WRITE 3=FLAG_READ_WRITE
#
# API Functions
#
34     OSFunc Param=%OSFlag | Returns %OSErr
```

10.1.5.2 Task State description

When a task pauses execution its state is recorded in the SystemView event.

This task state can be converted to a textual representation in SystemView with the TaskState desctiption.

TaskState has following format:

```
TaskState <Mask> <Key>=<Value>, [<Key1>=<Value1>, ...]
```

Example

```
#
```



```
# Task States
#
TaskState 0xFF 0=Ready, 1=Delayed or Timeout, 2=Pending, 3=Pending with Timeout,
 4=Suspended, 5=Suspended with Timeout, 6=Suspended and Pending, 7=Suspended and
 Pending with Timeout, 255=Deleted
```

10.1.5.3 Option description

OS-Specific options can also be set in the description file to configure SystemView.

Currently available options to be inserted in the description files are:

`Option ReversePriority`: Higher task priority value equals lower task priority.

10.1.6 OS integration sample

The code below shows where to integrate SystemView in an OS based on pseudo-code snippets and can be used as reference.

```
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
*                       The Embedded Experts                         *
*                          www.segger.com                            *
**********************************************************************
-------------------------- END-OF-HEADER -----------------------------

Purpose : Pseudo-code OS with SEGGER SystemView integration.
*/

/*********************************************************************
*
*       OS_CreateTask()
*
*  Function description
*    Create a new task and add it to the system.
*/
void OS_CreateTask(TaskFunc* pF, unsigned Prio, const char* sName, void* pStack) {
  SEGGER_SYSVIEW_TASKINFO Info;
  OS_TASK*                pTask;  // Pseudo struct to be replaced

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskCreate((unsigned)pTask);
  memset(&Info, 0, sizeof(Info));
  //
  // Fill elements with current task information
  //
  Info.TaskID     = (U32)pTask;
  Info.sName      = pTask->Name;
  Info.Prio       = pTask->Priority;
  Info.StackBase  = (U32)pTask->pStack;
  Info.StackSize  = pTask->StackSize;
  SEGGER_SYSVIEW_SendTaskInfo(&Info);
}

/*********************************************************************
*
*       OS_TerminateTask()
*
*  Function description
*    Terminate a task and remove it from the system.
*/
void OS_TerminateTask(void) {

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskStopExec();
```



```
}

/*********************************************************************
*
*       OS_Delay()
*
*  Function description
*    Delay and suspend a task for the given time.
*/
void OS_Delay(unsigned NumTicks) {

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskStopReady(OS_Global.pCurrentTask, OS_CAUSE_WAITING);
}

/*********************************************************************
*
*       OS_HandleTick()
*
*  Function description
*    OS System Tick handler.
*/
int OS_HandleTick(void) {
  int TaskReady = 0;   // Pseudo variable indicating a task is ready

  [OS specific code ...]

  if (TaskReady) {
    SEGGER_SYSVIEW_OnTaskStartReady((unsigned)pTask);
  }
}

/*********************************************************************
*
*       OS_Switch()
*
*  Function description
*    Switch to the next ready task or go to idle.
*/
void OS_Switch(void) {

  [OS specific code ...]

  //
  // If a task is activated
  //
  SEGGER_SYSVIEW_OnTaskStartExec((unsigned)pTask);
  //
  // Else no task activated, go into idle state
  //
  SEGGER_SYSVIEW_OnIdle()
}

/*********************************************************************
*
*       OS_EnterInterrupt()
*
*  Function description
*    Inform the OS about start of interrupt execution.
*/
void OS_EnterInterrupt(void) {

  [OS specific code ...]

  SEGGER_SYSVIEW_RecordEnterISR();
}
```

```
/*********************************************************************
*
*       OS_ExitInterrupt()
*
*  Function description
*    Inform the OS about end of interrupt execution and switch to
*    Scheduler if necessary.
*/
void OS_ExitInterrupt(void) {

  [OS specific code ...]
  //
  // If the interrupt will switch to the Scheduler
  //
  SEGGER_SYSVIEW_RecordExitISRToScheduler();
  //
  // Otherwise
  //
  SEGGER_SYSVIEW_RecordExitISR();
}
```


10.2 Integrating SEGGER SystemView into a middleware module

SEGGER SystemView can also be integrated into middleware modules or even application modules to get information about execution of these modules, like API calls or interrupt-triggered events. This integration is for example used in SEGGER embOS/IP to monitor sending and receiving packets via IP and SEGGER emFile to record API calls.

For integration into other modules, contact your distributor or do the integration following the instructions in this section.

10.2.1 Registering the module

To be able to record middleware module events, the module has to register at SystemView via `SEGGER_SYSVIEW_RegisterModule()`.

The module passes a `SEGGER_SYSVIEW_MODULE` struct pointer, which contains information about the module and receives the event offset for the event Ids the module can generate.

`sDescription` and `NumEvents` have to be set in the `SEGGER_SYSVIEW_MODULE` struct when registering. Optionally `pfSendModuleDesc` can be set, too.

Upon return of `SEGGER_SYSVIEW_RegisterModule()`, `EventOffset` of the `SEGGER_SYSVIEW_MODULE` struct is set to the lowest event Id the module may generate, and `pNext` is set to point to the next registered module to create a linked list. Because of this, the `SEGGER_SYSVIEW_MODULE` struct has to be writeable and may not be allocated on the stack.

SEGGER_SYSVIEW_MODULE

```
struct SEGGER_SYSVIEW_MODULE {
  const char*                   sModule;
        U32                     NumEvents;
        U32                     EventOffset;
        void                    (*pfSendModuleDesc)(void);
        SEGGER_SYSVIEW_MODULE*  pNext;
};
```

Parameters

Parameter	Description
sModule	Pointer to a string containing the module name and optionally the module event description.
NumEvents	Number of events the module wants to register.
EventOffset	Offset to be added to the event Ids. Out parameter, set by this function. Do not modify after calling this function.
pfSendModuleDesc	Callback function pointer to send more detailed module description to SystemView.
pNext	Pointer to next registered module. Out parameter, set by this function. Do not modify after calling this function.

Example

