Using Event Recorder

The following steps enable the MDK debugger views for static information *and* dynamic events.

For User Code:

1. Add the **Event Recorder** to the project.
2. Optionally **locate the Event Recorder data to uninitialized memory** to avoid overwriting the entries on program reset.
3. Add **Event Annotations** in the C source to be able to stream dynamic event information.
4. Create an **SCVD file** to **Format Event Information** that matches with application code.

For MDK Middleware, Keil RTX5, and FreeRTOS:

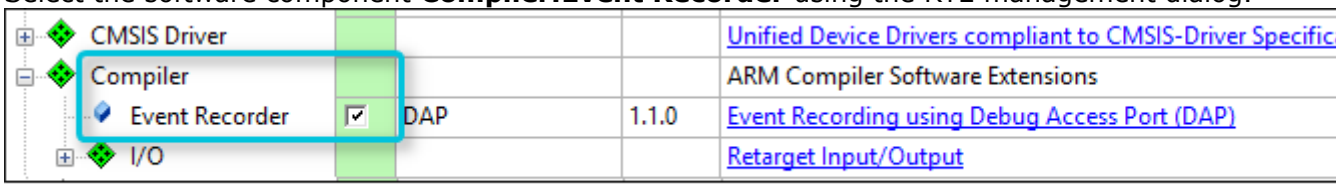
The software packs for MDK Middleware, CMSIS, CMSIS-FreeRTOS already contain the relevant *.SCVD files and the related event annotations in the C source code.

1. **Add Event Recorder** to the project.
2. Select a **Debug** variant for the middleware component to enable event information (for RTX5 select **Source** variant).

Add Event Recorder

To use the Event Recorder in an application, you need to:

- Select the software component **Compiler:Event Recorder** using the RTE management dialog.



- Include the **EventRecorder.h** header file and add the event recorder initialization function to the source code:

```
  :
#include "EventRecorder.h"                        // Keil::Compiler:Event Messaging
  :
int main (void) {
  :
  HAL_Init();                                     // configure hardware abstraction layer
  SystemClock_Config();                           // configure system clock
  MemoryBus_Config();                             // configure external memory bus
  EventRecorderInitialize (EventRecordAll, 1);    // initialize and start Event Recorder
  :
  // other application code
}
```

Note

- By default, the Event Recorder uses the DWT Cycle Counter as a time stamp source. This is not available on Cortex-M0/M0+/M23. Change the **configuration** to use an alternative timer instead.
- For Keil RTX5 (version 5.4.0 and above), no call to **EventRecorderInitialize** is required. Instead enable **Event Recorder Configuration - Global Initialization** in the RTX_Config.h file. Refer to the CMSIS-RTOS2 - RTX v5 Implementation for more information.

