

# Application Note

## Integrating SEGGER SystemView into an OS

Document: AN08003  
Software Version: 1.00  
Revision: 0  
Date: November 12, 2015



A product of SEGGER Microcontroller GmbH & Co. KG

[www.segger.com](http://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

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of SEGGER. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2015 SEGGER Microcontroller GmbH & Co. KG, Hilden / Germany

## 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](mailto:support@segger.com)

Internet: [www.segger.com](http://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: November 12, 2015

Revision	Date	By	Description
0	151109	JL	Initial Version.



# Table of contents

---

1	Integrating SEGGER SystemView into an OS .....	7
2	Instrumenting the OS core .....	9
2.1	SYSTEMVIEW OS Callback API .....	10
2.1.1	pfGetTime .....	10
2.1.2	pfSendTaskList .....	10
2.2	Recording task activity .....	11
2.2.1	Task Create .....	11
2.2.2	Task Start Ready .....	11
2.2.3	Task Start Exec .....	12
2.2.4	Task Stop Ready .....	12
2.2.5	Task Stop Exec .....	12
2.2.6	System Idle .....	13
2.3	Interrupts .....	14
2.3.1	Enter Interrupt .....	14
2.3.2	Exit Interrupt .....	14
2.3.3	Example ISRs .....	14
2.4	Sample pseudo-code OS integration .....	16
3	SEGGER SystemView API functions .....	19
3.0.1	SEGGER_SYSVIEW_EncodeData() .....	21
3.0.2	SEGGER_SYSVIEW_EncodeId() .....	22
3.0.3	SEGGER_SYSVIEW_EncodeString() .....	23
3.0.4	SEGGER_SYSVIEW_EncodeU32() .....	24
3.0.5	SEGGER_SYSVIEW_Init() .....	25
3.0.6	SEGGER_SYSVIEW_NameResource() .....	26
3.0.7	SEGGER_SYSVIEW_OnIdle() .....	27
3.0.8	SEGGER_SYSVIEW_OnTaskCreate() .....	28
3.0.9	SEGGER_SYSVIEW_OnTaskStartExec() .....	29
3.0.10	SEGGER_SYSVIEW_OnTaskStartReady() .....	30
3.0.11	SEGGER_SYSVIEW_OnTaskStopExec() .....	31
3.0.12	SEGGER_SYSVIEW_OnTaskStopReady() .....	32
3.0.13	SEGGER_SYSVIEW_RecordEnterISR() .....	33
3.0.14	SEGGER_SYSVIEW_RecordExitISR() .....	34
3.0.15	SEGGER_SYSVIEW_RecordExitISRToScheduler() .....	35
3.0.16	SEGGER_SYSVIEW_RecordSystemtime() .....	36
3.0.17	SEGGER_SYSVIEW_RecordU32() .....	37
3.0.18	SEGGER_SYSVIEW_RecordU32x2() .....	38
3.0.19	SEGGER_SYSVIEW_RecordU32x3() .....	39
3.0.20	SEGGER_SYSVIEW_RecordVoid() .....	40

3.0.21	SEGGER_SYSVIEW_SendPacket()	41
3.0.22	SEGGER_SYSVIEW_SendSysDesc()	42
3.0.23	SEGGER_SYSVIEW_SendTaskInfo()	43

# Chapter 1

## Integrating SEGGER SystemView into an OS

---

This application note describes how to integrate SEGGER SystemView into an (RT)OS to enable monitoring OS and task execution.

SEGGER SystemView is a real-time recording and visualization tool to gain a deep understanding of the runtime behavior of an application, going far beyond what debuggers are offering. This is particularly advantageous when developing and working in complex systems with multiple tasks and interrupts, to analyze problems and to find inefficiencies.

SEGGER SystemView can be used to monitor the execution of an embedded system with an OS. The following sections describe how to integrate SystemView into an OS by instrumenting it with the SYSTEMVIEW module.