```
SEGGER_SYSVIEW_MODULE IPModule = {
  "M=embOSIP, " \
  "0 SendPacket IFace=%u NumBytes=%u, " \
  "1 ReceivePacket Iface=%d NumBytes=%u", // sModule
  2,                                      // NumEvents
  0,
  // EventOffset, Set by SEGGER_SYSVIEW_RegisterModule()
  NULL,
  // pfSendModuleDesc, NULL: No additional module description
```



```
  NULL,
  // pNext, Set by SEGGER_SYSVIEW_RegisterModule()
};

static void _IPTraceConfig(void) {
  //
  // Register embOS/IP at SystemView.
  // SystemView has to be initialized before.
  //
  SEGGER_SYSVIEW_RegisterModule(&IPModule);
}
```

10.2.2 Recording module activity

In order to be able to record module activity, the module has to be instrumented to generate SystemView events in the appropriate functions.

Instrumenting a module can be done by integrating the SystemView functions directly, via configurable macro functions or with an API structure which can be filled and set by SystemView.

Recording events with SystemView can be done with the ready-to-use `SEGGER_SYSVIEW_RecordXXX()` functions when passing simple parameters, or by using the appropriate `SEGGER_SYSVIEW_EncodeXXX()` functions to create a SystemView event and calling `SEGGER_SYSVIEW_SendPacket()` to record it.

Example

```
int SendPacket(IP_PACKET *pPacket) {
  //
  // The IP stack sends a packet.
  // Record it according to the module description of SendPacket.
  //
  SEGGER_SYSVIEW_RecordU32x2(
  // Id of SendPacket (0) + Offset for the registered module
                             ID_SENDPACKET + IPModule.EventOffset,
  // First parameter  (displayed as event parameter IFace)
                             pPacket->Interface,
  // Second parameter (displayed as event parameter NumBytes)
                             pPacket->NumBytes
                            );

  [Module specific code...]
}
```

For more information refer to *Recording OS API calls* on page 112 and the *API reference* on page 121.

As with OSes, the middleware module description can be made available in a description file with the name of the module (Value of M=). Refer to *OS description file* on page 112.

10.2.3 Providing the module description

`SEGGER_SYSVIEW_MODULE.sModule` points to a string which contains the basic information of the registered module, which is a comma-separated list and can contain following items:

Item	Identifier	Example
Module name	M	"M=embOSIP"
Module token	T	"T=IP"
Description	S	"S='embOS/IP V12.09'"
Module event	<ID> <Event> <Parameter>	"0 SendPacket IFace=%u NumBytes=%u"

The string length may not exceed SEGGER_SYSVIEW_MAX_STRING_LEN which is 128 by default.

To send additional description strings and to send the name of resources which are used and recorded by the module, SEGGER_SYSVIEW_MODULE.pfSendModuleDesc can be set when registering the module.

SEGGER_SYSVIEW_MODULE.pfSendModuleDesc is called periodically when SystemView is connected. It can call SEGGER_SYSVIEW_RecordModuleDescription() and SEGGER_SYSVIEW_NameResource().

Example

```
static void _cbSendIPModuleDesc(void) {
  SEGGER_SYSVIEW_NameResource((U32)&(RxPacketFifo), "Rx FIFO");
  SEGGER_SYSVIEW_NameResource((U32)&(TxPacketFifo), "Tx FIFO");
  SEGGER_SYSVIEW_RecordModuleDescription(&IPModule, "T=IP, S='embOS/IP V12.09'");
}