Locate Event Recorder in uninitialized memory

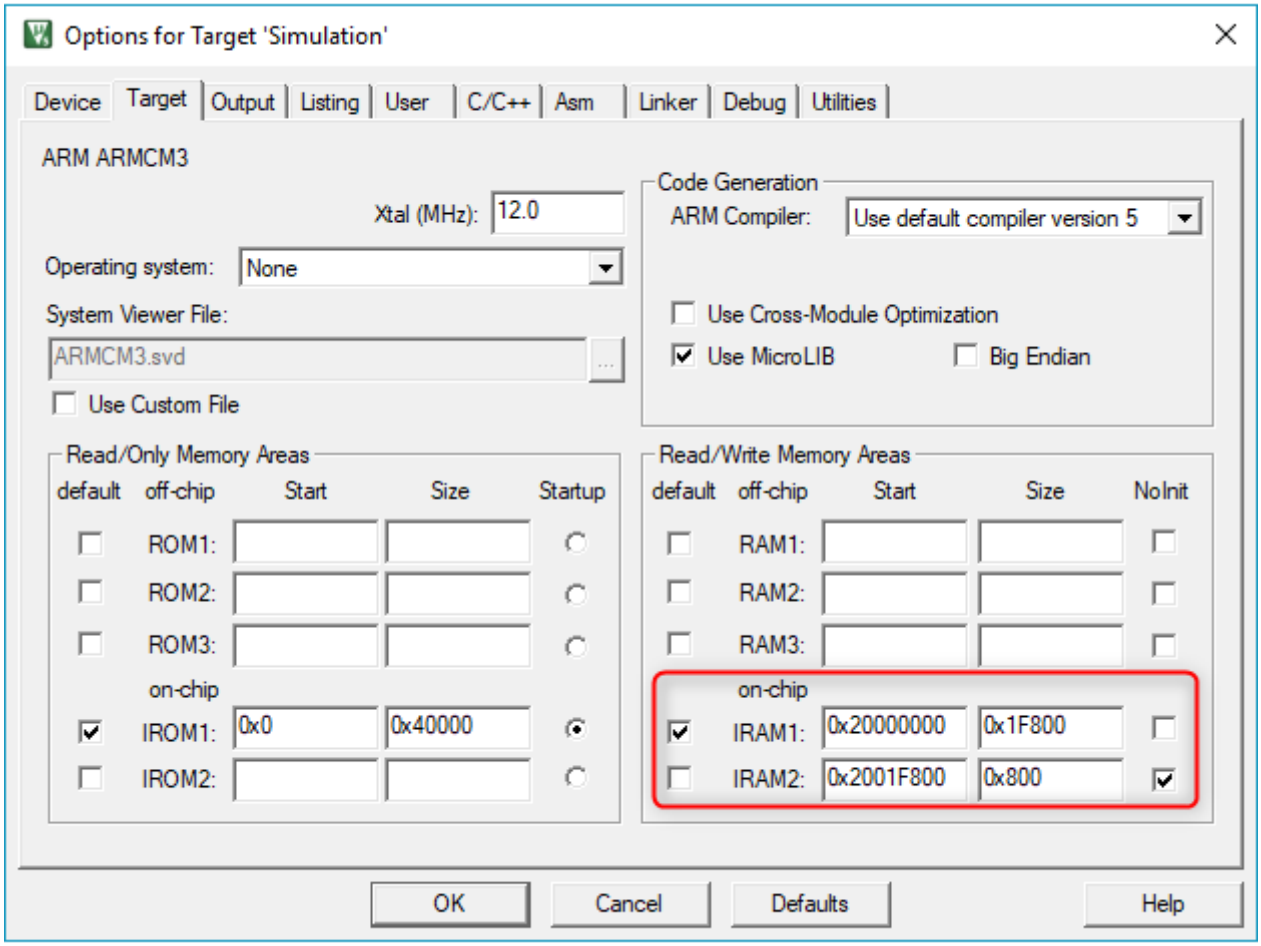
For uninterrupted recording of program resets, the RAM for the Event Recorder component should be placed to a memory region that is not cleared (or initialized) by a system restart (reset). The memory required for the Event Recorder is calculated with the formula:

```
164 + 16 x Number_of_Records (defined by \c EVENT_RECORD_COUNT in EventRecorderConf.h)
```