# Chapter 2

## Instrumenting the OS core

---

The mandatory part of integrating SystemView into an OS is to instrument the OS core, that is the OS Scheduler and routines which handle task activity and execution.

In order to be able to visualize task execution and context switches with SystemViewer, the SYSTEMVIEW module must record these events, which is done by adding SYSTEMVIEW event recording functions at appropriate code locations in the OS core.

Some OSes already include functionality to instrument the OS for tracing which can be used by SystemView. This could for example be possible by passing a trace API to the OS or by defining trace function macros.

If the OS does not already include this functionality, SYSTEMVIEW can either be included directly in the OS or by adding the methods mentioned above.

The examples in the following sections are pseudo-code to illustrate when to call SYSTEMVIEW functions. The methods mentioned above could be used in these places, too.

## 2.1 SYSTEMVIEW OS Callback API

Upon initialization a pointer to a `SEGGER_SYSVIEW_OS_API` struct with callbacks which are called by SYSTEMVIEW on specific events to retrieve information from the OS is passed to SYSTEMVIEW.

The OS API and its members are optional, but should be passed to get all information in SystemView.

### 2.1.1 pfGetTime

#### Description

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

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

#### Prototype

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

### 2.1.2 pfSendTaskList

#### Description

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

If `pfSendTaskList` is NULL SystemViewer might only get task information of tasks which are newly created while recording. `pfSendTaskList` is called periodically 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.
}
```

## 2.2 Recording task activity

SystemView can record a set of pre-defined system events, which provide the main information of the system and OS activity. The SYSTEMVIEW API provides functions for these events which should be added to the OS when an event happens.

The usual events are:

Event	Description
Task Create	A new task is created.
Task Start Ready	A task is marked as ready to start or resume execution.
Task Start Exec	A task is activated, it starts or resumes execution.
Task Stop Ready	A task is blocked or suspended.
Task Stop Exec	A task stops execution.
System Idle	No task is executing, the system goes into Idle state.

### 2.2.1 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);
}
```

### 2.2.2 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);
    }
}

```

## 2.2.3 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()
}

```

## 2.2.4 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);
}

```

## 2.2.5 Task Stop Exec

A task stops execution.

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();
}

```

## 2.2.6 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 SystemViewer.

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()
}
```

## 2.3 Interrupts

SystemView can record enter interrupt and exit interrupt events. 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 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 SYSVIEW function.

### 2.3.1 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();
}
```

### 2.3.2 Exit Interrupt

When the OS provides a function to inform the OS that interrupt code has executed, to be called at the end 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();
}
```

### 2.3.3 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();
}

```

## 2.4 Sample pseudo-code OS integration

```

/*****
*          (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();  
}
```

# Chapter 3

## SEGGER SystemView API functions

The following functions can be used to integrate SYSTEMVIEW into an OS.

Function	Description
Control and initialization	
<code>SEGGER_SYSVIEW_Init()</code>	Initializes the SYSVIEW module.
<code>SEGGER_SYSVIEW_SendTaskList()</code>	Send all tasks descriptors to the host.
<code>SEGGER_SYSVIEW_SendTaskInfo()</code>	Send a Task Info Packet, containing TaskId for identification, task priority and task name.
<code>SEGGER_SYSVIEW_SendSysDesc()</code>	Send the system description string to the host.
Event recording	
<code>SEGGER_SYSVIEW_RecordVoid()</code>	Formats and sends a SystemView packet with an empty payload.
<code>SEGGER_SYSVIEW_RecordU32()</code>	Formats and sends a SystemView packet containing a single U32 parameter payload.
<code>SEGGER_SYSVIEW_RecordU32x2()</code>	Formats and sends a SystemView packet containing 2 U32 parameter payload.
<code>SEGGER_SYSVIEW_RecordU32x3()</code>	Formats and sends a SystemView packet containing 3 U32 parameter payload.
<code>SEGGER_SYSVIEW_RecordSystime()</code>	Formats and sends a SystemView Systime containing a single U64 or U32 parameter payload.
<code>SEGGER_SYSVIEW_RecordEnterISR()</code>	Format and send an ISR entry event.
<code>SEGGER_SYSVIEW_RecordExitISR()</code>	Format and send an ISR exit event.
<code>SEGGER_SYSVIEW_RecordExitISRToScheduler()</code>	Format and send an ISR exit into scheduler event.
<code>SEGGER_SYSVIEW_OnIdle()</code>	Record an Idle event.
<code>SEGGER_SYSVIEW_OnTaskCreate()</code>	Record a Task Create event.
<code>SEGGER_SYSVIEW_OnTaskStartExec()</code>	Record a Task Start Execution event.
<code>SEGGER_SYSVIEW_OnTaskStopExec()</code>	Record a Task Stop Execution event.
<code>SEGGER_SYSVIEW_OnTaskStartReady()</code>	Record a Task Start Ready event.