SEGGER_SYSVIEW_MODULE IPModule = {
  "M=embOSIP, " \
  "0 SendPacket IFace=%u NumBytes=%u, " \
  "1 ReceivePacket Iface=%d NumBytes=%u", // sModule
  2,                                      // NumEvents
  0,                                      // EventOffset, Set by RegisterModule()
  _cbSendIPModuleDesc,                    // pfSendModuleDesc
  NULL,                                   // pNext, Set by RegisterModule()
};
```


Chapter 11

API reference

This section describes the public API of SEGGER SystemView.

11.1 SEGGER SystemView API functions

The following functions can be used to include SEGGER SystemView into an application and for integration of SEGGER SystemView into OSes and middleware modules.

Function	Description
Control and initialization functions	
SEGGER_SYSVIEW_Init()	Initializes the SYSVIEW module.
SEGGER_SYSVIEW_SetRAMBase()	Sets the RAM base address, which is subtracted from IDs in order to save bandwidth.
SEGGER_SYSVIEW_Start()	Start recording SystemView events.
SEGGER_SYSVIEW_Stop()	Stop recording SystemView events.
SEGGER_SYSVIEW_GetSysDesc()	Triggers a send of the system information and description.
SEGGER_SYSVIEW_SendTaskList()	Send all tasks descriptors to the host.
SEGGER_SYSVIEW_SendTaskInfo()	Send a Task Info Packet, containing TaskId for identification, task priority and task name.
SEGGER_SYSVIEW_SendSysDesc()	Send the system description string to the host.
Event recording functions	
SEGGER_SYSVIEW_OnIdle()	Record an Idle event.
SEGGER_SYSVIEW_OnTaskCreate()	Record a Task Create event.
SEGGER_SYSVIEW_OnTaskStartExec()	Record a Task Start Execution event.
SEGGER_SYSVIEW_OnTaskStartReady()	Record a Task Start Ready event.
SEGGER_SYSVIEW_OnTaskStopExec()	Record a Task Stop Execution event.
SEGGER_SYSVIEW_OnTaskStopReady()	Record a Task Stop Ready event.
SEGGER_SYSVIEW_OnTaskTerminate()	Record a Task termination event.
SEGGER_SYSVIEW_OnUserStart()	Send a user event start, such as start of a subroutine for profiling.
SEGGER_SYSVIEW_OnUserStop()	Send a user event stop, such as return of a subroutine for profiling.
SEGGER_SYSVIEW_RecordEndCallU32()	Format and send an End API Call event with return value.
SEGGER_SYSVIEW_RecordEndCall()	Format and send an End API Call event without return value.
SEGGER_SYSVIEW_RecordEnterISR()	Format and send an ISR entry event.
SEGGER_SYSVIEW_RecordEnterTimer()	Format and send a Timer entry event.
SEGGER_SYSVIEW_RecordExitISRToScheduler()	Format and send an ISR exit into scheduler event.
SEGGER_SYSVIEW_RecordExitISR()	Format and send an ISR exit event.
SEGGER_SYSVIEW_RecordExitTimer()	Format and send a Timer exit event.
SEGGER_SYSVIEW_RecordString()	Formats and sends a SystemView packet containing a string.
SEGGER_SYSVIEW_RecordSystime()	Formats and sends a SystemView Systime containing a single U64 or U32 parameter payload.
SEGGER_SYSVIEW_RecordVoid()	Formats and sends a SystemView packet with an empty payload.

Function	Description
SEGGER_SYSVIEW_RecordU32()	Formats and sends a SystemView packet containing a single U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x10()	Formats and sends a SystemView packet containing 10 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x2()	Formats and sends a SystemView packet containing 2 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x3()	Formats and sends a SystemView packet containing 3 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x4()	Formats and sends a SystemView packet containing 4 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x5()	Formats and sends a SystemView packet containing 5 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x6()	Formats and sends a SystemView packet containing 6 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x7()	Formats and sends a SystemView packet containing 7 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x8()	Formats and sends a SystemView packet containing 8 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x9()	Formats and sends a SystemView packet containing 9 U32 parameter payload.
SEGGER_SYSVIEW_SendPacket()	Send an event packet.
SEGGER_SYSVIEW_NameResource()	Send the name of a resource to be displayed in SystemViewer.
Event parameter encoding functions	
SEGGER_SYSVIEW_EncodeU32()	Encode a U32 in variable-length format.
SEGGER_SYSVIEW_EncodeData()	Encode a byte buffer in variable-length format.
SEGGER_SYSVIEW_EncodeString()	Encode a string in variable-length format.
SEGGER_SYSVIEW_EncodeId()	Encode a 32-bit Id in shrunken variable-length format.
SEGGER_SYSVIEW_ShrinkId()	Get the shrunken value of an Id for further processing like in SEGGER_SYSVIEW_NameResource().
Middleware module registration	
SEGGER_SYSVIEW_RecordModuleDescription()	Sends detailed information of a registered module to the host.
SEGGER_SYSVIEW_RegisterModule()	Register a middleware module for recording its events.
SEGGER_SYSVIEW_SendModule()	Sends the information of a registered module to the host.
SEGGER_SYSVIEW_SendModuleDescription()	Triggers a send of the registered module descriptions.
SEGGER_SYSVIEW_SendNumModules()	Send the number of registered modules to the host.
printf-Style functions	
SEGGER_SYSVIEW_PrintfHostEx()	Print a string which is formatted on the host by SystemViewer with Additional information.

Function	Description
SEGGER_SYSVIEW_PrintfTargetEx()	Print a string which is formatted on the target before sent to the host with Additional information.
SEGGER_SYSVIEW_PrintfHost()	Print a string which is formatted on the host by SystemViewer.
SEGGER_SYSVIEW_PrintfTarget()	Print a string which is formatted on the target before sent to the host.
SEGGER_SYSVIEW_Print()	Print a string to the host.
SEGGER_SYSVIEW_WarnfHost()	Print a warnin string which is formatted on the host by SystemViewer.
SEGGER_SYSVIEW_WarnfTarget()	Print a warning string which is formatted on the target before sent to the host.
SEGGER_SYSVIEW_Warn()	Print a warning string to the host.
SEGGER_SYSVIEW_ErrorfHost()	Print an error string which is formatted on the host by SystemViewer.
SEGGER_SYSVIEW_ErrorfTarget()	Print an error string which is formatted on the target before sent to the host.
SEGGER_SYSVIEW_Error()	Print an error string to the host.
Run-time configuration functions	
SEGGER_SYSVIEW_EnableEvents()	Enable standard SystemView events to be generated.
SEGGER_SYSVIEW_DisableEvents()	Disable standard SystemView events to not be generated.
Application-provided functions	
SEGGER_SYSVIEW_Conf()	Initializes and configures SystemView for the specific OS
SEGGER_SYSVIEW_X_GetTimestamp()	Callback called by SystemView to get the timestamp in cycles.

11.1.1 SEGGER_SYSVIEW_Conf()

Description

Can be used with OS integration to allow easier initialization of SystemView and the OS SystemView interface.

This function is usually provided in the `SEGGER_SYSVIEW_Config_<OS>.c` configuration file of the used OS.

Prototype

```
void SEGGER_SYSVIEW_Conf(void);
```

Example implementation

```
void SEGGER_SYSVIEW_Conf(void) {
  //
  // Initialize SystemView
  //
  SEGGER_SYSVIEW_Init(SYSVIEW_TIMESTAMP_FREQ,   // Frequency of the timestamp.
                      SYSVIEW_CPU_FREQ,         // Frequency of the system.
                      &SYSVIEW_X_OS_TraceAPI,
  // OS-specific SEGGER_SYSVIEW_OS_API
                      _cbSendSystemDesc
  // Callback for application-specific description
                     );
  SEGGER_SYSVIEW_SetRAMBase(SYSVIEW_RAM_BASE);
  // Explicitly set the RAM base address.
  OS_SetTraceAPI(&embOS_TraceAPI_SYSVIEW);
  // Configure embOS to use SystemView via the Trace-API.
}
```

11.1.2 SEGGER_SYSVIEW_DisableEvents()

Description

Disable standard SystemView events to not be generated.

Prototype

```
void SEGGER_SYSVIEW_DisableEvents(U32 DisableMask);
```

Parameters

Parameter	Description
DisableMask	Events to be disabled.

11.1.3 SEGGER_SYSVIEW_EnableEvents()

Description

Enable standard SystemView events to be generated.

Prototype

```
void SEGGER_SYSVIEW_EnableEvents(U32 EnableMask);
```

Parameters

Parameter	Description
EnableMask	Events to be enabled.

11.1.4 SEGGER_SYSVIEW_EncodeData()

Description

Encode a byte buffer in variable-length format.

Prototype