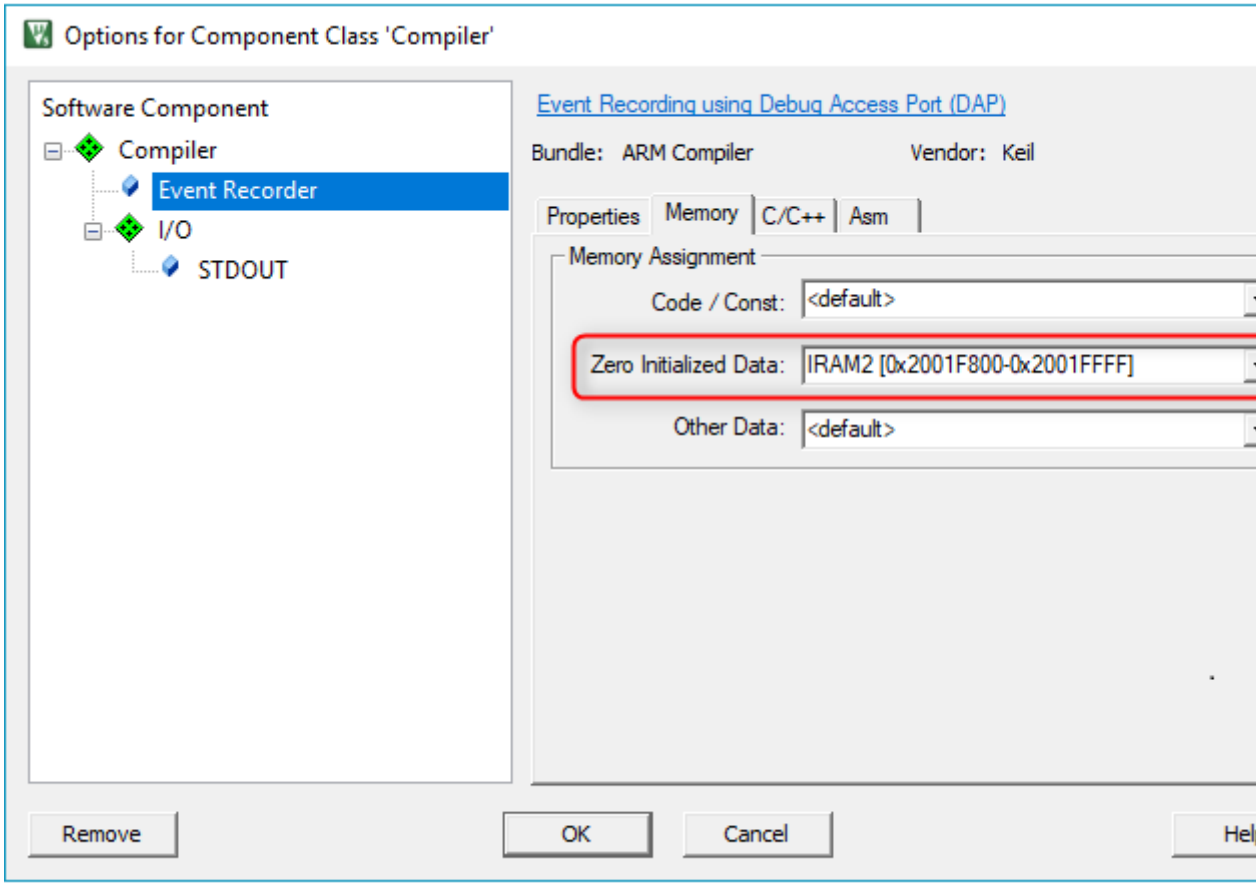
In this example we configure 0x800 bytes (as it is easier) which can more than 64 records.

Use either a linker script, or configure this uninitialized RAM in uVision with these steps:

1. In dialog **Options for Target** define a Read/Write memory area is not initialized; for example split IRAM1 into two regions:
    - Reduce size of IRAM1 by 0x800 and create an IRAM2 area with start 0x2001F800 and size 0x800. Enable **NoInit** for this IRAM2 region.



2. In dialog **Options for Component Class 'Compiler'** (opens with right-click on **EventRecorder.c** in the **Project** window):
    - Under the **Memory** assign **Zero Initialized Data** to the IRAM2 region.



3. Build the application to place the Event Recorder data buffers to uninitialized RAM. You may verify the generated scatter file:

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************
LR_IROM1 0x00000000 0x00040000  {    ; load region size_region
  ER_IROM1 0x00000000 0x00040000  {  ; load address = execution address
    .o (RESET, +First)
    (InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x0001F800  {  ; RW data
   .ANY (+RW +ZI)
  }
  RW_IRAM2 0x2001F800 UNINIT 0x00000800  {
    EventRecorder.o (+ZI)
  }
}
```

Note

- If the Event Recorder data buffer is not in uninitialized memory, the **Command** window of the debugger displays: "Warning: Event Recorder not located in uninitialized memory!".
- Knowledgebase article 4012 explains how to create custom memory areas in uVision.

Event Annotations

To to stream dynamic event information, insert calls to the **Event Data Recording** functions on relevant code locations:

- **EventRecordData** to record a data field with flexible length.
- **EventRecord2** to record up to two 32-bit integer values.
- **EventRecord4** to record up to four 32-bit integer values.

These **Event Data Recording** functions receive as first parameter an *id* event identifier used for filtering and displaying. The macro **EventID** may be used to compose *id* values to include `level` and `component` numbers.

Example:

```
#include "EventRecorder.h"                          // Keil::Compiler:Event Messaging

int some_error = 0;                                 // error flag
char string[10] = "MyTest";                         // some test string

void MyFunction (int parameter) {
  EventRecord2 (1+EventLevelAPI, parameter, 0);              // Event at Start
    ;
  if (some_error)  {
    EventRecord2 (2+EventLevelError, 0, 0);                  // Event at Error
    return;
  }
  EventRecordData (3+EventLevelOp, string, sizeof(string));  // Event at Finish
  return;
}

int main (void) {
  EventRecorderInitialize (EventRecordAll, 1);  // initialize and start Event Recorder

  MyFunction (0x10);
  some_error = 1;                               // set error flag
  MyFunction (0x60);
}
```

When executing this example in the µVision debugger, use the menu command **View - Analysis Windows - Event Recorder** to open the Event Recorder window. This should show the following output:

Event Recorder

O: All Operations | All Components | Find: | Stopped

Event	Time (sec)	Type	Property	Value
0	4.10822950		id=0xFF00	0x00000000,0x00000000
1	4.10823140		id=0x0001	0x00000010,0x00000000
2	4.10823350		id=0x0003	0x6554794D,0x00007473,0x00000000,0x00000000
3	4.10823690		id=0x0001	0x00000060,0x00000000
4	4.10823890		id=0x0002	0x00000000,0x00000000

Output shown in Event Recorder window

Format Event Information

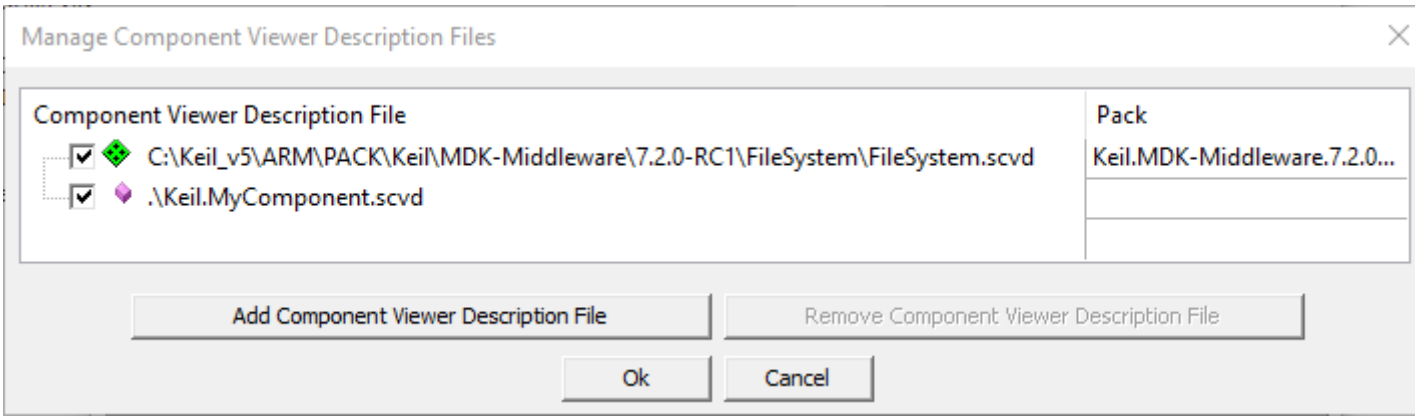
You may create an ***.SCVD (Software Component View Description) file** to format the event output so that matches the application. The event output is created using the **/component_viewer/events**.

SCVD file example

```
<?xml version="1.0" encoding="utf-8"?>

<component_viewer schemaVersion="0.1" xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
        xs:noNamespaceSchemaLocation="Component_Viewer.xsd">
  <component name="MyExample" version="1.0.0"/>    <!-- name and version of the component  -->

    <events>
      <group name="My Events Group">
         <component name="MyApp"      brief="My Application"    no="0x00" prefix="EvrNetMM_"
        info="Network - System - Dynamic Memory Management"/>
      </group>

      <event id="1" level="API"   property="MyFunction"        value="parameter=%x[val1]"
        info="Event on start of MyFunction"  />
      <event id="2" level="Error" property="MyFunctionError"
        info="Event on error in MyFunction" />
      <event id="3" level="Op"    property="MyFunctionProcess" value="string=%t[val1]"
        info="Event on operation in MyFunction"  />
    </events>