Function	Description
<code>SEGGER_SYSVIEW_OnTaskStopReady()</code>	Record a Task Stop Ready event.
<code>SEGGER_SYSVIEW_OnUserStart()</code>	Send a user event start, such as start of a subroutine for profiling.
<code>SEGGER_SYSVIEW_OnUserStop()</code>	Send a user event stop, such as return of a subroutine for profiling.
<code>SEGGER_SYSVIEW_NameResource()</code>	Send the name of a resource to be displayed in SysViewer.
<code>SEGGER_SYSVIEW_SendPacket()</code>	Send an event packet.
Event parameter encoding	
<code>SEGGER_SYSVIEW_EncodeU32()</code>	Encode a U32 in variable-length format.
<code>SEGGER_SYSVIEW_EncodeData()</code>	Encode a byte buffer in variable-length format.
<code>SEGGER_SYSVIEW_EncodeString()</code>	Encode a string in variable-length format.
<code>SEGGER_SYSVIEW_EncodeId()</code>	Encode a 32-bit Id in shrunken variable-length format.

### 3.0.1 SEGGER\_SYSVIEW\_EncodeData()

#### Description

Encode a byte buffer in variable-length format.

#### Prototype

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

#### Parameters

Parameter	Description
<code>pPayload</code>	Pointer to where string will be encoded.
<code>pSrc</code>	Pointer to data buffer to be encoded.
<code>NumBytes</code>	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.

## 3.0.2 SEGGER\_SYSVIEW\_EncodeId()

### Description

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

### Prototype

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

### Parameters

Parameter	Description
<code>pPayload</code>	Pointer to where the <code>Id</code> will be encoded.
<code>Id</code>	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)

### 3.0.3 SEGGER\_SYSVIEW\_EncodeString()

#### Description

Encode a string in variable-length format.

#### Prototype

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

#### Parameters

Parameter	Description
<code>pPayload</code>	Pointer to where string will be encoded.
<code>s</code>	String to encode.
<code>MaxLen</code>	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.

### 3.0.4 SEGGER\_SYSVIEW\_EncodeU32()

#### Description

Encode a U32 in variable-length format.

#### Prototype

```
U8 *SEGGER_SYSVIEW_EncodeU32(U8 * pPayload,  
                             unsigned 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.



### 3.0.5 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
<a href="#">SysFreq</a>	Frequency of timestamp, i.e. CPU core clock frequency.
<a href="#">CPUFreq</a>	CPU core clock frequency.
<a href="#">pOSAPI</a>	Pointer to the API structure for OS-specific functions.
<a href="#">pfSendSysDesc</a>	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.

### 3.0.6 SEGGER\_SYSVIEW\_NameResource()

#### Description

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

#### Prototype

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

#### Parameters

Parameter	Description
<code>ResourceId</code>	Id of the resource to be named. i.e. its address.
<code>sName</code>	Pointer to the resource name. (Max. 128 Bytes)

### 3.0.7 SEGGER\_SYSVIEW\_OnIdle()

#### Description

Record an Idle event.

#### Prototype

```
void SEGGER_SYSVIEW_OnIdle();
```

### 3.0.8 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(unsigned TaskId);
```

#### Parameters

Parameter	Description
TaskId	Task ID of created task.