```
U8 *SEGGER_SYSVIEW_EncodeData(      U8           * pPayload,
                              const char         * pSrc,
                                    unsigned int   NumBytes);
```

Parameters

Parameter	Description
pPayload	Pointer to where string will be encoded.
pSrc	Pointer to data buffer to be encoded.
NumBytes	Number of bytes in the buffer to be encoded.

Return value

Pointer to the byte following the value, i.e. the first free byte in the payload and the next position to store payload content.

Additional information

The data is encoded as a count byte followed by the contents of the data buffer. Make sure NumBytes + 1 bytes are free for the payload.

11.1.5 SEGGER_SYSVIEW_EncodeId()

Description

Encode a 32-bit `Id` in shrunken variable-length format.

Prototype

```
U8 *SEGGER_SYSVIEW_EncodeId(U8  * pPayload,
                            U32   Id);
```

Parameters

Parameter	Description
`pPayload`	Pointer to where the `Id` will be encoded.
`Id`	The 32-bit value to be encoded.

Return value

Pointer to the byte following the value, i.e. the first free byte in the payload and the next position to store payload content.

Additional information

The parameters to shrink an `Id` can be configured in `SEGGER_SYSVIEW_Conf.h` and via `SEGGER_SYSVIEW_SetRAMBase()`. `SEGGER_SYSVIEW_ID_BASE`: Lowest `Id` reported by the application. (i.e. `0x20000000` when all Ids are an address in this RAM) `SEGGER_SYSVIEW_ID_SHIFT`: Number of bits to shift the `Id` to save bandwidth. (i.e. 2 when Ids are 4 byte aligned)

11.1.6 SEGGER_SYSVIEW_EncodeString()

Description

Encode a string in variable-length format.

Prototype

```
U8 *SEGGER_SYSVIEW_EncodeString(      U8           * pPayload,
                                 const char         * s,
                                       unsigned int   MaxLen);
```

Parameters

Parameter	Description
pPayload	Pointer to where string will be encoded.
s	String to encode.
MaxLen	Maximum number of characters to encode from string.

Return value

Pointer to the byte following the value, i.e. the first free byte in the payload and the next position to store payload content.

Additional information

The string is encoded as a count byte followed by the contents of the string. No more than 1 + MaxLen bytes will be encoded to the payload.

11.1.7 SEGGER_SYSVIEW_EncodeU32()

Description

Encode a U32 in variable-length format.

Prototype

```
U8 *SEGGER_SYSVIEW_EncodeU32(U8  * pPayload,
                             U32   Value);
```

Parameters

Parameter	Description
pPayload	Pointer to where U32 will be encoded.
Value	The 32-bit value to be encoded.

Return value

Pointer to the byte following the value, i.e. the first free byte in the payload and the next position to store payload content.

11.1.8 SEGGER_SYSVIEW_Error()

Description

Print an error string to the host.

Prototype

```
void SEGGER_SYSVIEW_Error(const char * s);
```

Parameters

Parameter	Description
s	String to sent.

11.1.9 SEGGER_SYSVIEW_ErrorfHost()

Description

Print an error string which is formatted on the host by SystemViewer.

Prototype

```
void SEGGER_SYSVIEW_ErrorfHost(const char * s,
                                           ...);
```

Parameters

Parameter	Description
s	String to be formatted.

Additional information

All format arguments are treated as 32-bit scalar values.

11.1.10 SEGGER_SYSVIEW_ErrorfTarget()

Description

Print an error string which is formatted on the target before sent to the host.

Prototype

```
void SEGGER_SYSVIEW_ErrorfTarget(const char * s,
                                 ...);
```

Parameters

Parameter	Description
s	String to be formatted.

11.1.11 SEGGER_SYSVIEW_GetSysDesc()

Description

Triggers a send of the system information and description.

Prototype

```
void SEGGER_SYSVIEW_GetSysDesc(void);
```

11.1.12 SEGGER_SYSVIEW_Init()

Description

Initializes the SYSVIEW module. Must be called before SystemViewer attaches to the system.

Prototype

```
void SEGGER_SYSVIEW_Init(      U32                                SysFreq,
                               U32                                CPUFreq,
                         const SEGGER_SYSVIEW_OS_API            * pOSAPI,
                               SEGGER_SYSVIEW_SEND_SYS_DESC_FUNC  pfSendSysDesc);
```

Parameters

Parameter	Description
SysFreq	Frequency of timestamp, i.e. CPU core clock frequency.
CPUFreq	CPU core clock frequency.
pOSAPI	Pointer to the API structure for OS-specific functions.
pfSendSysDesc	Pointer to SendSysDesc callback function.

Additional information

This function initializes the RTT channel used to transport SEGGER SystemView packets. The channel is assigned the label “SysView” for client software to identify the SystemView channel.

11.1.13 SEGGER_SYSVIEW_NameResource()

Description

Send the name of a resource to be displayed in SystemViewer.

Prototype

```
void SEGGER_SYSVIEW_NameResource(      U32    ResourceId,
                                 const char * sName);
```

Parameters

Parameter	Description
ResourceId	Id of the resource to be named. i.e. its address.
sName	Pointer to the resource name. (Max. SEGGER_SYSVIEW_MAX_STRING_LEN Bytes)

11.1.14 SEGGER_SYSVIEW_OnIdle()

Description

Record an Idle event.

Prototype

```
void SEGGER_SYSVIEW_OnIdle(void);
```


11.1.15 SEGGER_SYSVIEW_OnTaskCreate()

Description

Record a Task Create event. The Task Create event corresponds to creating a task in the OS.

Prototype

```
void SEGGER_SYSVIEW_OnTaskCreate(U32 TaskId);
```

Parameters

Parameter	Description
TaskId	Task ID of created task.

11.1.16 SEGGER_SYSVIEW_OnTaskStartExec()

Description

Record a Task Start Execution event. The Task Start event corresponds to when a task has started to execute rather than when it is ready to execute.

Prototype

```
void SEGGER_SYSVIEW_OnTaskStartExec(U32 TaskId);
```

Parameters

Parameter	Description
TaskId	Task ID of task that started to execute.

11.1.17 SEGGER_SYSVIEW_OnTaskStartReady()

Description

Record a Task Start Ready event.

Prototype

```
void SEGGER_SYSVIEW_OnTaskStartReady(U32 TaskId);
```

Parameters

Parameter	Description
TaskId	Task ID of task that started to execute.

11.1.18 SEGGER_SYSVIEW_OnTaskStopExec()

Description

Record a Task Stop Execution event. The Task Stop event corresponds to when a task stops executing and terminates.

Prototype