</component_viewer>
```

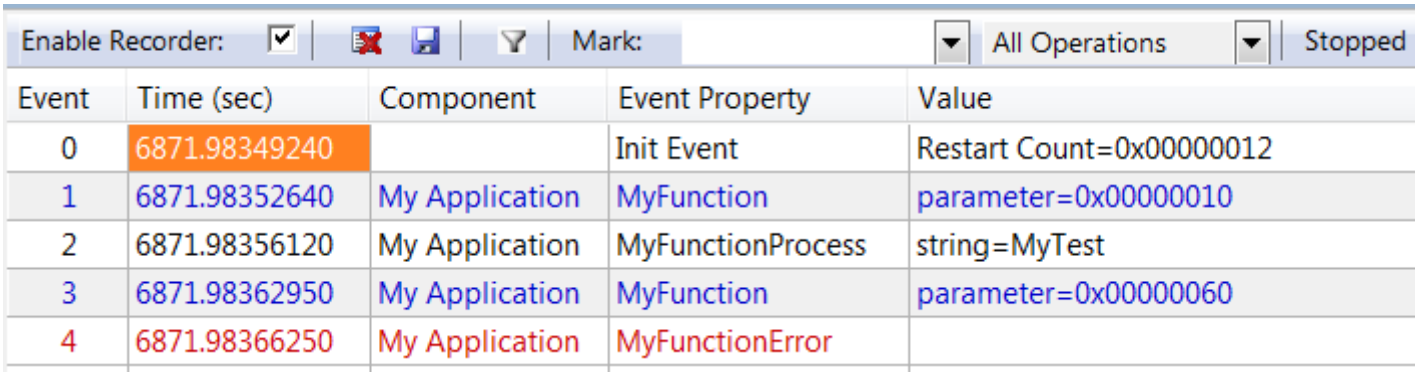
In the µVision debugger, this ***.SCVD file** is specified in the dialog **Options for Target -> Debug -> Manage Component Viewer Description Files**. Click on **Add Component Viewer Description File**

and add the related .SCVD file.



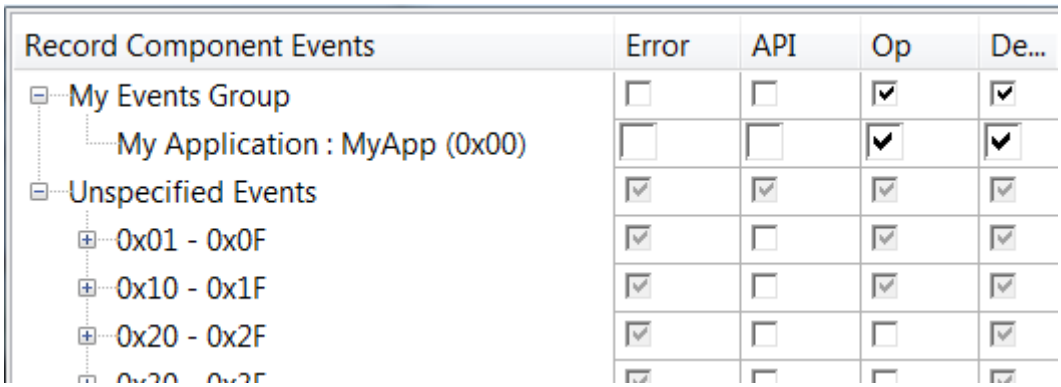
Manage *.SCVD files

The Event Recorder displays the events as shown below.



Event Recorder output formatted with *.SCVD file

The described groups and events also show up in the filter dialog.

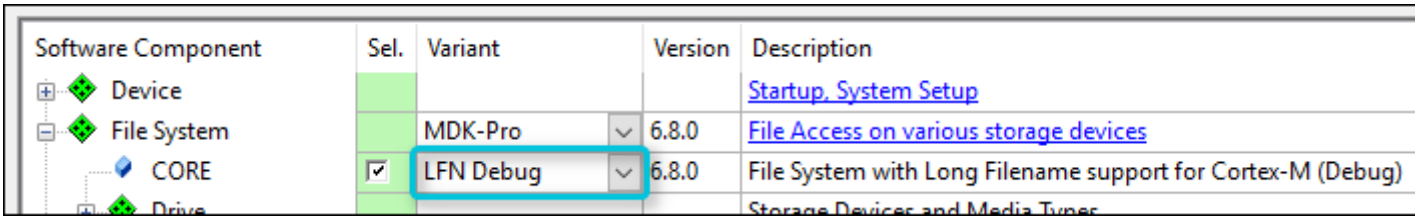


Event Recorder Filter dialog

Software Component Variants

The software packs for MDK Middleware and CMSIS already contain SCVD files that match the related event annotations in the C source code. However, you need to select the right component **Variant**. For MDK Middleware, you need to select the **Debug** variants, whereas for Keil RTX5, you need to add the **Source** variant.