### 3.0.9 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(unsigned TaskId);
```

#### Parameters

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

### 3.0.10 SEGGER\_SYSVIEW\_OnTaskStartReady()

#### Description

Record a Task Start Ready event.

#### Prototype

```
void SEGGER_SYSVIEW_OnTaskStartReady(unsigned TaskId);
```

#### Parameters

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

### 3.0.11 SEGGER\_SYSVIEW\_OnTaskStopExec()

#### Description

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

#### Prototype

```
void SEGGER_SYSVIEW_OnTaskStopExec( ) ;
```

### 3.0.12 SEGGER\_SYSVIEW\_OnTaskStopReady()

#### Description

Record a Task Stop Ready event.

#### Prototype

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

#### Parameters

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



### 3.0.13 SEGGER\_SYSVIEW\_RecordEnterISR()

#### Description

Format and send an ISR entry event.

#### Prototype

```
void SEGGER_SYSVIEW_RecordEnterISR();
```

#### Additional information

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

### 3.0.14 SEGGER\_SYSVIEW\_RecordExitISR()

#### Description

Format and send an ISR exit event.

#### Prototype

```
void SEGGER_SYSVIEW_RecordExitISR();
```

#### Additional information

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

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

### 3.0.15 SEGGER\_SYSVIEW\_RecordExitISRToScheduler()

#### Description

Format and send an ISR exit into scheduler event.

#### Prototype

```
void SEGGER_SYSVIEW_RecordExitISRToScheduler();
```

#### 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)

### 3.0.16 SEGGER\_SYSVIEW\_RecordSysTime()

#### Description

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

#### Prototype

```
void SEGGER_SYSVIEW_RecordSysTime();
```

### 3.0.17 SEGGER\_SYSVIEW\_RecordU32()

#### Description

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

#### Prototype

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

#### Parameters

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

### 3.0.18 SEGGER\_SYSVIEW\_RecordU32x2()

#### Description

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

#### Prototype

```
void SEGGER_SYSVIEW_RecordU32x2(unsigned 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.

### 3.0.19 SEGGER\_SYSVIEW\_RecordU32x3()

#### Description

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

#### Prototype

```
void SEGGER_SYSVIEW_RecordU32x3(unsigned 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.

### 3.0.20 SEGGER\_SYSVIEW\_RecordVoid()

#### Description

Formats and sends a SystemView packet with an empty payload.

#### Prototype

```
void SEGGER_SYSVIEW_RecordVoid(unsigned EventID);
```

#### Parameters

Parameter	Description
EventID	SystemView event ID.



### 3.0.21 SEGGER\_SYSVIEW\_SendPacket()

#### Description

Send an event packet.

#### Prototype

```
int SEGGER_SYSVIEW_SendPacket(U8      * pPacket,
                               U8      * pPayloadEnd,
                               unsigned EventId);
```

#### Parameters

Parameter	Description
<code>pPacket</code>	Pointer to the start of the packet.
<code>pPayloadEnd</code>	Pointer to the end of the payload. Make sure there are at least 5 bytes free after the payload.
<code>EventId</code>	Id of the event packet.

#### Return value

$\neq 0$  : Success, Message sent.  
 $= 0$  : Buffer full, Message \*NOT\* sent.

## 3.0.22 SEGGER\_SYSVIEW\_SendSysDesc()

### Description

Send the system description string to the host. The system description is used by SysViewer 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 128 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 ','.

### 3.0.23 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
<code>pInfo</code>	Pointer to task information to send.