```
void SEGGER_SYSVIEW_OnTaskStopExec(void);
```


11.1.19 SEGGER_SYSVIEW_OnTaskStopReady()

Description

Record a Task Stop Ready event.

Prototype

```
void SEGGER_SYSVIEW_OnTaskStopReady(U32          TaskId,
                                    unsigned int Cause);
```

Parameters

Parameter	Description
TaskId	Task ID of task that completed execution.
Cause	Reason for task to stop (i.e. Idle/Sleep)

11.1.20 SEGGER_SYSVIEW_OnTaskTerminate()

Description

Record a Task termination event. The Task termination event corresponds to terminating a task in the OS. If the `TaskId` is the currently active task, `SEGGER_SYSVIEW_OnTaskStopExec` may be used, either.

Prototype

```
void SEGGER_SYSVIEW_OnTaskTerminate(U32 TaskId);
```

Parameters

Parameter	Description
`TaskId`	Task ID of terminated task.

11.1.21 SEGGER_SYSVIEW_OnUserStart()

Description

Send a user event start, such as start of a subroutine for profiling.

Prototype

```
void SEGGER_SYSVIEW_OnUserStart(unsigned UserId);
```

Parameters

Parameter	Description
UserId	User defined ID for the event.

11.1.22 SEGGER_SYSVIEW_OnUserStop()

Description

Send a user event stop, such as return of a subroutine for profiling.

Prototype

```
void SEGGER_SYSVIEW_OnUserStop(unsigned UserId);
```

Parameters

Parameter	Description
UserId	User defined ID for the event.

11.1.23 SEGGER_SYSVIEW_Print()

Description

Print a string to the host.

Prototype

```
void SEGGER_SYSVIEW_Print(const char * s);
```

Parameters

Parameter	Description
s	String to sent.

11.1.24 SEGGER_SYSVIEW_PrintfHost()

Description

Print a string which is formatted on the host by SystemViewer.

Prototype

```
void SEGGER_SYSVIEW_PrintfHost(const char * s,
                                           ...);
```

Parameters

Parameter	Description
s	String to be formatted.

Additional information

All format arguments are treated as 32-bit scalar values.

11.1.25 SEGGER_SYSVIEW_PrintfHostEx()

Description

Print a string which is formatted on the host by SystemViewer with Additional information.

Prototype

```
void SEGGER_SYSVIEW_PrintfHostEx(const char * s,
                                 U32    Options,
                                        ...);
```

Parameters

Parameter	Description
s	String to be formatted.
Options	Options for the string. i.e. Log level.

Additional information

All format arguments are treated as 32-bit scalar values.

11.1.26 SEGGER_SYSVIEW_PrintfTarget()

Description

Print a string which is formatted on the target before sent to the host.

Prototype

```
void SEGGER_SYSVIEW_PrintfTarget(const char * s,
                                 ...);
```

Parameters

Parameter	Description
s	String to be formatted.

11.1.27 SEGGER_SYSVIEW_PrintfTargetEx()

Description

Print a string which is formatted on the target before sent to the host with Additional information.

Prototype

```
void SEGGER_SYSVIEW_PrintfTargetEx(const char * s,
                                   U32    Options,
                                   ...);
```

Parameters

Parameter	Description
s	String to be formatted.
Options	Options for the string. i.e. Log level.

11.1.28 SEGGER_SYSVIEW_RecordEndCall()

Description

Format and send an End API Call event without return value.

Prototype

```
void SEGGER_SYSVIEW_RecordEndCall(unsigned int EventID);
```

Parameters

Parameter	Description
EventID	Id of API function which ends.

11.1.29 SEGGER_SYSVIEW_RecordEndCallU32()

Description

Format and send an End API Call event with return value.

Prototype

```
void SEGGER_SYSVIEW_RecordEndCallU32(unsigned int EventID,
                                     U32          Para0);
```

Parameters

Parameter	Description
EventID	Id of API function which ends.
Para0	Return value which will be returned by the API function.

11.1.30 SEGGER_SYSVIEW_RecordEnterISR()

Description

Format and send an ISR entry event.

Prototype

```
void SEGGER_SYSVIEW_RecordEnterISR(void);
```

Additional information

Example packets sent 02 0F 50 // ISR(15) Enter. Timestamp is 80 (`0x50`)

11.1.31 SEGGER_SYSVIEW_RecordEnterTimer()

Description

Format and send a Timer entry event.

Prototype

```
void SEGGER_SYSVIEW_RecordEnterTimer(U32 TimerId);
```

Parameters

Parameter	Description
TimerId	Id of the timer which starts.

11.1.32 SEGGER_SYSVIEW_RecordExitISR()

Description

Format and send an ISR exit event.

Prototype

```
void SEGGER_SYSVIEW_RecordExitISR(void);
```

Additional information

Format as follows: 03 <TimeStamp> // Max. packet len is 6

Example packets sent 03 20 // ISR Exit. Timestamp is 32 (`0x20`)

11.1.33 SEGGER_SYSVIEW_RecordExitISRToScheduler()

Description

Format and send an ISR exit into scheduler event.

Prototype

```
void SEGGER_SYSVIEW_RecordExitISRToScheduler(void);
```

Additional information

Format as follows: 18 <TimeStamp> // Max. packet len is 6

Example packets sent 18 20 // ISR Exit to Scheduler. Timestamp is 32 (`0x20`)

11.1.34 SEGGER_SYSVIEW_RecordExitTimer()

Description

Format and send a Timer exit event.

Prototype

```
void SEGGER_SYSVIEW_RecordExitTimer(void);
```


11.1.35 SEGGER_SYSVIEW_RecordModuleDescription()

Description

Sends detailed information of a registered module to the host.

Prototype

```
void SEGGER_SYSVIEW_RecordModuleDescription
                                      (const SEGGER_SYSVIEW_MODULE * pModule,
                                       const char                  * sDescription);
```

Parameters

Parameter	Description
pModule	Pointer to the described module.
sDescription	Pointer to a description string.

11.1.36 SEGGER_SYSVIEW_RecordString()

Description

Formats and sends a SystemView packet containing a string.

Prototype