The example below enables event recording for the MDK-Middleware **File System** component:



Redirecting printf output

The Event Recorder can be used to retarget printf output. This is especially interesting for targets without ITM, such as Cortex-M0/M0+/M23. Steps to enable this:

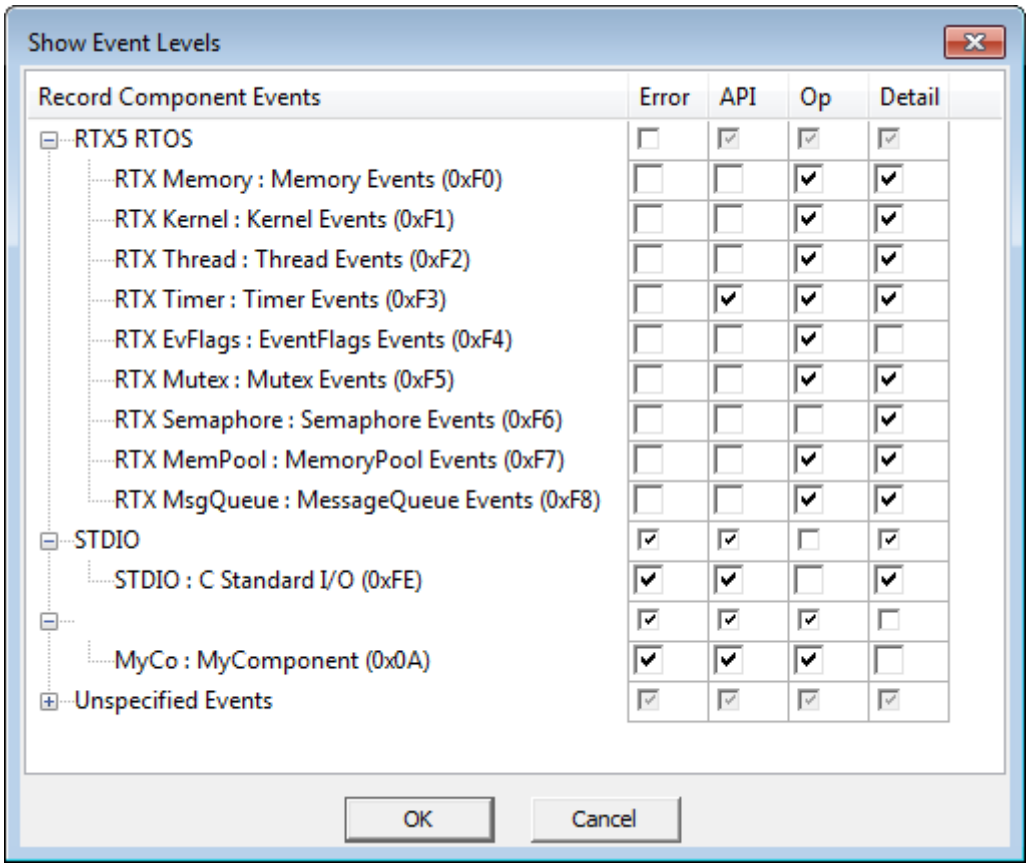
1. In the Manage Run-Time Environment window, set the component **Compiler:I/O:STDOUT** to use **Variant** *EVR*.
2. Select the component **Compiler:Event Recorder** or use the **Resolve** button.
3. In the user code, include `EventRecorder.h` and call the `EventRecorderInitialize()` function in main().

Refer to the example **Retarget STDOUT via Event Recorder** in "I/O Retargeting".

Event filtering

Filtering for events reduces the amount of data transmitted from the target to the debugger. To filter for events, use the button **Configure Target Event Recording**:

A new window opens up that lets you filter for events that you are interested in:



Filtering events