```
void SEGGER_SYSVIEW_RecordString(      unsigned int   EventID,
                                 const char         * pString);
```

Parameters

Parameter	Description
EventID	SystemView event ID.
pString	The string to be sent in the SystemView packet payload.

Additional information

The string is encoded as a count byte followed by the contents of the string. No more than SEGGER_SYSVIEW_MAX_STRING_LEN bytes will be encoded to the payload.

11.1.37 SEGGER_SYSVIEW_RecordSystime()

Description

Formats and sends a SystemView Systime containing a single U64 or U32 parameter payload.

Prototype

```
void SEGGER_SYSVIEW_RecordSystime(void);
```

11.1.38 SEGGER_SYSVIEW_RecordU32()

Description

Formats and sends a SystemView packet containing a single U32 parameter payload.

Prototype

```
void SEGGER_SYSVIEW_RecordU32(unsigned int EventID,
                              U32          Value);
```

Parameters

Parameter	Description
EventID	SystemView event ID.
Value	The 32-bit parameter encoded to SystemView packet payload.

11.1.39 SEGGER_SYSVIEW_RecordU32x10()

Description

Formats and sends a SystemView packet containing 10 U32 parameter payload.

Prototype

```
void SEGGER_SYSVIEW_RecordU32x10(unsigned int EventID,
                                 U32          Para0,
                                 U32          Para1,
                                 U32          Para2,
                                 U32          Para3,
                                 U32          Para4,
                                 U32          Para5,
                                 U32          Para6,
                                 U32          Para7,
                                 U32          Para8,
                                 U32          Para9);
```

Parameters

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.
Para4	The 32-bit parameter encoded to SystemView packet payload.
Para5	The 32-bit parameter encoded to SystemView packet payload.
Para6	The 32-bit parameter encoded to SystemView packet payload.
Para7	The 32-bit parameter encoded to SystemView packet payload.
Para8	The 32-bit parameter encoded to SystemView packet payload.
Para9	The 32-bit parameter encoded to SystemView packet payload.

11.1.40 SEGGER_SYSVIEW_RecordU32x2()

Description

Formats and sends a SystemView packet containing 2 U32 parameter payload.

Prototype

```
void SEGGER_SYSVIEW_RecordU32x2(unsigned int EventID,
                                U32          Para0,
                                U32          Para1);
```

Parameters

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.

11.1.41 SEGGER_SYSVIEW_RecordU32x3()

Description

Formats and sends a SystemView packet containing 3 U32 parameter payload.

Prototype

```
void SEGGER_SYSVIEW_RecordU32x3(unsigned int EventID,
                                U32          Para0,
                                U32          Para1,
                                U32          Para2);
```

Parameters

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.

11.1.42 SEGGER_SYSVIEW_RecordU32x4()

Description

Formats and sends a SystemView packet containing 4 U32 parameter payload.

Prototype

```
void SEGGER_SYSVIEW_RecordU32x4(unsigned int EventID,
                                U32          Para0,
                                U32          Para1,
                                U32          Para2,
                                U32          Para3);
```

Parameters

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.

11.1.43 SEGGER_SYSVIEW_RecordU32x5()

Description

Formats and sends a SystemView packet containing 5 U32 parameter payload.

Prototype

```
void SEGGER_SYSVIEW_RecordU32x5(unsigned int EventID,
                                U32          Para0,
                                U32          Para1,
                                U32          Para2,
                                U32          Para3,
                                U32          Para4);
```

Parameters

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.
Para4	The 32-bit parameter encoded to SystemView packet payload.

11.1.44 SEGGER_SYSVIEW_RecordU32x6()

Description

Formats and sends a SystemView packet containing 6 U32 parameter payload.

Prototype

```
void SEGGER_SYSVIEW_RecordU32x6(unsigned int EventID,
                                U32          Para0,
                                U32          Para1,
                                U32          Para2,
                                U32          Para3,
                                U32          Para4,
                                U32          Para5);
```

Parameters

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.
Para4	The 32-bit parameter encoded to SystemView packet payload.
Para5	The 32-bit parameter encoded to SystemView packet payload.

11.1.45 SEGGER_SYSVIEW_RecordU32x7()

Description

Formats and sends a SystemView packet containing 7 U32 parameter payload.

Prototype

```
void SEGGER_SYSVIEW_RecordU32x7(unsigned int EventID,
                                U32          Para0,
                                U32          Para1,
                                U32          Para2,
                                U32          Para3,
                                U32          Para4,
                                U32          Para5,
                                U32          Para6);
```

Parameters

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.
Para4	The 32-bit parameter encoded to SystemView packet payload.
Para5	The 32-bit parameter encoded to SystemView packet payload.
Para6	The 32-bit parameter encoded to SystemView packet payload.

11.1.46 SEGGER_SYSVIEW_RecordU32x8()

Description

Formats and sends a SystemView packet containing 8 U32 parameter payload.

Prototype

```
void SEGGER_SYSVIEW_RecordU32x8(unsigned int EventID,
                                U32          Para0,
                                U32          Para1,
                                U32          Para2,
                                U32          Para3,
                                U32          Para4,
                                U32          Para5,
                                U32          Para6,
                                U32          Para7);
```

Parameters

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.
Para4	The 32-bit parameter encoded to SystemView packet payload.
Para5	The 32-bit parameter encoded to SystemView packet payload.
Para6	The 32-bit parameter encoded to SystemView packet payload.
Para7	The 32-bit parameter encoded to SystemView packet payload.

11.1.47 SEGGER_SYSVIEW_RecordU32x9()

Description

Formats and sends a SystemView packet containing 9 U32 parameter payload.

Prototype

```
void SEGGER_SYSVIEW_RecordU32x9(unsigned int EventID,
                                U32          Para0,
                                U32          Para1,
                                U32          Para2,
                                U32          Para3,
                                U32          Para4,
                                U32          Para5,
                                U32          Para6,
                                U32          Para7,
                                U32          Para8);
```

Parameters

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.
Para4	The 32-bit parameter encoded to SystemView packet payload.
Para5	The 32-bit parameter encoded to SystemView packet payload.
Para6	The 32-bit parameter encoded to SystemView packet payload.
Para7	The 32-bit parameter encoded to SystemView packet payload.
Para8	The 32-bit parameter encoded to SystemView packet payload.

11.1.48 SEGGER_SYSVIEW_RecordVoid()

Description

Formats and sends a SystemView packet with an empty payload.

Prototype

```
void SEGGER_SYSVIEW_RecordVoid(unsigned int EventID);
```

Parameters

Parameter	Description
EventID	SystemView event ID.

11.1.49 SEGGER_SYSVIEW_RegisterModule()

Description

Register a middleware module for recording its events.

Prototype

```
void SEGGER_SYSVIEW_RegisterModule(SEGGER_SYSVIEW_MODULE * pModule);
```

Parameters

Parameter	Description
pModule	The middleware module information.

Additional information

SEGGER_SYSVIEW_MODULE elements: sDescription - Pointer to a string containing the module name and optionally the module event description. NumEvents - Number of events the module wants to register. EventOffset - Offset to be added to the event Ids. Out parameter, set by this function. Do not modify after calling this function. pfSendModuleDesc - Callback function pointer to send more detailed module description to SystemViewer. pNext - Pointer to next registered module. Out parameter, set by this function. Do not modify after calling this function.

11.1.50 SEGGER_SYSVIEW_SendModule()

Description

Sends the information of a registered module to the host.

Prototype

```
void SEGGER_SYSVIEW_SendModule(U8 ModuleId);
```

Parameters

Parameter	Description
ModuleId	Id of the requested module.

11.1.51 SEGGER_SYSVIEW_SendModuleDescription()

Description

Triggers a send of the registered module descriptions.

Prototype

```
void SEGGER_SYSVIEW_SendModuleDescription(void);
```

11.1.52 SEGGER_SYSVIEW_SendNumModules()

Description

Send the number of registered modules to the host.

Prototype

```
void SEGGER_SYSVIEW_SendNumModules(void);
```


11.1.53 SEGGER_SYSVIEW_SendPacket()

Description

Send an event packet.

Prototype

```
int SEGGER_SYSVIEW_SendPacket(U8           * pPacket,
                              U8           * pPayloadEnd,
                              unsigned int   EventId);
```

Parameters

Parameter	Description
pPacket	Pointer to the start of the packet.
pPayloadEnd	Pointer to the end of the payload. Make sure there are at least 5 bytes free after the payload.
EventId	Id of the event packet.

Return value

≠ 0 Success, Message sent.
= 0 Buffer full, Message *NOT* sent.

11.1.54 SEGGER_SYSVIEW_SendSysDesc()

Description

Send the system description string to the host. The system description is used by SystemViewer to identify the current application and handle events accordingly.

Prototype

```
void SEGGER_SYSVIEW_SendSysDesc(const char * sSysDesc);
```

Parameters

Parameter	Description
sSysDesc	Pointer to the 0-terminated system description string.

Additional information

One system description string may not exceed SEGGER_SYSVIEW_MAX_STRING_LEN characters.

The Following items can be described in a system description string. Each item is identified by its identifier, followed by '=' and the value. Items are separated by ','.

Item	Identifier	Example
Application name	N	"N=Test Application"
Operating system	O	"O=embOS"
Additional module	M	"M=embOS/IP"
Target device	D	"D=MK66FN2M0xxx18"
Target core	C	"C=Cortex-M4"
Interrupt	I#<InterruptID>	"I#15=SysTick"

Example strings

- N=Test Application,O=embOS,D=MK66FN2M0xxx18
- I#15=SysTick,I#99=ETH_Tx,I#100=ETH_Rx

11.1.55 SEGGER_SYSVIEW_SendTaskInfo()

Description

Send a Task Info Packet, containing TaskId for identification, task priority and task name.

Prototype

```
void SEGGER_SYSVIEW_SendTaskInfo(const SEGGER_SYSVIEW_TASKINFO * pInfo);
```

Parameters

Parameter	Description
pInfo	Pointer to task information to send.

11.1.56 SEGGER_SYSVIEW_SendTaskList()

Description

Send all tasks descriptors to the host.

Prototype

```
void SEGGER_SYSVIEW_SendTaskList(void);
```


11.1.57 SEGGER_SYSVIEW_SetRAMBase()

Description

Sets the RAM base address, which is subtracted from IDs in order to save bandwidth.

Prototype

```
void SEGGER_SYSVIEW_SetRAMBase(U32 RAMBaseAddress);
```

Parameters

Parameter	Description
RAMBaseAddress	Lowest RAM Address. (i.e. 0x20000000 on most Cortex-M)

11.1.58 SEGGER_SYSVIEW_ShrinkId()

Description

Get the shrunken value of an `Id` for further processing like in `SEGGER_SYSVIEW_NameResource()`.

Prototype

```
U32 SEGGER_SYSVIEW_ShrinkId(U32 Id);
```

Parameters

Parameter	Description
`Id`	The 32-bit value to be shrunken.

Return value

Shrunken `Id`.

Additional information

The parameters to shrink an `Id` can be configured in `SEGGER_SYSVIEW_Conf.h` and via `SEGGER_SYSVIEW_SetRAMBase()`. `SEGGER_SYSVIEW_ID_BASE`: Lowest `Id` reported by the application. (i.e. `0x20000000` when all Ids are an address in this RAM) `SEGGER_SYSVIEW_ID_SHIFT`: Number of bits to shift the `Id` to save bandwidth. (i.e. 2 when Ids are 4 byte aligned)

11.1.59 SEGGER_SYSVIEW_Start()

Description

Start recording SystemView events. This function is triggered by the host application.

Prototype

```
void SEGGER_SYSVIEW_Start(void);
```

Additional information

This function enables transmission of SystemView packets recorded by subsequent trace calls and records a SystemView Start event.

As part of start, a SystemView Init packet is sent, containing the system frequency. The list of current tasks, the current system time and the system description string is sent, too.

11.1.60 SEGGER_SYSVIEW_Stop()

Description

Stop recording SystemView events.

Prototype

```
void SEGGER_SYSVIEW_Stop(void);
```

Additional information

This function disables transmission of SystemView packets recorded by subsequent trace calls. If transmission is enabled when this function is called, a single SystemView Stop event is recorded to the trace, send, and then trace transmission is halted.

11.1.61 SEGGER_SYSVIEW_Warn()

Description

Print a warning string to the host.

Prototype

```
void SEGGER_SYSVIEW_Warn(const char * s);
```

Parameters

Parameter	Description
s	String to sent.

11.1.62 SEGGER_SYSVIEW_WarnfHost()

Description

Print a warnin string which is formatted on the host by SystemViewer.

Prototype

```
void SEGGER_SYSVIEW_WarnfHost(const char * s,
                                          ...);
```

Parameters

Parameter	Description
s	String to be formatted.

Additional information

All format arguments are treated as 32-bit scalar values.

11.1.63 SEGGER_SYSVIEW_WarnfTarget()

Description

Print a warning string which is formatted on the target before sent to the host.

Prototype

```
void SEGGER_SYSVIEW_WarnfTarget(const char * s,
                                ...);
```

Parameters

Parameter	Description
s	String to be formatted.

11.1.64 SEGGER_SYSVIEW_X_GetTimestamp()

Description

This function needs to be implemented when SEGGER_SYSVIEW_GET_TIMESTAMP() is configured to call it. By default this is done on all non-Cortex-M3/4 targets.

Prototype

```
U32 SEGGER_SYSVIEW_X_GetTimestamp(void);
```

Return value

Returns the current system timestamp in timestamp cycles. On Cortex-M3 and Cortex-M4 this is the cycle counter.

Example implementation

```
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
  U32 TickCount;
  U32 Cycles;
  U32 CyclesPerTick;
  //
  // Get the cycles of the current system tick.
  // SysTick is down-counting, subtract the current value from the number of cycles per
  //
  CyclesPerTick = SYST_RVR + 1;
  Cycles = (CyclesPerTick - SYST_CVR);
  //
  // Get the system tick count.
  //
  TickCount = SEGGER_SYSVIEW_TickCnt; // SEGGER_SYSVIEW_TickCnt is incremented by the sy
  //
  // If a SysTick interrupt is pending increment the TickCount
  //
  if ((SCB_ICSR & SCB_ICSR_PENDSTSET_MASK) != 0) {
    TickCount++;
  }
  Cycles += TickCount * CyclesPerTick;

  return Cycles;
}
```


Chapter 12

Frequently asked questions

Q: *Can I use the SystemView App while I am debugging my application?*

A: Yes. SystemView can run in parallel to a debugger and do continuous recording. To make sure data can be read fast enough, configure the debugger connection to a high interface speed (≥ 4 MHz).

Q: *Can I do continuous recording without a J-Link?*

A: No. Continuous recording requires the J-Link Real Time Transfer (RTT) technology to automatically read the data from the target. Single-shot and post-mortem recording can be done with any debug probe.

Q: *Can I do continuous recording on Cortex-A, Cortex-R or ARM7, ARM9?*

A: No. RTT requires memory access on the target while the target is running. If you have one of these devices, only one-time recording can be done.

Q: *I get overflow events when continuously recording. How can I prevent this?*

A: Overflow events occur when the SystemVIew RTT buffer is full. This can happen for following reasons:

- J-Link is kept busy by a debugger and cannot read the data fast enough.
- The target interface speed is too low to read the data fast enough.
- The application generates too many events to fit into the buffer.

To prevent this:

- Minimize the interactions of the debugger with J-Link while the target is running. (i.e. disable live watches)
- Select a higher interface speed in all instances connected to J-Link. (i.e. The debugger and SystemView)
- Choose a larger buffer for SystemView. (1 - 4 kByte)
- Run SystemView stand-alone without a debugger.

Q: *SystemView cannot find the RTT Control Block, how can I configure it?*

A: Auto-detection of the RTT Control Block can only be done in a known RAM address range after it is initialized. Make sure the application startup has ran when starting to record. If the RTT Control Block is outside the known range for the selected device, either select ’Address’ and enter the exact address of the RTT Control Block or select ’Address Range’ and enter an address range in which the RTT Control Block will be.

Q: *Do I have to select a Target Device to start recording?*

A: Yes. J-Link needs to now which target device is connected. The drop-down lists the most recently used devices. To select another device simply enter its name. A list of supported devices can be found here.

Q: *My question is not listed above. Where can I get more information?*

A: For more information and help please ask your question in the SEGGER forum {https://forum.segger.com}

}

LifeElectronics
Живое партнерство

ООО «ЛайфЭлектроникс» **“LifeElectronics” LLC**

ИНН 7805602321 КПП 780501001 Р/С 40702810122510004610 ФАКБ "АБСОЛЮТ БАНК" (ЗАО) в г.Санкт-Петербурге К/С 30101810900000000703 БИК 044030703

Компания «Life Electronics» занимается поставками электронных компонентов импортного и отечественного производства от производителей и со складов крупных дистрибьюторов Европы, Америки и Азии.

С конца 2013 года компания активно расширяет линейку поставок компонентов по направлению коаксиальный кабель, кварцевые генераторы и конденсаторы (керамические, пленочные, электролитические), за счёт заключения дистрибьюторских договоров

Мы предлагаем:

- *Конкурентоспособные цены и скидки постоянным клиентам.*
- *Специальные условия для постоянных клиентов.*
- *Подбор аналогов.*
- *Поставку компонентов в любых объемах, удовлетворяющих вашим потребностям.*

- *Приемлемые сроки поставки, возможна ускоренная поставка.*
- *Доставку товара в любую точку России и стран СНГ.*
- *Комплексную поставку.*
- *Работу по проектам и поставку образцов.*
- *Формирование склада под заказчика.*

- *Сертификаты соответствия на поставляемую продукцию (по желанию клиента).*
- *Тестирование поставляемой продукции.*
- *Поставку компонентов, требующих военную и космическую приемку.*
- *Входной контроль качества.*
- *Наличие сертификата ISO.*

В составе нашей компании организован Конструкторский отдел, призванный помогать разработчикам, и инженерам.

Конструкторский отдел помогает осуществить:

- *Регистрацию проекта у производителя компонентов.*
- *Техническую поддержку проекта.*
- *Защиту от снятия компонента с производства.*
- *Оценку стоимости проекта по компонентам.*
- *Изготовление тестовой платы монтаж и пусконаладочные работы.*

Тел: +7 (812) 336 43 04 (многоканальный)
Email: org@lifeelectronics.ru

www.lifeelectronics.ru